Applikationen objektorientiert konzipieren | OAOD

Objektorientierung 4
- Definition Objektorientierung 4
- Vorteile Objektorientierung 4
- Risiken Objektorientierung 4
- Vorgehensmodelle 4

Objektorientierte Systementwicklung 5
- OO-Vorgehensmodell 5
- Ergebnisse Anforderungsspezifikation 5
- Architektur-Sichten 5
- MVC-Schichtenmodell 5

Objektorientierte Analyse 6
- Definition OO-Analyse 6
- Tätigkeiten OO-Analyse 6
- RAD 6
- CRC 6
- Notation und Beispiel CRC-Karte 6
- Ergebnisse OO-Analyse 6
- Analyse-Modell 6

Objektorientiertes Design 7
- Definition OO-Design 7
- Tätigkeiten OO-Design 7
- OO-Entwurfsprinzipien 7
- Ergebnisse OO-Design 7

OO-Techniken 8
- OO-Techniken 8
- MDA 8
- UML 8
- Extreme Programming XP 8
- XP-Bereiche 8

Rational Unified Process RUP 9
- Definition RUP 9
- Aufbau RUP 9
- Grafik RUP 9
- Workflows 9
- Inception 9
- Elaboration 9
- Construction 9
- Transition 9

UCP 10
- Definition UCP 10
- UUCP 10
- TCF 10
- ECF 10
- Aufwand 10

UML 11
- Geschichte der UML 11
- Basis-Elemente 11
- Extensions-Elemente 11
- Statische Beziehungselemente 11
- Dynamische Beziehungselemente 11
- Strukturdiagramme (structural diagrams) 11
- Verhaltensdiagramme (behaviour diagrams) 11

Basis-Elemente 12
- Objekt 12
- Abstraktion 12
- Klasse 12
- Kapselung (encapsulation) 12
- Attribut 12
- Operation 12
- Sichtbarkeit 12

Notation Basis-Elemente 13
- Notation Klasse 13

Regeln Klasse ... 13
Beispiele Klasse ... 13
Spezialklassen ... 13
Notation Objekt ... 13
Regeln Objekt ... 13
Beispiel Objekt ... 13

Extensions-Elemente ... 14
Zusicherung (constraint) ... 14
Merkmal (tagged value) ... 14
Stereotyp ... 14
Stereotyp-Kategorien ... 14
Notiz ... 14
Entwurfsmuster ... 14

Statische Beziehungselemente ... 15
Vererbung (inheritance) ... 15
Diskriminator ... 15
Assoziation (association) ... 15
Referentielle Integrität ... 15
Aggregation (aggregation) ... 15
Komposition (composite) ... 15
Abhängigkeit (dependency) ... 15
Verfeinerung (refinement) ... 15

Notation Vererbung ... 16
Einzel-Notation Vererbung ... 16
Regeln Vererbung ... 16
Gemeinsame Notation Vererbung ... 16
Beispiel Vererbung ... 16

Notation Assoziation ... 17
Notation Assoziation ... 17
Regeln Assoziation ... 17
Beispiele Assoziation ... 17
Rekursive Assoziation ... 17
Attributierte Assoziation ... 17
Mehrgliedrige Assoziation ... 17
Aggregation ... 17
Komposition ... 17

Dynamische Beziehungselemente ... 18
Nachricht (message) ... 18
Polymorphismus ... 18

Klassendiagramm ... 19
Definition Klassendiagramm ... 19
Elemente Klassendiagramm ... 19
Notation Klassendiagramm ... 19
Schnittstelle ... 19

Komponentendiagramm ... 20
Definition Komponentendiagramm ... 20
Elemente Komponentendiagramm ... 20
Notation Komponentendiagramm ... 20
Regeln Komponentendiagramm ... 20
Abhängigkeiten im Komponentendiagramm ... 20
Beispiel Komponente ... 20

Kompositionsstrukturdiagramm ... 21
Definition Kompositionsstrukturdiagramm ... 21
Elemente Kompositionsstrukturdiagramm ... 21
Notation Kompositionsstrukturdiagramm ... 21
Beispiel Kompositionsstrukturdiagramm ... 21

Verteilungsdiagramm ... 22
Definition Verteilungsdiagramm ... 22
Elemente Verteilungsdiagramm ... 22
Notation Verteilungsdiagramm ... 22
Regeln Verteilungsdiagramm ... 22
Beispiel Verteilungsdiagramm ... 22

Objektdiagramm ... 23
Definition Objektdiagramm ... 23
Elemente Objektdiagramm ... 23
Beispiel Objektdiagramm ... 23

Abhängigkeiten im Objektdiagramm ... 23

Paketdiagramm ... 24
Definition Paket ... 24
Definition Paketdiagramm ... 24
Elemente Paketdiagramm ... 24
Notation Paketdiagramm ... 24
Abhängigkeiten im Paketdiagramm ... 24

Use Case ... 25
Definition Use Case ... 25
Darstellungsarten ... 25
Use Cases in Tabellenform ... 25

Anwendungsfalldiagramm ... 26
Definition Anwendungsfalldiagramm ... 26
Elemente Anwendungsfalldiagramm ... 26
Notation Anwendungsfalldiagramm ... 26
Regeln Anwendungsfalldiagramm ... 26
Beispiel Anwendungsfalldiagramm ... 26
Abhängigkeiten im Anwendungsfalldiagramm ... 26

Aktivitätsdiagramm ... 27
Definition Aktivitätsdiagramm ... 27
Elemente Aktivitätsdiagramm ... 27
Control Nodes ... 27
Spezielle Knoten ... 27
Notation Aktivitätsdiagramm ... 27
Strukturierte Knoten ... 27
Aktivitätsbereiche ... 27

Aktivitätsdiagramm Beispiele ... 28
Mini-Beispiele ... 28
Seminar durchführen ... 28
Strukturierte Knoten ... 28
Aktivitätsbereiche ... 28

Zustandsdiagramm ... 29
Definition Zustandsdiagramm ... 29
Elemente Zustandsdiagramm ... 29
Notation Zustandsdiagramm ... 29
Regeln Zustandsdiagramm ... 29
Beispiele Zustandsübergänge ... 29
Zustandsinterne Ereignisse ... 29
Beispiel 1 Zustandsdiagramm ... 29
Beispiel 2 Zustandsdiagramm ... 29

Sequenzdiagramm ... 30
Definition Sequenzdiagramm ... 30
Elemente Sequenzdiagramm ... 30
Notation Sequenzdiagramm ... 30
Regeln Sequenzdiagramm ... 30
Beispiel Sequenzdiagramm ... 30
Fragmente ... 30

Kommunikationsdiagramm ... 31
Definition Kommunikationsdiagramm ... 31
Elemente Kommunikationsdiagramm ... 31
Notation Kommunikationsdiagramm ... 31
Regeln Kommunikationsdiagramm ... 31
Beispiel Kommunikationsdiagramm ... 31
Nachrichtenarten ... 31

Interaktionsüberblicksdiagramm ... 32
Definition Interaktionsüberblicksdiagramm ... 32
Elemente Interationsüberblicksdiagramm ... 32
Notation Interaktionsüberblicksdiagramm ... 32

Zeitverlaufsdiagramm ... 33
Definition Zeitverlaufsdiagramm ... 33
Elemente Zeitverlaufsdiagramm ... 33
Notation Zeitverlaufsdiagramm ... 33
Beispiel Zeitverlaufsdiagramm ... 33

Definition Objektorientierung

Die Gegenstände der realen Welt werden als Objekte angesehen und auf die in der jeweiligen Situation bedeutsamen Eigenschaften reduziert (Abstraktion).

Systeme werden als miteinander kommunizierende, zustandsbehaftete Einzelkomponenten betrachtet.

Vorteile Objektorientierung

- ganzheitliche, antropomorphe Sichtweise:
  - bessere Kommunikationsbasis Entwickler / Anwender
  - Kunde ist involviert, ständige Optimierung
- evolutionäre Entwicklung:
  - kurze Reaktionszeiten, einfache Integration neuer Anf.
  - Erschliessung komplexer Anwendungsgebiete
  - hohe Motivation durch nahes Ziel
- durchgängige Modellierung:
  - konsistente Qualität, stabile Modelle
  - bessere Dokumentation
- Wiederverwendung:
  - kürzere Entwicklung, geringerer Wartungsaufwand
  - höhere Zuverlässigkeit
  - Vorbild- und Ausbildungseffekt

Risiken Objektorientierung

- wenig Erfahrung
- erfordert viel Flexibilität
- Anforderungen ändern immer wieder
- nicht-funktionale Anforderungen wenig integriert
- Überschätzung von Einzelaspekten, z.B. Vererbung, und deren exzessive Verwendung
- Überspezifikation
- bei kleinen Projekten Overkill
- unausgereifte Werkzeuge
- schlechtere Performance, hoher Ressourcenverbrauch
- Strukturbruch bei relationalen Datenbanken
- mehr Code

Vorgehensmodelle

Klassische:
- strukturierte Programmierung
- statisch
- dokumentorientiert
- teuer
- für grosse Projekte

Agile:
- objektorientierte Programmierung
- flexibel
- kundenorientiert
- chaotisch
- für kleine Projekte und Teams

OO-Vorgehensmodell

Evolutionäres, iteratives, spiralförmiges Vorgehensmodell:
- Anforderungsspezifikation: System als Blackbox, Anforderungen aus Sicht des Benutzers
- OOA: System als White Box, detaillierte Analyse eines Teilbereichs des Systems
- OOD: Vorlage für die Programmierung
- OOP: Realisierung dieses Teilsystems
- operativer Einsatz
- Ergebnisprüfung

Ergebnisse Anforderungsspezifikation

- Funktionalität mit Uses Cases
  - zunächst konkrete Beispielszenarien
  - danach allgemeine Typszenarien
- Überblick mit Bereichsmodell, vereinfachtes Klassendiagramm ohne Attribute / Operationen
- Prototyping

Architektur-Sichten

- Logische Sicht (Endanwender): Funktionalität
- Implementierungssicht (Programmierer): Software-Management
- Prozess-Sicht (System-Integrator): Performanz, Skalierbarkeit
- Verteilungssicht (System-Ingenieur): Installation, Kommunikation
- Use Case-Sicht (Analyst / Tester): Verhalten

MVC-Schichtenmodell

- model: Geschäftslogik
- controller: Applikationslogik
- view: Präsentationslogik

Definition OO-Analyse

Erforschen des Anwendungsbereichs.

