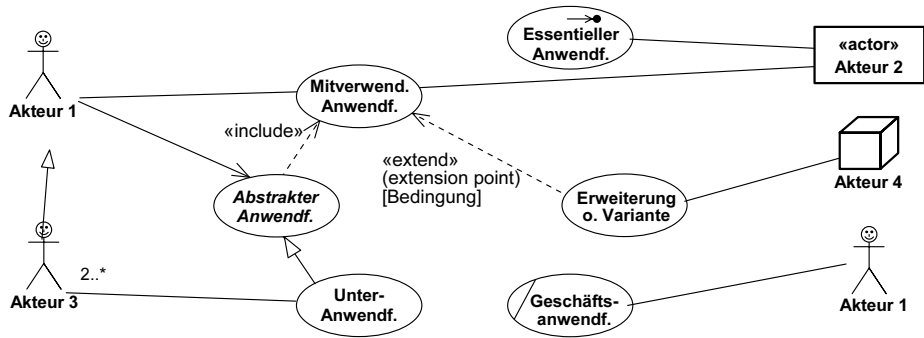
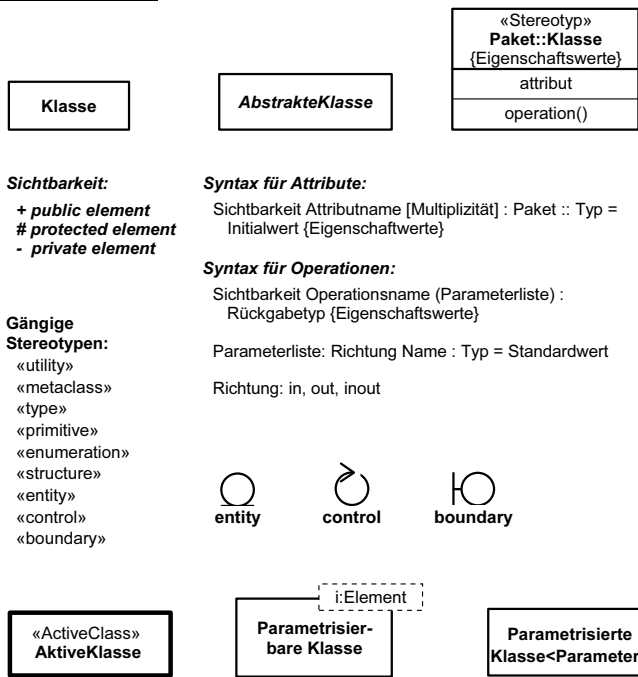