- Objekte entdecken
- Klassen ableiten
- Klassen strukturieren
- Aufgaben zuordnen
- Zusammenarbeit festlegen

Tätigkeiten OO-Analyse

- Vorstudie: Zielsetzungen, Lösungsvarianten
- Geschäftsprozessanalyse: Systemumfeld
- RAD-Workshop
- Anwendungsfallanalyse
- Fachlexikon / Glossar
- explorativ Prototyping, Dialogentwürfe
- CRC-Karten
- Businessklassen: fachliche Objekte in grober Granularität

RAD

Rapid Analysis and Design

Initial-Workshop zur Erforschung der grundlegenden Systemanforderungen.

Basis für Aufwandschätzung und Planung.

Teilnehmer: Entwickler, Anwender, Auftraggeber

CRC

Class – Responsibilities – Collaborators

Technik, um gemeinsam mit dem Fachbereich die fachlichen Anforderungen zu modellieren.

Auf Karten mit T-Layout (CRC-Karten, Klassenkarten) werden der Klassenname, die Verantwortlichkeiten (abstrakte Beschreibung von Aufgaben) und die Beteiligten (Zusammenarbeit mit anderen Klassen) festgehalten.

Zweck:
- Findung und Festlegung der richtigen Klassen
- Klärung der Struktur

Notation und Beispiel CRC-Karte

Klasse	
Verantwort.1	Beteiligter1
Verantwort.2	Beteiligter2
Verantwort.3	Beteiligter3

Bestellung	
Lagervorrat prüfen	Lagerbestand
Preis bestimmen	Kunde
Zahlung prüfen	
Ausliefern	

Ergebnisse OO-Analyse

Analyse-Modell: Diagramme und Beschreibungen aus der OO-Analyse.

Merkmale:
- leicht verständlich
- nachvollziehbar
- anpassbar und erweiterbar

Zweck:
- Vorlage für OO-Design
- Grundlage für Systemtest

Analyse-Modell

statische Struktur des Systems:
- Paketdiagramm: Klassendiagramm auf höherer Ebene
- Klassendiagramm: pro Paket

dynamischer Teil des Systems:
- Interaktionsdiagramm: Nachrichtenaustausch, Interaktion
  - Kommunikationsdiagramm: Zusammenarbeit
  - Sequenzdiagramm: zeitlicher Ablauf
- Zustandsdiagramm: Lebenslauf eines komplexen Objekts

Objektorientiertes Design

OAOD

Definition OO-Design

Ergänzen der Strukturen aufgrund technischer Erfordernisse.

Tätigkeiten OO-Design

- Anwendungsarchitektur festlegen
- Anwendungsbausteine: enthalten Dialogsteuerung, Interaktionssteuerung, Vorgangssteuerung, Businessklassen, Datenhaltung und Schnittstellen
- Fachklassen und Beziehungen identifizieren
- Operationen spezifizieren
- Attribute spezifizieren
- Aktivitäten modellieren
- Zustände modellieren
- Objektinteraktionen modellieren
- Datenbankanbindung
- experimentelles Prototyping

OO-Entwurfsprinzipien

- Substitutionsprinzip:
  Instanzen einer Unterklasse können als Instanzen ihrer Oberklasse angesprochen werden (Polymorphismus)
- Programmieren gegen Schnittstellen:
  Operationsaufrufe erfolgen an definierte Schnittstellen (Kapselung)
- Objektkomposition vor Klassenvererbung:
  Verantwortlichkeit verteilen und bei Bedarf dynamische Objektzusammenarbeit (Delegation)
- Entkoppelung variabler von konstanten Teilen:
  sich ändernde Teile hinter konstanten Schnittstellen verbergen (information hiding)

Ergebnisse OO-Design

Verfeinerung des Analyse-Modells sowie weitere Diagramme.

logische Struktur des Systems:
- Design Patterns: Programmier-Muster
- Aktivitätsdiagramm: interne Abläufe eines Objekts

physische Struktur des Systems:
- Komponentendiagramm: Schichten-Architektur
- Verteilungsdiagramm: physische Komponenten

OO-Techniken

- RUP
- UCP
- MDA
- UML
- Agile Softwareentwicklung
- Extreme Programming

MDA

Model Driven Architecture

Vorgehensweise beim Softwareentwicklungsprozess unter Verwendung von UML.

- Platform Independent Model PIM
- Platform Specific Model PSM

www.omg.org/mda

UML

Unified Modeling Language

Sprache und Notation zur Spezifikation, Konstruktion (Forward- und Reverse-Engineering), Visualisierung und Dokumentation von Modellen für Softwaresysteme.

keine Methode / Vorgehensweise

www.uml.org
http://www.oose.de/uml.htm
http://www.jeckle.de/uml.de/
http://www.jeckle.de/vorlesung/uml/index.html

Extreme Programming XP

Leichte, effiziente, risikoarme, flexible, kalkulierbare, exakte und vergnügliche Art und Weise der Softwareentwicklung, entwickelt 1999 von Kent Beck.

- Programmierung in kleinen Teilen (Puzzle-Prog.)
- Kunde ist Teil des Projektteams
- konsequente Orientierung an Kundenanforderungen
- kurze Release-Zeiten
- iteratives Vorgehen
- integriertes Testen

XP-Bereiche

- Planen:
  - Ausarbeitung der Anforderungen in “Stories”
  - Einteilen des Projekts in Iterationen
- Kodieren:
  - Code-Entwicklung nach vereinbarten Standards
  - Paar-Programmierung
- Modellieren:
  - einfache und einheitliche Strukturierung
  - Refactoring: Verbesserung der Lesbar- und Testbarkeit
- Testen:
  - Testen jedes Codefragments
  - Veröffentlichung der Ergebnisse von Akzeptanztests

Rational Unified Process RUP

OAOD

Definition RUP

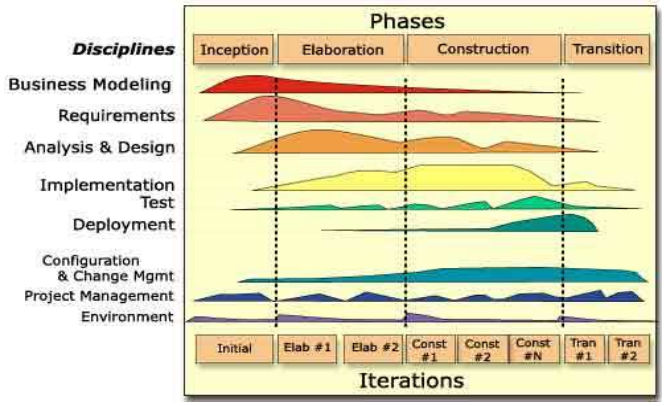
OO-Vorgehensweise zur Entwicklung von Software, entwickelt 1998 durch die Firma Rational.

- Rahmenwerk für den gesamten Entwicklungsprozess
- Entwicklung in kleinen, zeitgesteuerten Iterationen
- Entwicklung in Zweiergruppen (4-Augen-Prinzip)
- anwendungsfall-gesteuerte Softwareentwicklung
- architekturzentrierter Prozess
- nach jeder Phase Meilensteine mit definiertem Ergebnis

Aufbau RUP

- System-Lebenslauf: besteht aus Zyklen
- Zyklus: besteht aus 4 Phasen
  führt zu auslieferbarer Produkt-Version
- Phase: Fortschritt im Fertigstellungsgrad
  besteht aus Kombination von Workflows
  kann aus mehreren Inkrementen bestehen
- Inkrement: führt zu internem Release

Grafik RUP



Workflows

Name	Beschreibung	Ergebnis
Business Modeling	Geschäftsanford. ausarbeiten	Glossar
Requirements	In Systemanford. umwandeln	Use Cases
Analysis & Design	SW-Architektur definieren	Klassendiag.
Implementation	Anforderungen umsetzen	SW-Code
Test	Anforderungen überprüfen	Testfälle
Deployment	SW verteilen	Produkt
Change & Config.	Versionen / Abhängigkeiten	RfC, CMDB
Project Mgmt	Termine und Ressourcen	Risks, Pläne
Environment	Entwicklungsumgebung	Werkzeuge

Inception

Etablierungsphase, Einstieg, Konzeptualisierung

- Ausprobieren verschiedener Varianten
- Erstellen von einfachen Wegwerf-Prototypen
- Spezifikation der Kernanforderungen und der hauptsächlichen Rahmenbedingungen
- Spezifikation der Endproduktvision
- Definition des Projektumfangs
- Festlegen der Erfolgskriterien
- Abschätzung der Kosten und Risiken
- Erstellen eines groben Zeit- und Ressourcenplans

Abschluss: Life Cycle Objective Milestone

Elaboration

Entwurfsphase, Ausarbeitung

- Planung der notwendigen Aktivitäten und Ressourcen
- Entwickler: Aufwandschätzung anhand bisheriger Ergebnisse (load factor)
- Auftraggeber: Priorisierung
- Design der Architektur
- Spezifikation der Produkteigenschaften
- Überprüfung und evtl. Implementation der wichtigsten Anforderungen mit Testfällen
- Eliminierung hoher Risiken

Abschluss: Life Cycle Architecture Milestone

Construction

Konstruktionsphase, Implementierung

- Entwicklung der Architektur
- Erstellung von Testfällen
- Erstellung des Produkts
- Erstellung von Benutzerhandbüchern
- Ergebnis: brauchbares Produkt (Beta Release)

Abschluss: Initial Operational Capability Milestone

Transition

Übergangsphase, Produktübergabe

- möglichst frühe Freigabe des Produkts an die Benutzer
- Hinzufügen weiterer Features
- Überprüfung des Qualitätslevels
- Anwender-Training
- Einsatzunterstützung
- Wartung
- Marketing und Distributionsaktivitäten

Abschluss: Product Release Milestone

Definition UCP

Use Case Points, Methode zur Aufwandschätzung.

UCP = UUCP * TCF * ECF

UUCP Unadjusted Use Case Points
TCF Technical Complexity Factor
ECF Environment Complexity Factor

UUCP

UUCP = UAW + UUCW

UAW Unadjusted Actor Weight
UUCW Unadjusted Use Case Weight

Actors und Use Cases nach Gewicht einteilen und mit den entsprechenden Punkten multiplizieren:

Actors: einfach=1, mittel=2, komplex=3
Use Cases: Anzahl Transaktionen 1-3=5, 4-7=10, >7=15

TCF

TCF = 0.6 + (0.01 * TF)

Offizielle Tabelle mit 13 Faktoren zur technischen Komplexität mit Gewichten von 0.5 bis 2.

TF: je multiplizieren mit Projekt-Wichtigkeitsfaktor von 0=irrelevant bis 5=sehr wesentlich

ECF

ECF = 1.4 – (0.03 * EF)

Offizielle Tabelle mit 8 Faktoren zur Erfahrung des Teams mit Gewichten von -1 bis 2.

EF: je multiplizieren mit Projekt-Wichtigkeitsfaktor von 0=irrelevant bis 5=sehr wesentlich

Aufwand

Aufwand in Stunden = UCP * 20 Personen-Stunden

Aufwand in Jahren = UCP * 20 / (40 Stunden * 50 Wochen)
= UCP / 100

Geschichte der UML

1995 Unified Method 0.8, Object Modeling Technique OMT von Jim Rumbaugh und OOAD von Grady Booch
1996 UML 0.9, Use Cases von Ivar Jacobson (“Amigos”)
1997 UML 1.0 bei Object Management Group OMG zur Standardisierung eingereicht
1997 UML 1.1, Integration Object Constraint Language OCL
1998 UML 1.2
1999 UML 1.3, XML Metadata Interchange
2001 UML 1.4
2003 UML 1.5
2004 UML 2.0, Erfahrung der Anwender

Basis-Elemente

- Objekt
- Klasse
- Attribut
- Operation

Extensions-Elemente

- Zusicherung
- Merkmal
- Stereotyp
- Notiz
- Entwurfsmuster

Statische Beziehungselemente

Beziehungen zwischen Klassen.

- Vererbung
- Assoziation
- Aggregation
- Komposition
- Abhängigkeit
- Verfeinerung

Dynamische Beziehungselemente

Beziehungen zwischen Objekten.

- Nachricht
- Polymorphismus

Strukturdiagramme (structural diagrams)

- Klassendiagramm (class diagram)
- Objektdiagramm (instances diagram, früher object diagram)
- Paketdiagramm (package diagram)
- Komponentendiagramm (component diagram)
- Kompositionsstrukturdiagramm (composite structure)
- Verteilungsdiagramm (deployment diagram) früher Einsatzdiagramm

Verhaltensdiagramme (behaviour diagrams)

- Anwendungsfalldiagramm (use case diagram)
- Aktivitätsdiagramm (activity diagram)
- Zustandsdiagramm (statechart diagram)
- Interaktionsdiagramm (interaction diagram) (zeigt Nachrichtenfluss):
  - Sequenzdiagramm (sequence diagram)
  - Kommunikationsdiagramm (communication diagram) früher Kollaborationsdiagramm (collaboration diagram)
  - Interaktionsüberblicksdiagramm (interaction overview)
  - Zeitverlaufsdiagram (timing diagram)

Basis-Elemente

Objekt

Abstraktion der realen Welt.
Konkrete Ausprägung / Instanz / Implementation einer Klasse.
Eine zur Ausführungszeit vorhandene und für ihre Instanzvariablen Speicher allozierende Instanz, die sich entsprechend dem Protokoll ihrer Klasse verhält.
Zustand und Verhalten wird im Objekt gekapselt.

Kennzeichen:
- Identität: macht es einzigartig
- Zustand: wird durch die Werte der Attribute bestimmt
- Verhalten: wird durch Methoden beschrieben

Abstraktion

Methode, bei der unter einem bestimmten Gesichtspunkt die wesentlichen Merkmale eines Objekts herausgefiltert werden (Verallgemeinerung).

Zweck:
- besseres Verständnis der wesentlichen Elemente
- ermöglicht Modellbau

Klasse

Bauplan einer Abstraktion.
Beschreibt die mittels Attributen die gemeinsame Datenstruktur und mittels Operationen das Verhalten einer Menge gleichartiger Objekte.

Bestandteile:
- Attribute
- Operation

Kapselung (encapsulation)

Verbergen von Implementierungsdetails (Geheimnis-Prinzip, information hiding). Der direkte Zugriff auf die interne Datenstruktur wird unterbunden und erfolgt statt dessen über definierte Schnittstellen.

Vorteile:
- erhöhte Übersichtlichkeit
- der Aufrufer muss nur die Schnittstelle kennen
- Änderungen können lokal gehalten werden
- Stabilität, weniger Fehler
- Flexibilität, rasche Umstellungen
- bessere Testbarkeit

Attribut

Attribute definieren die Struktur eines Objekts.
Attributwerte enthalten die Daten des Objekts.

Operation

Operationen definieren das Verhalten eines Objekts.

Signatur:
beschreibt die Operation: Name, Datentypen der Parameter, Datentyp des Rückgabewerts

Schnittstelle:
Signaturen der als öffentlich deklarierten Operationen

Methode:
implementiert die Operation: Sequenz von Anweisungen

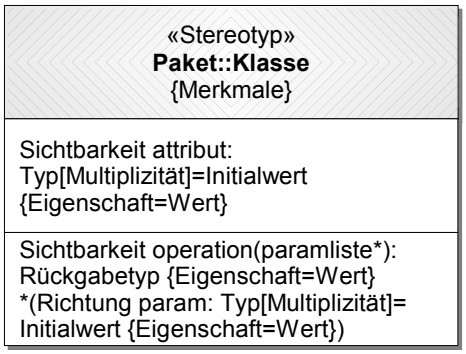
Sichtbarkeit

Zugreifbarkeit von Attributen und Operationen.
Darstellung: Kennzeichen direkt vor dem Namen

- +public: für alle Klassen sichtbar und benutzbar
- ~package: für alle Klassen im selben Paket
- #protected: für die Klasse selbst und ihre Unterklassen
- –private: nur für die Klasse selbst

Notation Basis-Elemente

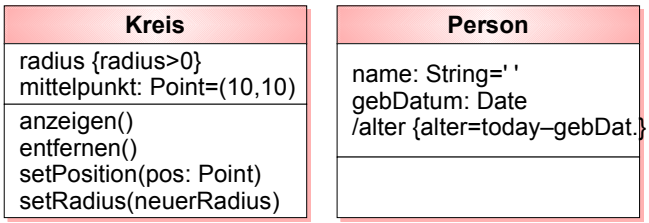
Notation Klasse



Regeln Klasse

- Klassenname: einziges obligatorisches Element
- Datentyp: andere Klasse, Schnittstelle oder selbstdefinierter Typ
- abgeleitetes Attribut/Operation: vorangestellter Schrägstrich, wird zur Laufzeit berechnet
- Klassenattribut/Klassenoperation: unterstrichen, für alle Objekte der Klasse
- Richtung: in, out, inout

Beispiele Klasse



Spezialklassen

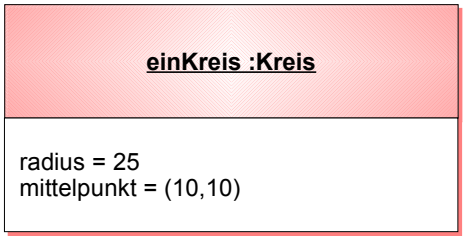
- Metaklasse: Exemplare sind wiederum Klassen, Stereotyp «metaclass»
- parametrisierbare Klasse: Schablone mit generischen Parametern zur Erzeugung von Klassen, z.B. Behälterklasse
- abstrakte Klasse: Oberklasse ohne eigene Objekte, Merkmal {abstrakt} oder kursiver Klassenname
- Hilfsmittelklasse (utility class): Sammlung von globalen Variablen und Funktionen, Stereotyp «utility»
- Schnittstellenklasse: spezifizieren das extern sichtbare Verhalten von Klassen, Stereotyp «interface»

Notation Objekt

Regeln Objekt

- Objektbezeichnung wird unterstrichen, besteht aus Objektname und/oder Doppelpunkt Klassenname
- Attribute können Beispielwerte enthalten
- Operationen werden nicht angegeben

Beispiel Objekt



Extensions-Elemente

Zusicherung (constraint)

Einschränkung der möglichen Inhalte, Zustände oder Semantik eines Modellelements.

Darstellung: in geschweiften Klammern
Formulierung: Freitext

Zweck:
- zulässiger Wertebereich für Attribute
- Berechnungsvorschriften für abgeleitete Attribute
- Vor- oder Nachbedingungen für Operationen
- Abhängigkeitsbeziehungen für Assoziationen
- Ordnung
- zeitliche Bedingung

Merkmal (tagged value)

Erweiterung der Semantik eines Modellelements um spezielle charakteristische Eigenschaften (properties).

Darstellung: in geschweiften Klammern
Formulierung: Schlüsselwort=Wert oder nur Schlüsselwort bei true Boolean

Zweck:
- Code-Generierung

Beispiele:
abstract, derived, ordered

Stereotyp

Projekt-, unternehmens- oder methodenspezifische Ergänzungen vorhandener Modellelemente des UML-Metamodells.

Darstellung: in doppelten Winkelklammern

Zweck:
- Definition eines Anwendungskontexts
- Schaffung neuer Modellelemente

Stereotyp-Kategorien

- dekorativ (decorative):
  Symbole zur optischen Aufwertung von Diagrammen
  «actor», «boundary», «control», «entity»

- deskriptiv (descriptive):
  zur Kontextbeschreibung, Kommentierung
  «business class», «process control»
- restriktiv (restrictive):
  zur Einschränkung von Eigenschaften, Restriktion
  «interface», «metaclass»
- redefinierend (redefining):
  zur Veränderung fundamentaler Bedeutungen

Notiz

Kommentar oder Anmerkung ohne semantische Wirkung.