Anwendungsfalldiagramm



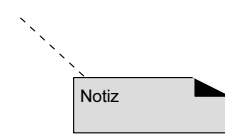
Klassen



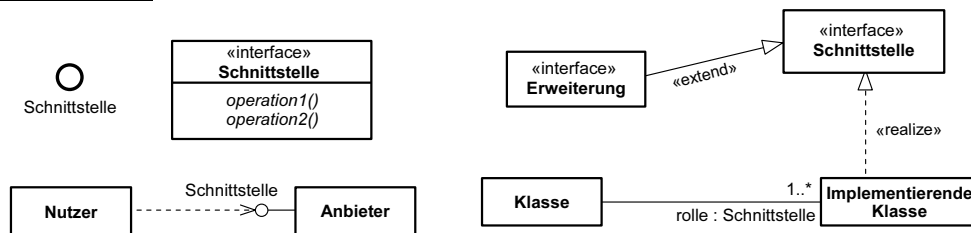
Objekte



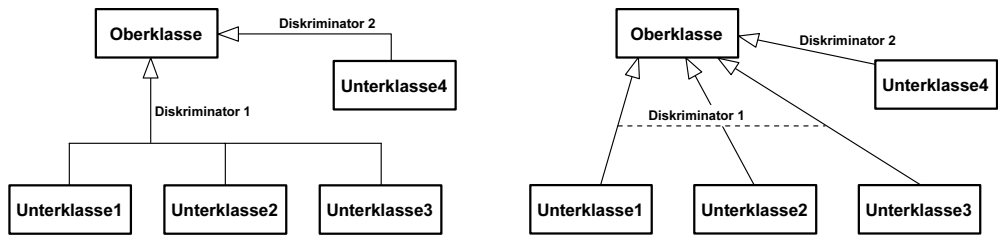
Notiz



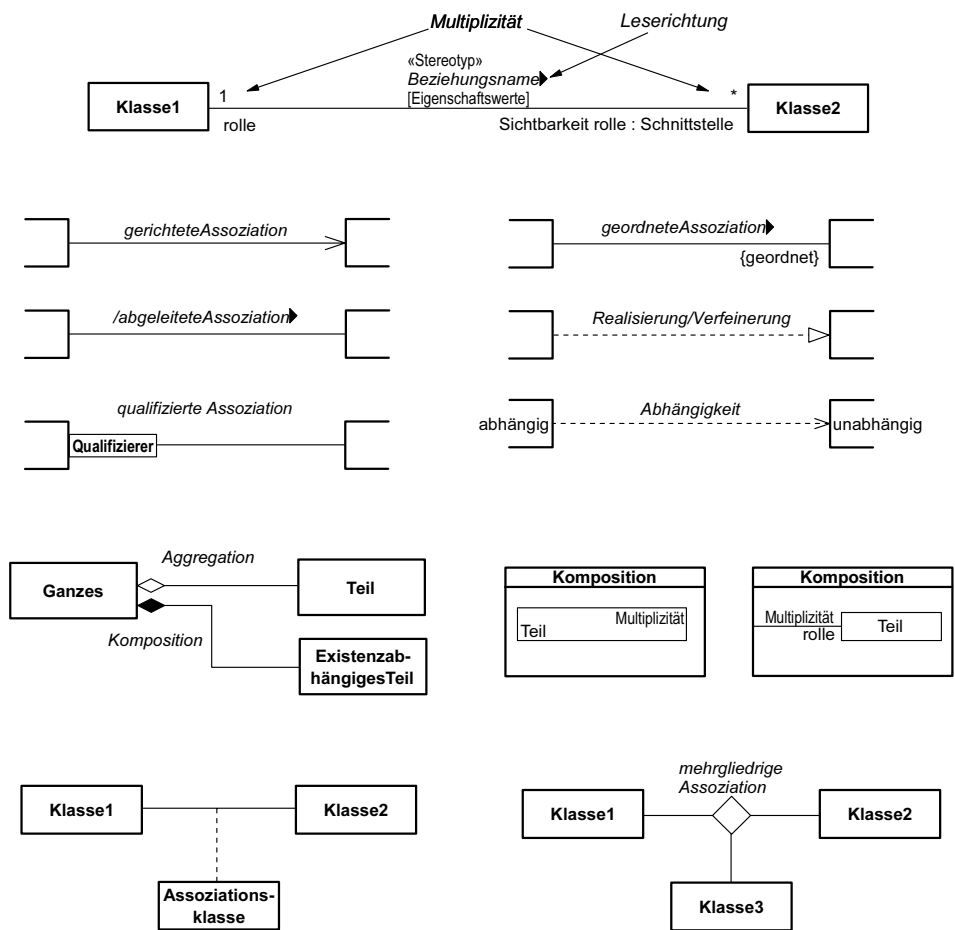
Schnittstellen



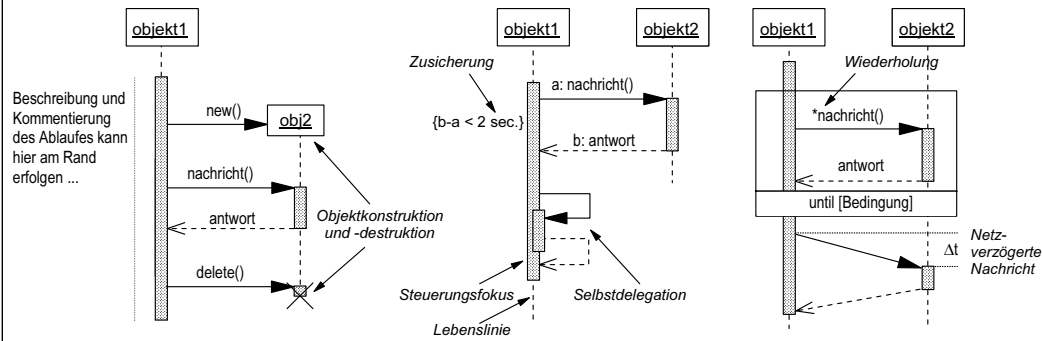
Vererbung



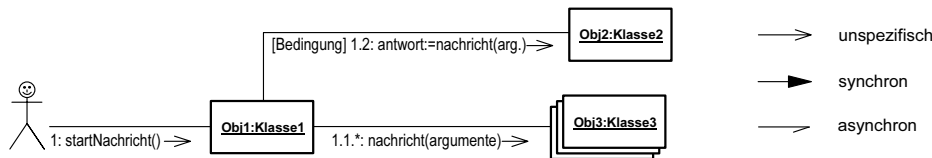
Assoziationen



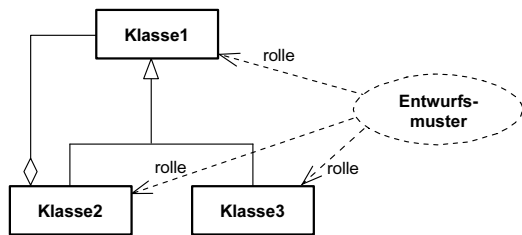
Sequenzdiagramme



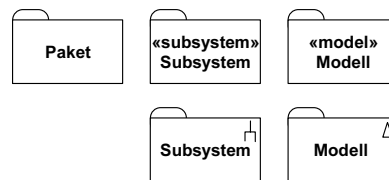