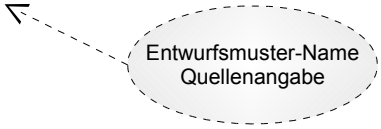
Darstellung: Rechteck mit Eselsohr

Zweck:
- mehrere betroffene Elemente

Entwurfsmuster

Beschreibt ein häufig auftretendes Entwurfsproblem und einen Lösungsansatz dazu.

Darstellung: gestrichelte Ellipse



Statische Beziehungselemente

Vererbung (inheritance)

Ist-Beziehung

Klassifikationssystem, hierarchische Strukturierung in Ober- und Unterklassen, Generalisierung / Spezialisierung:

Taxonomische Beziehung zwischen allgemeinem und speziellem Element. Das speziellere erbt die Eigenschaften des allgemeinen und führt selbst nur die zusätzlichen.

Mehrfachvererbung:
Heterarchie, mehr als eine direkte Oberklasse pro Unterklasse

Diskriminator

Unterscheidungsmerkmal für die Strukturierung in Ober- und Unterklassen.

Die Klassenhierarchie entsteht nicht nach Gesichtspunkten der Optimierung und Redundanzfreiheit, sondern folgt der Anschauung der zu modellierenden Welt.

Der Diskriminator kann auch als Klasse mit dem Stereotyp «powertype» definiert werden.

Assoziation (association)

Benutzt-Beziehung, Kennt-Beziehung, Verbindung

Gleichberechtigte Beziehung zwischen Klassen.

Modellierungsvarianten:
- bidirektionale Mehrfachbeziehungen können durch Assoziationsattribute (degenerierte Assoziationsklasse ohne Namen) oder als eigene aufgebrochene Assoziationsklasse dargestellt werden
- mehrgliedrige Assoziationen können auch explizit als Klasse modelliert werden

Referentielle Integrität

Regeln zur Sicherstellung der Integrität von Objektbeziehungen beim Löschen.

Beschreibung in Zusicherungen:
- delete related object:
  das gegenüberliegende Objekt wird ebenfalls gelöscht
- delete link:
  nur die Beziehung wird gelöscht
- prohibit deletion:
  Löschen wird nicht zugelassen

Aggregation (aggregation)

Hat-Beziehung, Zerlegung
Sonderform der Assoziation, Teile-Ganzes-Hierarchie.

Die Kardinalität auf der Seite des Aggregats ist häufig 1.

Delegation / Propagation:
Die Teile-Klassen können ihre Eigenschaften durch Delegation erweitern (propagieren), indem sie Operationen der Aggregat-Klasse mitverwenden. Die Aggregat-Klasse übernimmt Aufgaben für die Teile und leitet Nachrichten an diese weiter. Dadurch ist Delegation ein Mechanismus zur Vermeidung von Mehrfachvererbung.

Komposition (composite)

Sonderform der Aggregation, Einzelteile sind vom Aggregat (dem Ganzen) existenzabhängig.

Die Kardinalität auf der Seite des Aggregats ist immer 1.

Abhängigkeit (dependency)

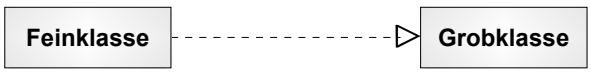
Beziehung zwischen zwei Modellelementen, die zeigt, dass eine Änderung im unabhängigen Element eine Änderung im abhängigen Element notwendig macht.

Beispiele: derive, instanceOf, instantiate

Verfeinerung (refinement)

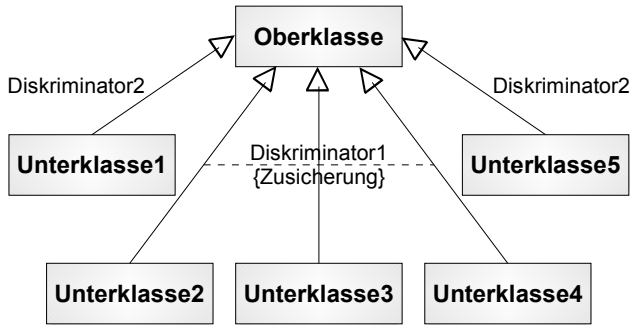
Beziehung zwischen gleichartigen Elementen unterschiedlichen Detaillierungs- bzw. Spezifikationsgrades.

Beispiele: refine, bind



Notation Vererbung

Einzel-Notation Vererbung

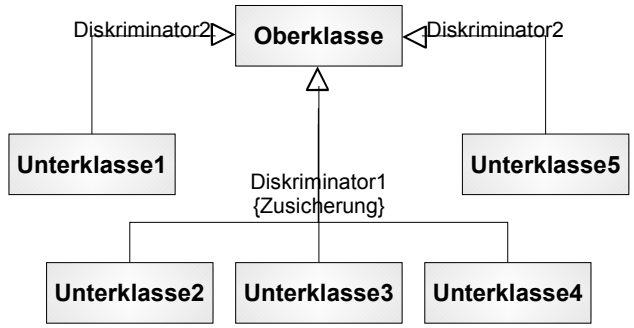


Regeln Vererbung

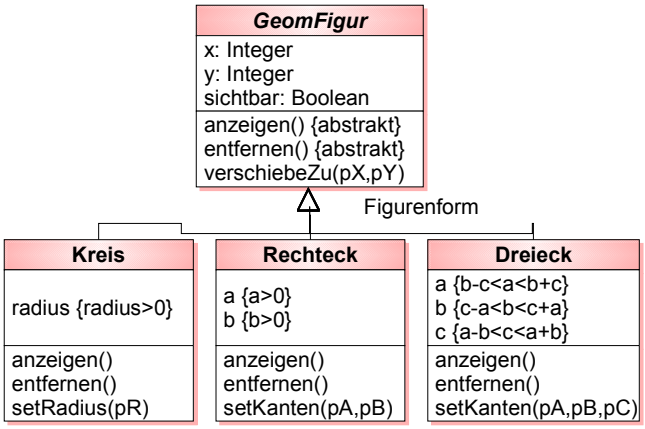
vordefinierte Zusicherungen:

- overlapping: Unterklassen können die gleiche Instanz enthalten
- disjoint: Unterklassen sind disjunkt, können nicht die gleiche Instanz enthalten
- complete: alle Unterklassen sind spezifiziert
- incomplete: weitere Unterklassen werden erwartet

Gemeinsame Notation Vererbung

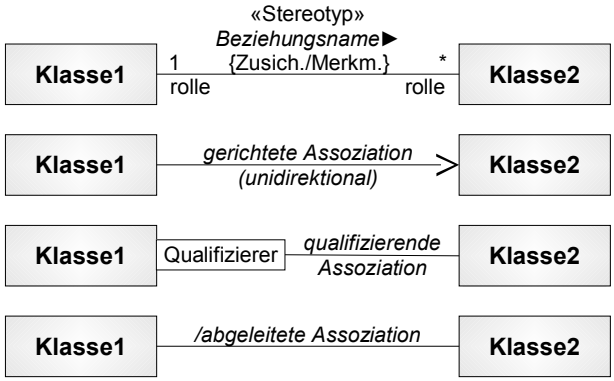


Beispiel Vererbung



Notation Assoziation

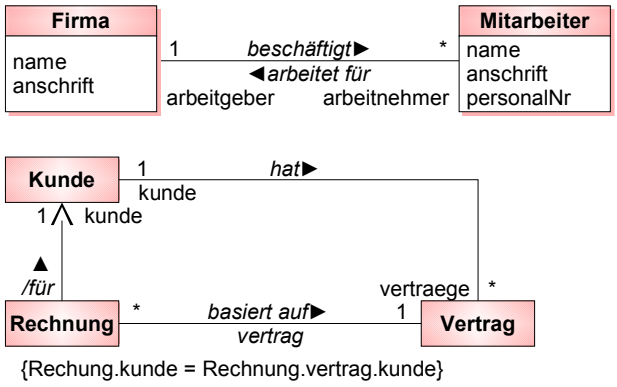
Notation Assoziation



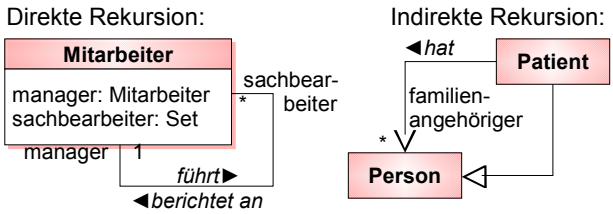
Regeln Assoziation

- Dreieck gibt die Leserichtung an
- Multiplizität / Kardinalität: Anzahl beteiligter Elemente, Aufzählung oder Wertebereich min..max, * heisst >=0
- Rolle beschreibt die Sichtweise der gegenüberliegenden Klasse, wird dort als Referenzattribut bzw. Behälterobjekt für mehrere Objekte (Set, Liste) implementiert
- der Schlüssel zu Behälterobjekten kann als Qualifizierer in einem Rechteck an der Seite der Klasse stehen

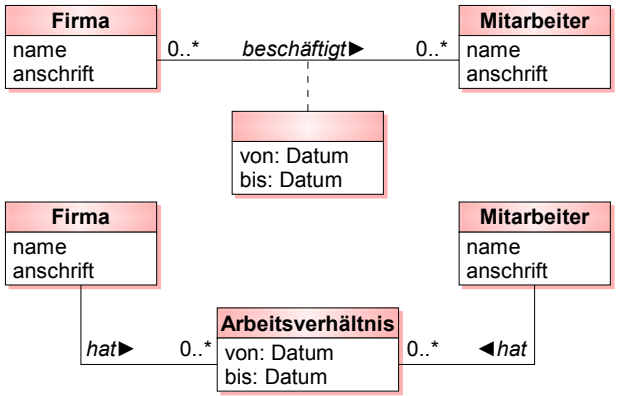
Beispiele Assoziation



Rekursive Assoziation

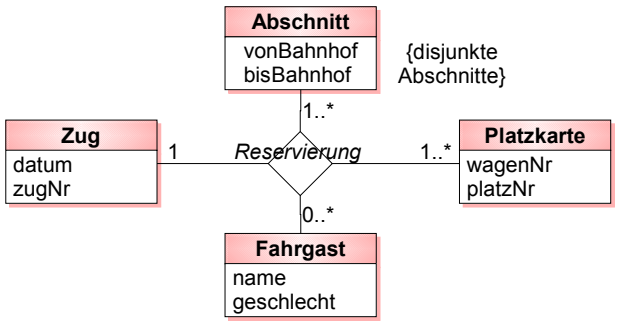


Attributierte Assoziation

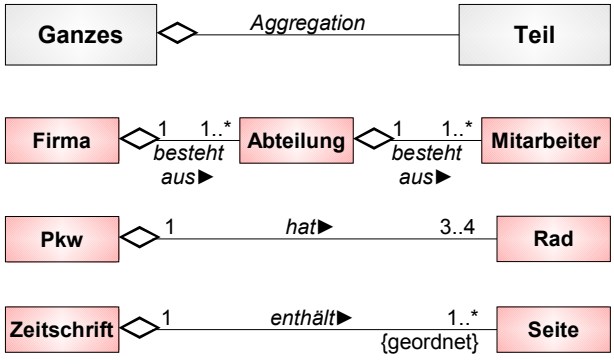


Mehrgliedrige Assoziation

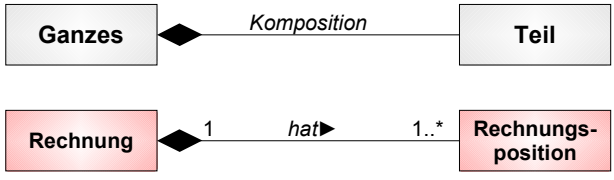
ternär: 3fach, quaternär: 4fach



Aggregation



Komposition



Dynamische Beziehungselemente

OAOD

Nachricht (message)

Überbringt einem Objekt die Information, welche Dienstleistung (Operation) angefordert wird.

- geht an genau einen Empfänger
- Arten:
  - Operationsaufruf
  - Signal
- Bestandteile:
  - Selektor (Name)
  - Liste von Argumenten
- Sender erhält genau ein Antwort-Objekt

Polymorphismus

Gleichlautende Nachrichten können bei kompatiblen Objekten unterschiedlicher Klassen ein unterschiedliches Verhalten bewirken (Vielgestaltigkeit).

- statischer Polymorphismus: Überladen, Zuordnung der Nachricht zur empfangenden Operation beim Kompilieren, andere Signatur, z.B. andere Argumente
- dynamischer Polymorphismus: Überschreiben, Zuordnung der Nachricht zur empfangenden Operation zur Laufzeit, Voraussetzung ist dynamisches Binden

Definition Klassendiagramm

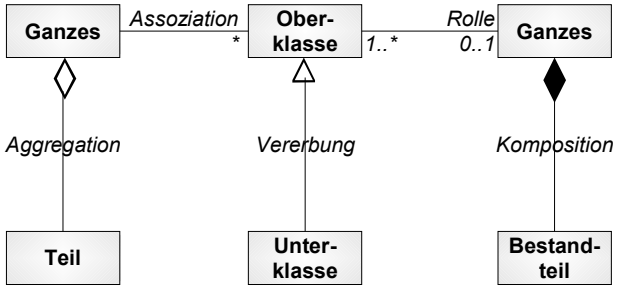
Beschreibt die statische Struktur der Klassen in einem System sowie ihre Beziehungen untereinander.

Anwendung:
- in allen Projektphasen möglich:
  - konzeptuell-analytische Modellierung: grundlegende Zusammenhänge
  - logische Modellierung: konkrete Implementierungsvorschrift
- statische, logische Sicht auf ein System

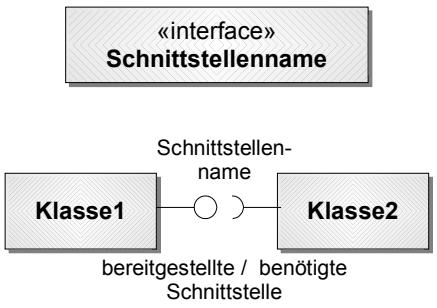
Elemente Klassendiagramm

- Klasse
- Attribut
- Operation
- Assoziation, Aggregation, Komposition
- Generalisierungsbeziehung
- Abhängigkeitsbeziehung
- Schnittstelle:
  - provided interface: wird angeboten
  - required interface: wird benötigt

Notation Klassendiagramm



Schnittstelle



Komponentendiagramm

Definition Komponentendiagramm

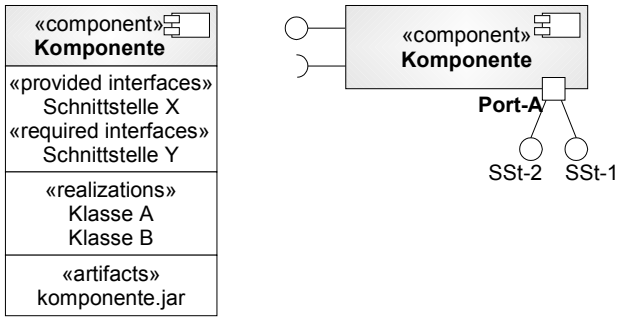
Zeigt Komponenten und ihre Beziehungen und Schnittstellen.

Anwendung:
- Zusammenhänge
- fachliche Architekturmuster (3tier, MVC)
- Systemstruktur zur Laufzeit
- Compiler- und Laufzeit-Abhängigkeiten

Elemente Komponentendiagramm

- Komponente
- Artefakt
- Schnittstelle
- Port
- Abhängigkeit (definiert Kompilationsreihenfolge):
  - «use»: Verwendungsbeziehung, verbindet Komponente mit Schnittstelle
  - «manifest»: Implementierungsbeziehung, verbindet Artefakt mit Komponente
- Realisierungsbeziehung

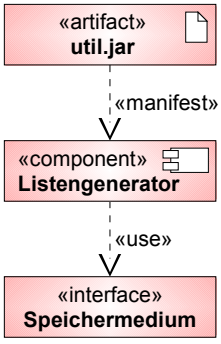
Notation Komponentendiagramm



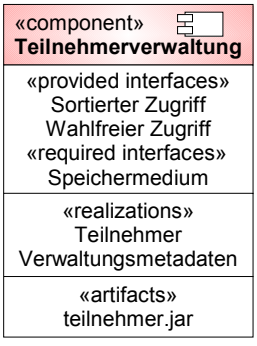
Regeln Komponentendiagramm

- Komponente: modularer Systemteil mit transparenter Kapselung seines Inhalts, physisches Stück Programmcode (Quellcode, Binärcode, ausführbares Programm)
- Komponenten können wiederum Elemente enthalten
- Artefakt: physische Informationseinheit, die während des Entwicklungsprozesses erzeugt oder verwendet wird (Modell, Quellcode, Objektdatei)
- mehrere Schnittstellen können mit Ports dargestellt werden
- Port: öffentlich sicht- und zugreifbarer Interaktionspunkt, der auf einen Operationsaufruf oder den Empfang eines Signals reagiert, Kommunikationsschnittstelle

Abhängigkeiten im Komponentendiagramm



Beispiel Komponente



Kompositionsstrukturdiagramm

Definition Kompositionsstrukturdiagramm

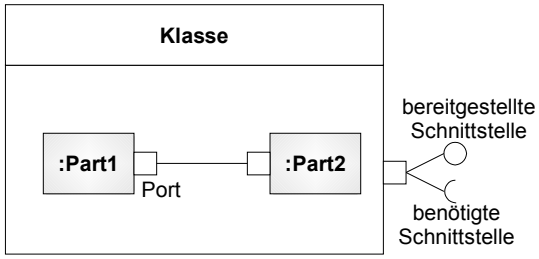
Zeigt die interne Struktur eines Classifiers und seine Beziehungen zu anderen Systembestandteilen.

Anwendung:
- Top-Down-Modellierung des Systems
- Abbildung innerer Zusammenhänge einer komplexen Systemarchitektur
- Darstellung von Design Patterns

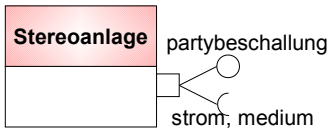
Elemente Kompositionsstrukturdiagramm

- Part: Ausprägung in einem Classifier (Objekt, Rolle)
- Port
- Schnittstelle
- Konnektor: Link

Notation Kompositionsstrukturdiagramm



Beispiel Kompositionsstrukturdiagramm



Verteilungsdiagramm

Definition Verteilungsdiagramm

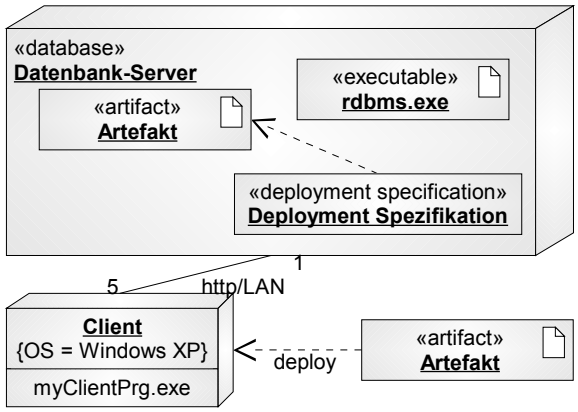
Zeigt die Zuordnung von Softwarekomponenten auf Hardwareknoten in einem Netzwerk und die Kommunikation und Abhängigkeit zwischen den Knoten.