Kollaborationsdiagramme



Zusammenarbeits-/ Entwurfsmuster-Notation



Pakete, Subsysteme



Zusicherung

context Klassenname context Person
 inv: Invariante inv: alter > 18
 pre: Vorbedingung
 post: Nachbedingung

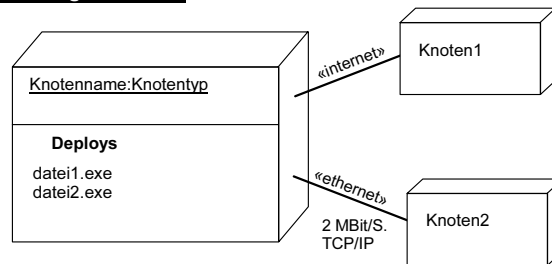
Eigenschaftswert

{schlüsselwort = wert}
 {abstract = true}
 {abstract}

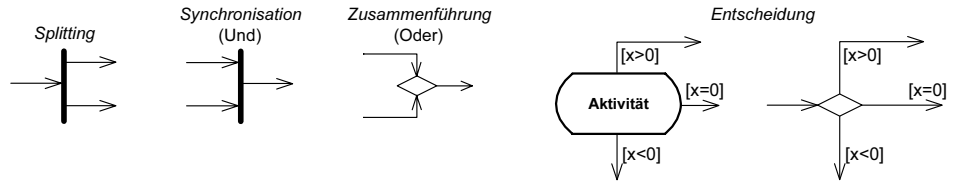
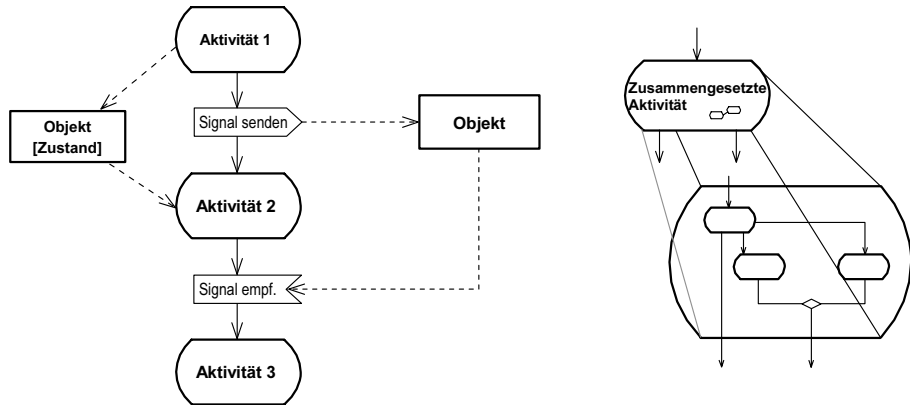
Stereotyp