Anwendung:
- Beschreibung Systemarchitektur
- Beschreibung Laufzeitumfeld
- Hardwareaufbau, Konfiguration
- Client-Server-Systeme
- nicht funktionale Anforderungen
- grössere Projekte

Elemente Verteilungsdiagramm

- Knoten: zur Laufzeit physisch verfügbare Ressource, dargestellt durch Quader
- Kommunikationspfad: verbindet Knoten für Nachrichtenaustausch, dargestellt durch Assoziation, kann gerichtet sein
- Abhängigkeit:
  - «deploy»: Verteilungsbeziehung, verbindet Artefakt mit Knoten

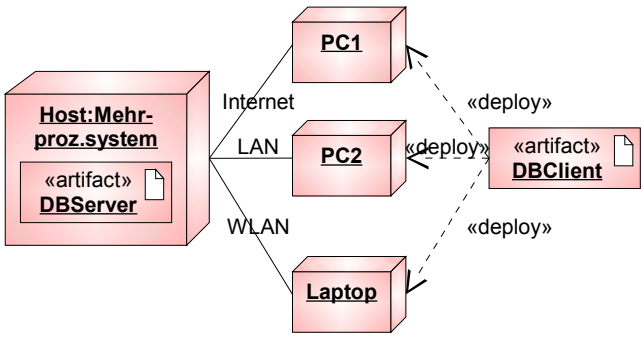
Notation Verteilungsdiagramm



Regeln Verteilungsdiagramm

- Knoten können wiederum Elemente enthalten
- Gerät: Knoten mit Stereotyp «device»

Beispiel Verteilungsdiagramm



Objektdiagramm

Definition Objektdiagramm

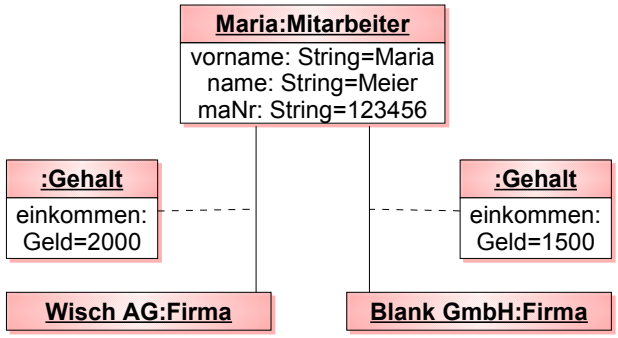
Zeigt Ausprägungen der im Klassendiagramm modellierten Typen zu einem bestimmten Zeitpunkt.

Anwendung:
- statische, logische Sicht aus der Perspektive realer Fälle
- Betrachtung komplexerer Strukturen in einer konkreten Momentaufnahme

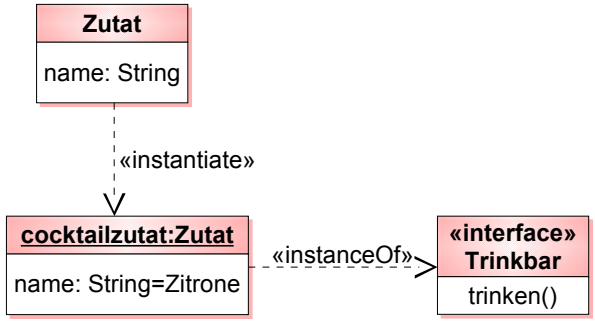
Elemente Objektdiagramm

- Objekt: Ausprägung einer oder mehrerer Klassen
- Link: Ausprägung einer Assoziation, Multiplizität immer 1
- Wert: Ausprägung eines Attributs
- keine Operationen (in Klasse definiert)
- Abhängigkeit:
  - «instanceOf»: Quelle ist eine Instanz des Ziels
  - «instantiate»: Quelle erstellt Instanzen des Ziels

Beispiel Objektdiagramm



Abhängigkeiten im Objektdiagramm



Definition Paket

Gruppierung von Modellelementen beliebigen Typs aufgrund logischer oder physischer Zusammenhänge.

Ein Element kann in mehreren Paketen referenziert werden, aber nur in einem enthalten sein.
Syntax: Paketname::Elementname

Darstellung: Aktenregister

Definition Paketdiagramm

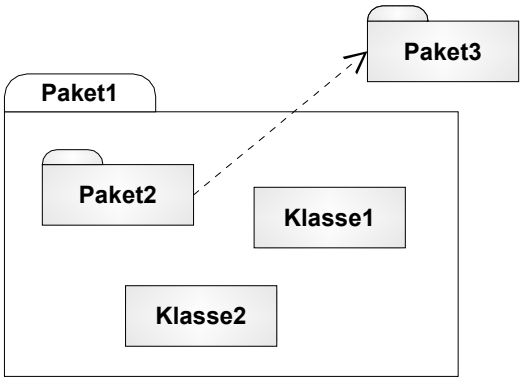
Gliedert Softwaresysteme in Untereinheiten.

Anwendung:
- Groborientierung
- Gliederung in überschaubare Einheiten
- hierarchische Gliederung möglich
- geeignete Arbeitsgrössen für Projektmanagement
- Strukturierung in Anwendungsbausteine (architektonisch vollständiger fachlicher Ausschnitt des Gesamtmodells)
- Vorteil: Wiederverwendbarkeit

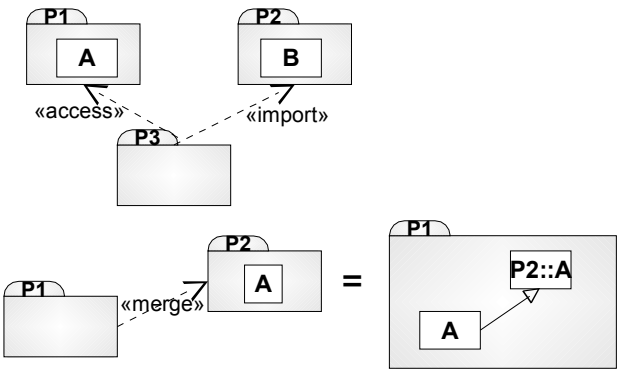
Elemente Paketdiagramm

- Paket
- Abhängigkeit:
  - «import»: public Paket-Import, Elemente aus anderen Paketen werden referenziert
  - «access»: private Paket-Import
  - «merge»: Erweiterung des Import, referenzierte Elemente werden implizit neu definiert

Notation Paketdiagramm



Abhängigkeiten im Paketdiagramm



Definition Use Case

Anwendungsfall, Beschreibung eines Arbeitsgangs (in sich geschlossener, zeitlich ununterbrochener Ausschnitt aus dem Geschäftsprozessmodell) aus Sicht des Anwenders.

Merkmale:
- Anstoss durch Akteur
- funktionales Verhalten
- für den Akteur sichtbares Ergebnis

Arten:
- Szenario: konkreter Anwendungsfall
- primärer Anwendungsfall: Standardfall
- sekundärer Anwendungsfall: Ausnahme-, Sonderfall

Darstellungsarten

- textliche Beschreibung
- evtl. illustriert mit Dialogentwürfen
- Gliederung in Tabellenform
- Anwendungsfalldiagramm

Use Cases in Tabellenform

- Anwendungsfall-Nr. und -name
- Kurzbeschreibung
- beteiligte Akteure
- Auslöser, eingehendes Ereignis
- Vorbedingung, vorheriger Zustand
- Ablaufschritte (normal flow)
- Nachbedingung, nachheriger Zustand
- Varianten mit abweichendem Verhalten (alternative flows)
- Verweis auf Szenarien (Aktivitäts- / Interaktionsdiagramm)

Anwendungsfalldiagramm

OAOD

Definition Anwendungsfalldiagramm

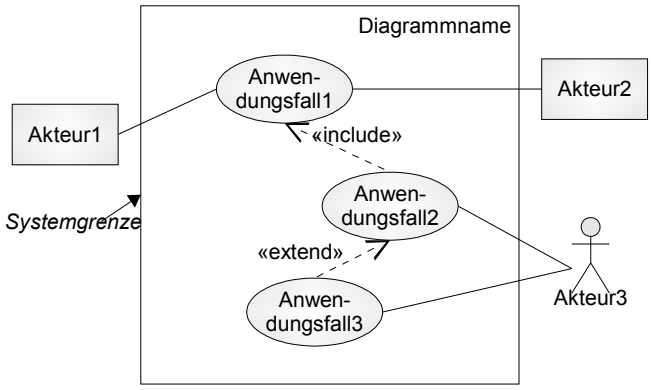
Beschreibt die Interaktionen zwischen Akteuren und Anwendungssystem und die Zusammenhänge zwischen Anwendungsfällen in einem Geschäftsprozess.

Anwendung:
- Beschreibung des Soll- und evtl. des Ist-Zustands
- Ermittlung der funktionalen Anforderungen
- Systemabgrenzung und Systemüberblick
- Verhalten des Systems nach aussen (Blackbox)
- Kommunikation mit Anwendern, Fachabteilung
- Erschliessung der Terminologie der Anwendungswelt
- Einarbeitung der Entwickler in das Anwendungsgebiet
- Erstellung planbarer Einheiten
- Vorlage für Testfälle

Elemente Anwendungsfalldiagramm

- Systemgrenze
- Anwendungsfall
- Akteur (Rolle): Person, System, Zeit usw.
- Assoziation: Beziehung zwischen Akteur und Use Case, Richtungsfunktion, Rollen und Multiplizitäten möglich
- Vererbungsbeziehung
- Abhängigkeit:
  - «include»: Beinhaltet-Beziehung, Pfeilrichtung zum inkludierten Anwendungsfall (Muss)
  - «extend»: Erweiterungsbeziehung Pfeilrichtung von der Erweiterung / Variante (Kann)

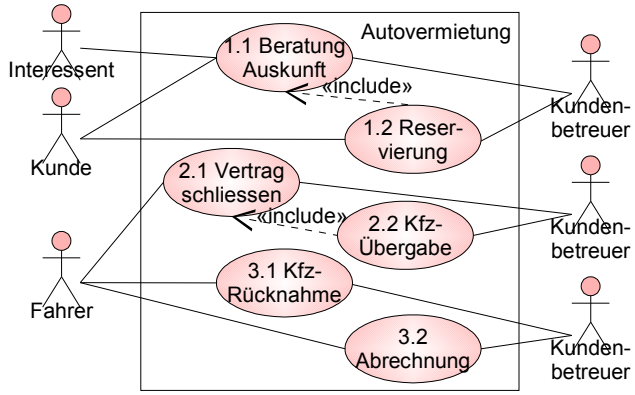
Notation Anwendungsfalldiagramm



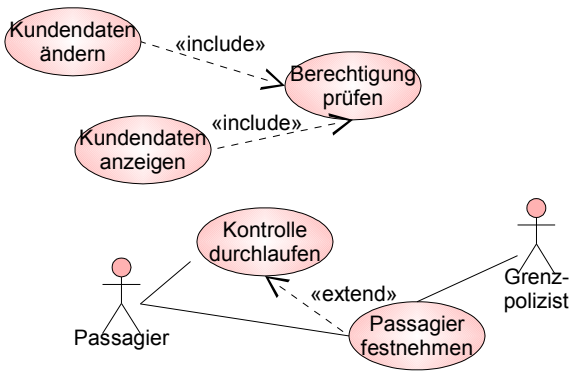
Regeln Anwendungsfalldiagramm

- hierarchische Verschachtelung möglich
- Anwendungsfälle können zu Paketen zusammengefasst werden

Beispiel Anwendungsfalldiagramm



Abhängigkeiten im Anwendungsfalldiagramm



Aktivitätsdiagramm

Definition Aktivitätsdiagramm

Beschreibt interne Ablaufmöglichkeiten von Modellelementen (Klasse, Operation, Anwendungsfall).
Token-Konzept:
Ablauf wird gedanklich von Token gesteuert (wie Petrinetze)

Anwendung:
- Geschäftsprozessmodellierung
- Modellierung von Fallunterscheidungen, Sequentialisierungen und Parallelitäten
- Darstellung von Datenflüssen
- Beschreibung von Use Cases
- Implementieren einer Operation

Elemente Aktivitätsdiagramm

- Knoten (node):
  - action node:
    Aktion, Ablaufschritt, Verhaltensaufruf bestehend aus Substantiv und Verb im Infinitiv, hierarchische Schachtelung mit Harke dargestellt
  - object node:
    Objekttyp, Zustand eines Objekts, beteiligte Daten
  - control node:
    Verhaltensaufrufe mit eigenem Layout
- Aktivitätskante (activity edge):
  Transitionen, Übergänge, automatisch nach Aktion
- Konnektor:
  zur Referenzierung von Kanten
- Aktivitätsbereich

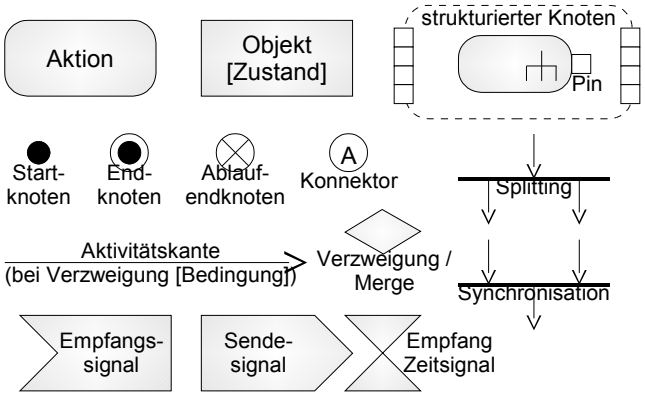
Control Nodes

- ein bis mehrere Start- und Endknoten
- Ablaufendknoten: beendet einzelnen Ablaufzweig
- Verzweigung (branch, decision):
  alternative (sich ausschliessende) Möglichkeiten, direkt aus dem Aktivitätssymbol oder mit Rautensymbol, Verzweigungsbedingung in eckigen Klammern
- Zusammenführung (Oder, merge):
  Fortführung des Aktivitätsflusses nach dem ersten eingehenden Fluss
- Teilung (splitting, fork):
  Verteilung des Aktivitätsflusses auf parallele Abläufe
- Synchronisation (Und, join):
  Fortführung des Aktivitätsflusses nach allen eingehenden Flüssen

Spezielle Knoten

- Signal: Benachrichtigung über Ereignis
  - SendSignalAction: Sendesignal
  - AcceptEventAction: Empfangssignal
- Pin:
  Objektknoten, der den Verhaltensaufrufen als Input oder Output für Parameter dient
- Listbox-Pin:
  mehrere Pins symbolisieren Mengenverarbeitung, Übergang von Aktionssammlung in Einzelelemente und zurück, iterativ, parallel, stream

Notation Aktivitätsdiagramm



Strukturierte Knoten

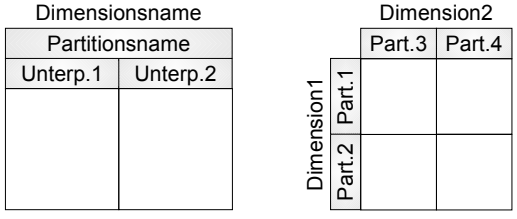
Gruppierung von Elementen.

- Schleifenknoten:
  for / while / do
- Entscheidungsknoten:
  if / then / else if / else

Aktivitätsbereiche

Einteilen des Diagramms in Bereiche mit gemeinsamen Eigenschaften.

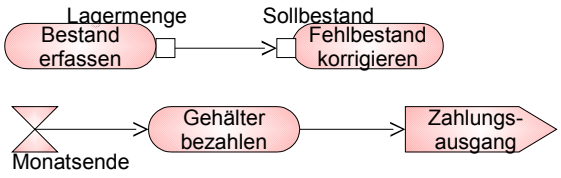
- hierarchische Aktivitätsbereiche
- mehrdimensionale Aktivitätsbereiche



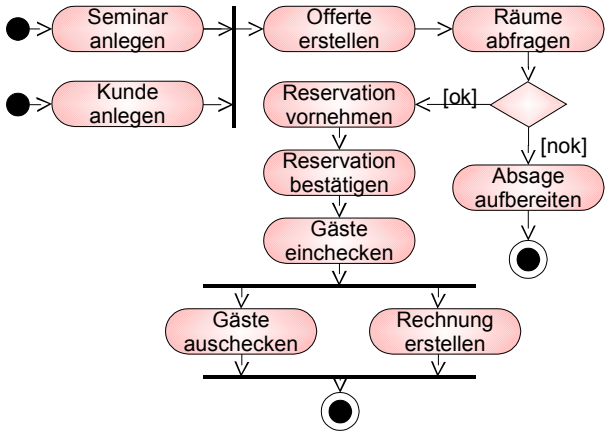
Aktivitätsdiagramm Beispiele

OAOD

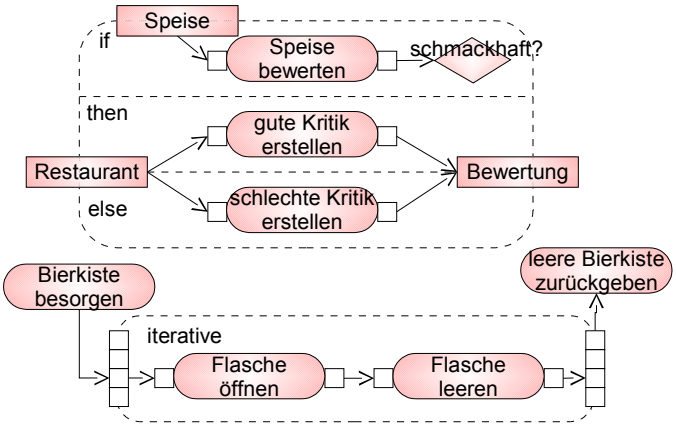
Mini-Beispiele



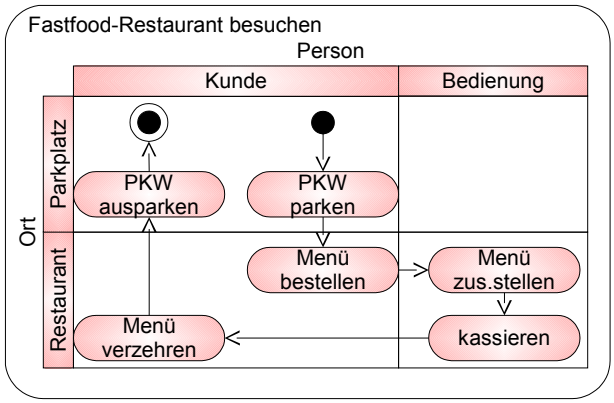
Seminar durchführen



Strukturierte Knoten



Aktivitätsbereiche



Zustandsdiagramm

Definition Zustandsdiagramm

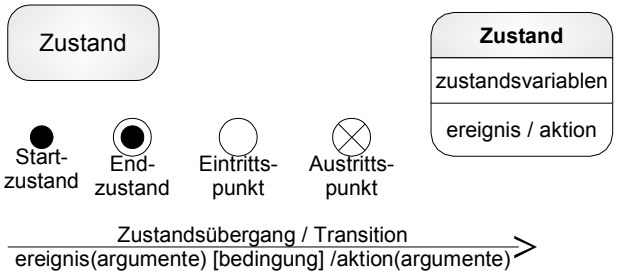
Zeigt eine Folge von Zuständen eines Objekts (Lebenslauf). Beschreibt, durch welche Ereignisse welche Transaktionen ausgelöst werden und wann diese zulässig sind.

Anwendung:
- dynamisches Verhalten
- komplexe Objekte
- Detaillierung von Use Cases
- Verhaltensbeschreibung von Interfaces und Ports

Elemente Zustandsdiagramm

- Interaktionsrahmen sm (state machine)
- Zustand: statische Situation zwischen zwei Ereignissen
- anonymer Zustand: Zustand ohne Namen
- ein Startzustand, ein bis mehrere Endzustände
- Ein- und Austrittspunkt: zum Betreten und Verlassen von Unterzuständen
- Pseudozustände: Verzweigung, Zusammenführung, Teilung, Synchronisation analog Aktivitätsdiagramm
- Transition: Übergang von Quell- in Zielzustand
- Selbsttransition: Quellzustand gleich Zielzustand

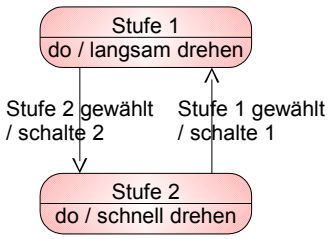
Notation Zustandsdiagramm



Regeln Zustandsdiagramm

- Zustandsvariablen: relevante Attribute dieser Klasse
- Ereignis (Trigger): löst Zustandsübergang oder Aktion innerhalb des Zustands aus
- Ereignisursachen: erfüllte Bedingung (Guard), Nachricht
- Übergänge ohne Ereignisbeschriftung werden automatisch nach Abschluss der Aktionen ausgelöst
- Funktionen beschreiben Zustandsübergänge