«stereotyp»
 «interface»

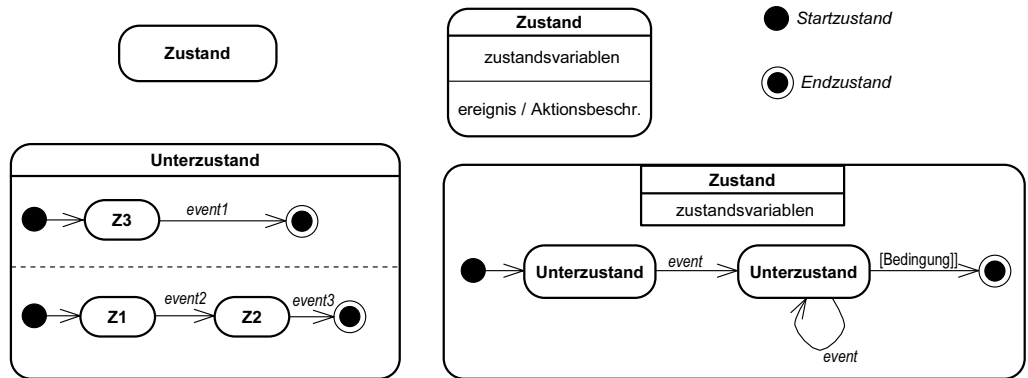
Einsatzdiagramm



Aktivitätsdiagramme



Zustandsdiagramme



Komponentendiagramme



6.1 Glossar

Die wichtigsten Begriffe der Objektorientierung und der UML werden hier definiert. Dieses Glossar ist auch separat in der jeweils aktuellen Fassung zu beziehen. Download unter: <http://oose.de/glossar>

Anmerkung zu den deutschen Begriffen

Immer wieder kommt es zu Auseinandersetzungen darüber, wie die „richtigen“ Übersetzungen der englischen Originalbegriffe lauten. Während „class“ noch zweifelsfrei übersetzt werden kann, sind „deployment diagram“ und „stakeholder“ nicht mehr so einfach einzudeutschen. Es stellt sich sogar die Frage, ob denn überhaupt deutsche Begriffe notwendig sind.

Solange Informatiker unter sich sind, werden häufig die Originalbegriffe verwendet. Bei den Anwendern handelt es sich gewöhnlich jedoch um Nicht-Informatiker, dafür aber beispielsweise um Versicherungsexperten, Buchhaltungsmitarbeiter u.ä. Um die Kommunikation mit dieser Gruppe möglichst fehlerarm zu führen, sollte deren Sprache verwendet und Informatik-Fachjargon vermieden werden. „Anwendungsfall“ ist auf Anhieb leichter zu verstehen als „use case“.

Daß Informatiker untereinander häufig die Originalbegriffe verwenden, ist weniger problematisch, wenngleich auch hier berücksichtigt werden muß, daß Begriffe nicht nur abstrakt *verstanden* werden müssen, sie müssen auch *gefühlt* werden, d.h. spontan das richtige innere Bild auslösen. Solange Informatiker für sich die Begriffe immer noch gedanklich übersetzen müssen, ist die Kommunikation nicht optimal. Das ist meistens jedoch nur eine Frage der Zeit – d.h. die Übersetzung wird allmählich automatisiert, der Originalbegriff verbindet sich dann sofort mit dem richtigen inneren Bild.

Etwas anders stellt sich das Problem für deutschsprachige Publikationen. Gewöhnlich findet hier keine Kommunikation zwischen gleichwertigen Partnern statt, wie vielleicht im Projektalltag zwischen Informatikern. Hier werden vorrangig Informationen transportiert, die für den Leser neu sind – das Lernen und Verstehen steht im Vordergrund. Der Autor kann nicht davon ausgehen, daß der Empfänger der Information den gleichen Kontext und ein vergleichbares Hintergrundwissen hat. Deshalb sind treffende und einheitliche Übersetzungen für Fachbücher u.ä. ein wichtiges Thema.

Abgeleitetes Attribut ⇨ **Attribut**, ⇨ **Abgeleitetes Element**

Ein abgeleitetes Attribut wird aus den Werten anderer Attribute berechnet. Abgeleitete Attribute können nicht direkt geändert werden und werden durch eine Berechnungsoperation implementiert oder gesetzt.

Abgeleitete Assoziation ⇨ **Assoziation**, ⇨ **Abgeleitetes Element**

Eine abgeleitete Assoziation ist eine Assoziation, deren konkrete Objektbeziehungen jederzeit aus den Werten anderer Objektbeziehungen und ihrer Objekte abgeleitet (berechnet) werden können.

Abgeleitetes Element (UML: *derived element*)

Ein Modellelement, das jederzeit durch ein anderes Element berechnet werden kann und nur der Klarheit wegen gezeigt wird oder für Designzwecke zugefügt wird, ohne daß es jedoch weitere semantische Information zufügt.

Abhängigkeit (UML: *dependency*)

Eine Abhängigkeit ist eine Beziehung zwischen zwei Modellelementen, die zeigt, daß eine Änderung in dem einen (unabhängigen) Element eine Änderung in dem anderen (abhängigen) Element notwendig macht.

Abstraktion

Abstraktion ist eine Methode, bei der unter einem bestimmten Gesichtspunkt die wesentlichen Merkmale eines Gegenstandes oder Begriffes herausgesondert werden.

abstract class ⇨ **Abstrakte Klasse****Abstrakte Klasse** (UML: *abstract class*)

Von einer abstrakten Klasse werden niemals Objektexemplare erzeugt; sie ist bewußt unvollständig und bildet somit die Basis für weitere Unterklassen, die Exemplare haben können. C++: virtuelle Klasse.

Abstrakte Operation ⇨ **Operation**

Eine Operation, für die nur eine ⇨ Signatur, jedoch keine Anweisungsfolge definiert ist, d.h. die Operation ist definiert, aber noch nicht implementiert. Sie wird in einer abgeleiteten Klasse implementiert. C++: virtuelle Operation.

Abstrakter Datentyp (ADT)

Das Konzept des abstrakten Datentyps ähnelt dem der Klasse. Unter einem abstrakten Datentyp versteht man die Zusammenfassung von Daten und der mit ihnen ausführbaren Operationen.

actor ⇨ **Akteur****action state** ⇨ **Aktivität****activity diagram** ⇨ **Aktivitätsdiagramm****aggregation** ⇨ **Aggregation**

Aggregation (UML: *aggregation*) ⇔ Assoziation, ⇔ Komposition

Eine Aggregation ist eine Assoziation, erweitert um den semantisch unverbindlichen Kommentar, daß die beteiligten Klassen **keine gleichwertige** Beziehung führen, sondern eine Ganzes-Teile-Hierarchie darstellen. Eine Aggregation soll beschreiben, wie sich etwas Ganzes aus seinen Teilen logisch zusammensetzt.

Akteur (UML: *actor*)

Ein Akteur ist eine außerhalb des zu realisierenden Systems liegende Einheit, die an der in einem Anwendungsfall beschriebenen Interaktion mit dem System beteiligt ist. Ein Akteur kann ein Mensch sein, z.B. ein Benutzer, ebenso aber auch ein anderes technisches System, z.B. SAP, das Betriebssystem o.ä.