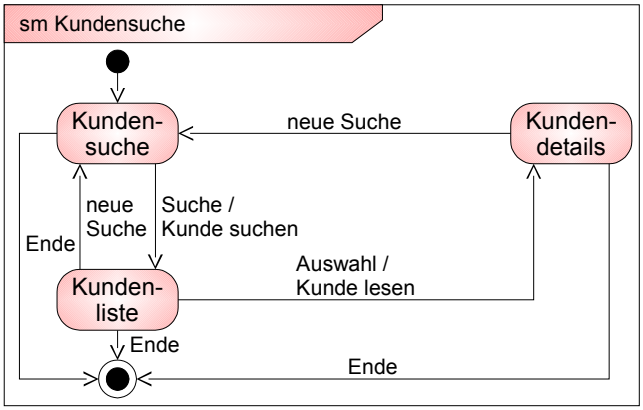
Beispiele Zustandsübergänge



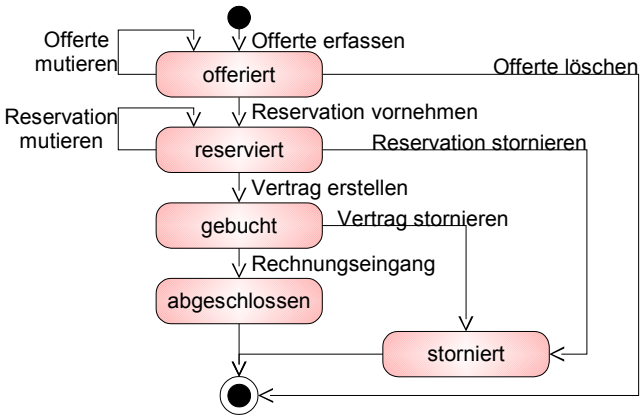
Zustandsinterne Ereignisse

- entry: löst automatisch beim Eintritt in einen Zustand aus
- exit: löst automatisch beim Verlassen eines Zustands aus
- do: wird immer wieder ausgelöst, solange der Zustand aktiv ist

Beispiel 1 Zustandsdiagramm



Beispiel 2 Zustandsdiagramm



Sequenzdiagramm

Definition Sequenzdiagramm

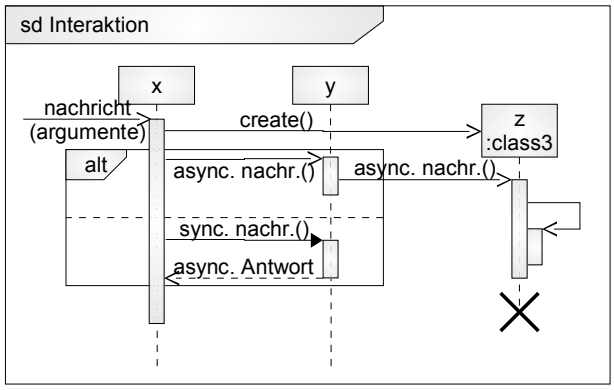
Zeigt den zeitlichen Ablauf von Nachrichten zwischen Objekten in einer bestimmten begrenzten Situation.

Anwendung:
- zeitliche Aufrufstruktur mit wenigen Klassen
- Betonung des zeitlichen Nachrichtenverlaufs
- Darstellung von Nachrichtenflüssen aus Anwendungsfällen

Elemente Sequenzdiagramm

- Interaktionsrahmen sd
- Kommunikationspartner: horizontal aufgereiht und als vertikale Lebenslinien dargestellt
- Aktionssequenz: Kommunikationspartner ist aktiv, dargestellt als breiter Balken
- Nachricht: verläuft horizontal
  - Asynchron: Sender arbeitet sofort weiter
  - Synchron: Sender muss auf Empfänger warten
- Terminator: beendet die Lebensdauer einer Instanz
- Vor- und Nachbedingung: mit Notizzettel notiert
- Fragment: Aufteilung in Bereiche

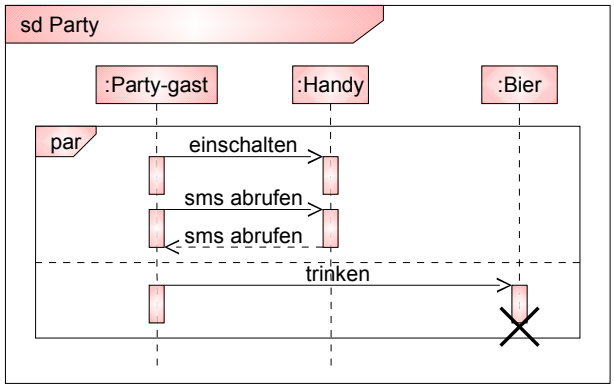
Notation Sequenzdiagramm



Regeln Sequenzdiagramm

- Zeit verläuft vertikal von oben nach unten
- Objekte können erzeugt und entfernt werden

Beispiel Sequenzdiagramm



Fragmente

Interaktionsoperatoren (links oben im Fragment):
- 'ref' = Referenz auf anderes Sequenzdiagramm (Zooming)
- 'alt' = Auswahl unter Alternativen
- 'opt' = optionale Ausführung
- 'par' = parallele Ausführung (Threading)
- 'loop' = iterative Schleife
- 'break' = Abbruch
- 'critical' = ununterbrechbarer Ablauf
- 'neg' = Negation
- 'assert' = bestimmte Reihenfolge
- 'ignore' = Ausblenden irrelevanter Nachrichten
- 'consider' = Konzentration auf relevante Nachrichten
- '{...}' = Constraint

Kommunikationsdiagramm

Definition Kommunikationsdiagramm

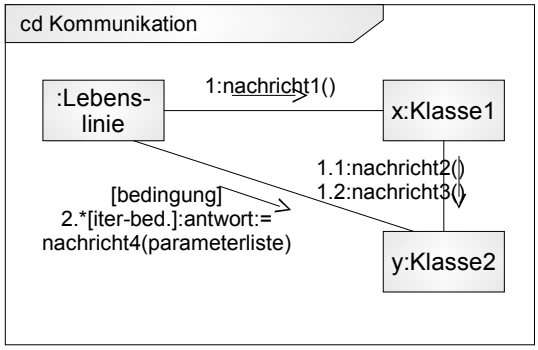
Zeigt Interaktionen zwischen ausgewählten Objekten in einer bestimmten begrenzten Situation.

Anwendung:
- zeitliche Aufrufstruktur mit wenigen Nachrichten
- Betonung der Beziehungen zwischen den Objekten
- Modellierung von Prinzipien und Konzepten

Elemente Kommunikationsdiagramm

- Interaktionsrahmen cd
- Kommunikationspartner: Lebenslinie ohne Linie
- Nachricht: Link (Ausprägung einer Assoziation)

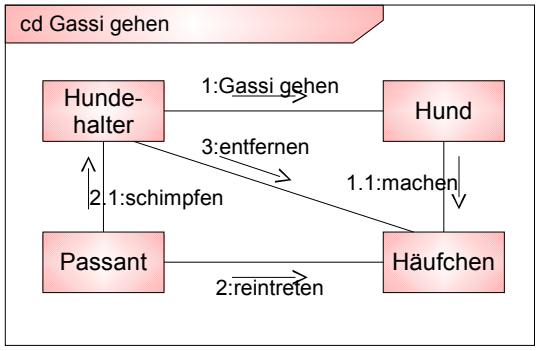
Notation Kommunikationsdiagramm



Regeln Kommunikationsdiagramm

- Nachrichten werden mit Methodenaufruf, Argumenten und Rückgabewert notiert
- Sequenzbezeichner: definiert Reihenfolge und Ebene der Abarbeitung
- kleiner Pfeil kennzeichnet Aufrufrichtung

Beispiel Kommunikationsdiagramm



Nachrichtenarten

- sequentiell: Sequenznummern
- verschachtelt: hierarchische Sequenznummern
- parallel: Kleinbuchstaben nach Sequenznummer
- bedingt: Bedingung in eckigen Klammern
- iterativ: Stern vor Nachricht, evtl. mit Iterations-Bedingung in eckigen Klammern

Interaktionsüberblicksdiagramm

Definition Interaktionsüberblicksdiagramm

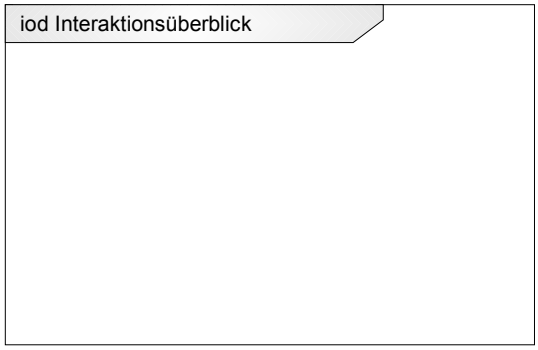
Interaktionsdiagramm zur Übersicht über Abfolgen von Interaktionen, Ablaufdarstellung wie Aktivitätsdiagramm.

Anwendung:
- logischer Zusammenhang zwischen mehreren Interaktionsdiagrammen

Elemente Interationsüberblicksdiagramm

- Interationsrahmen iod

Notation Interaktionsüberblicksdiagramm



Zeitverlaufsdiagramm

Definition Zeitverlaufsdiagramm

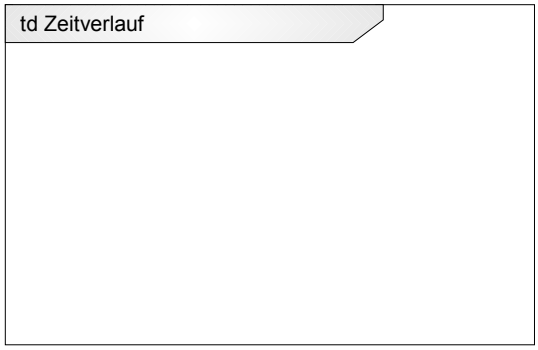
Interaktionsdiagramm mit Zeitverlaufskurven von Zuständen.

Anwendung:
- Darstellung der zeitlichen Veränderung eines Modellelements (Klasse, Objekt, Komponente)
- Modellierung zeitkritischer Zustands- und Wertänderungen

Elemente Zeitverlaufsdiagramm

- Interaktionsrahmen td
- Kommunikationspartner
- Zustandslinie: Wechsel der Zustände im Zeitverlauf
- Nachricht: wird zwischen Zustandslinien ausgetauscht

Notation Zeitverlaufsdiagramm



Beispiel Zeitverlaufsdiagramm