Aktive Klasse ⇔ Aktives Objekt

Eine Klasse, deren Instanzen **nebenläufig ausgeführt werden** und ihren eigenen Kontrollfokus (Thread) besitzen.

Aktives Objekt ⇔ Aktive Klasse

Instanz einer aktiven Klasse.

Aktivität (UML: *action state*)

Ein Zustand mit einer internen Aktion und einer oder mehreren ausgehenden Transitionen, die automatisch dem Abschluß der internen Aktion folgen. Eine Aktivität ist ein **einzelner Schritt in einem Ablauf**. Eine Aktivität kann mehrere ausgehende Transitionen haben, wenn diese durch Bedingungen unterschieden werden können.

Aktivitätsdiagramm (UML: *activity diagram*)

Ein Aktivitätsdiagramm ist eine spezielle Form des Zustandsdiagramms, das überwiegend oder ausschließlich ⇔ Aktivitäten enthält.

Analyse

Mit (objektorientierter) **Analyse werden alle Aktivitäten im Rahmen des Softwareentwicklungsprozesses bezeichnet, die der Ermittlung, Klärung und Beschreibung der Anforderungen an das System dienen** (d.h. die Klärung, *was* das System leisten soll).

Anforderung

Anforderungen sind Aussagen über zu erbringende Leistungen. Durch sie wird der Funktions- und Leistungsumfang eines Produktes, beispielsweise der zu erstellenden Software beschrieben.

Annotation ⇔ Notiz

Anwendungsfall (UML: *use case*)

Ein Anwendungsfall beschreibt eine Menge von Aktivitäten eines Systems aus der Sicht seiner Akteure, die für die Akteure zu einem wahrnehmbaren Ergebnis führen. Ein Anwendungsfall wird stets durch einen Akteur initiiert. Ein Anwendungsfall ist ansonsten eine komplette, unteilbare Beschreibung.

Anwendungsfalldiagramm (UML: *use case diagram*)

Ein Diagramm, das die Beziehungen zwischen \Rightarrow Akteuren und \Rightarrow Anwendungsfällen zeigt.

Anwendungsfallmodell (UML: *use case model*)

Ein Modell, das die funktionalen Anforderungen an ein System in Form von Anwendungsfällen beschreibt.

Anwendungskomponente \Rightarrow Komponente

Anwendungskomponenten sind fachliche \Rightarrow Subsysteme.

Architektur

ist die Spezifikation der grundlegenden Struktur eines Systems.

Argument

Konkreter Wert eines \Rightarrow Parameters.

assertion \Rightarrow Zusicherungen

Assertions sind boolesche Ausdrücke, die niemals unwahr werden sollten und anderenfalls auf Fehler hinweisen. Typischerweise sind *assertions* nur zur Entwicklungszeit aktiviert.

association \Rightarrow Assoziation**association class** \Rightarrow Assoziationsklasse**association role** \Rightarrow Assoziationsrolle**Assoziation** (UML: *association*) \Rightarrow gerichtete Assoziationen, \Rightarrow bidirektionale Assoziationen

Eine Assoziation beschreibt eine Relation zwischen Klassen, d.h. die gemeinsame Semantik und Struktur einer Menge von \Rightarrow Objektbeziehungen. Es werden \Rightarrow gerichtete Assoziationen (nur einseitig direkt navigierbar) und \Rightarrow bidirektionale Assoziationen (beidseitig direkt navigierbar) unterschieden. Die beiden Enden einer Assoziation sind \Rightarrow Assoziationsrollen.

Assoziationsklasse (UML: *association class*) \Rightarrow Attributierte Assoziation, \Rightarrow Aufgebrochene Assoziation, \Rightarrow Degenerierte Assoziationsklasse
 Ein Modellelement, das sowohl über die Eigenschaften einer Klasse als auch über die einer Assoziation verfügt. Es kann gesehen werden als Assoziation mit zusätzlichen Klasseneigenschaften (Attributierte Assoziation) oder als Klasse mit zusätzlichen Assoziations-eigenschaften (Assoziationsklasse).

Assoziationsrolle (UML: *association role*)

Die Rolle, die ein Typ oder eine Klasse in einer \Rightarrow Assoziation spielt. D.h. eine Rolle repräsentiert eine Klasse in einer Assoziation. Die Unterscheidung ist wichtig, da eine Klasse auch eine Assoziationsbeziehung zu sich selbst haben kann und in diesem Fall die beiden Enden der Assoziation nur durch ihre Rollenangabe unterschieden werden können.

attribute \Rightarrow Attribut

Attribut (UML: *attribute*)

Eine benannte Eigenschaft eines Typs. Ein Attribut ist ein Datenelement, das in jedem Objekt einer Klasse gleichermaßen enthalten ist und von jedem Objekt mit einem individuellen Wert repräsentiert wird. Im Gegensatz zu Objekten haben Attribute außerhalb des Objektes, von dem sie Teil sind, keine eigene Identität. Attribute sind vollständig unter der Kontrolle der Objekte, von denen sie Teil sind.

Attributierte Assoziation \Rightarrow Assoziation, \Rightarrow Assoziationsklasse

Eine Assoziation die über eigene Attribute verfügt.

Aufgebrochene Assoziation \Rightarrow Assoziation, \Rightarrow Assoziationsklasse

Eine \Rightarrow attributierte Assoziation, bei der die Attribute zu einer gewöhnlichen Klasse überführt und die attributierte Assoziation unter Einbeziehung dieser neuen Klasse in zwei gewöhnliche Assoziationen umgeformt wurde.

Basisklasse \Rightarrow Oberklasse

Behälterklasse \Rightarrow Sammlung

Beziehung (UML: *relationship*)

Eine Verbindung zwischen Modellelementen mit semantischem Gehalt. Oberbegriff für \Rightarrow Assoziation, \Rightarrow Aggregation, \Rightarrow Komposition, \Rightarrow Generalisierung und \Rightarrow Spezialisierung.

Bidirektionale Assoziation \Rightarrow Assoziation

Eine bidirektionale Assoziation ist eine beidseitig direkt navigierbare Assoziation, d.h. eine Assoziation, bei der von beiden beteiligten \Rightarrow Assoziationsrollen zur jeweils anderen direkt navigiert werden kann.

Bindung \Rightarrow Dynamische Bindung

Botschaft ⇒ Nachricht

bound element ⇒ Gebundenes Element

Businessklasse ⇒ Klasse

⇒ Instanziiert ⇒ Businessobjekte.

Businessmodell (Businessklassenmodell) ⇒ Fachklassenmodell

Ein Businessmodell ist ein Klassenmodell, daß ausschließlich oder vorwiegend ⇒ Businessklassen (fachlich elementare Begriffe) in Form von Klassen enthält.

Businessobjekt ⇒ Objekt

Ein Businessobjekt repräsentiert einen Gegenstand, ein Konzept, einen Ort oder eine Person aus dem realen Geschäftsleben in einem fachlichen geringen Detaillierungsgrad, d.h. einen fachlich elementaren Begriff (Vertrag, Rechnung etc.). Für die praktische Umsetzung sind Businessobjekte auf rein fachlich motivierte Eigenschaften reduzierte Aggregationen fundamentaler Fachobjekte (⇒ Fachklassen: Rechnungspositionen, Anschrift etc.), zu denen alles weitere delegiert wird. Sie definieren typischerweise vor allem Schnittstellen und sind eine Art Fassade.

class ⇒ Klasse

class diagram ⇒ Klassendiagramm

collaboration ⇒ Kollaboration

collaboration diagram ⇒ Kollaborationsdiagramm

collection ⇒ Sammlung

component ⇒ Komponente

component diagram ⇒ Komponentendiagramm

composition ⇒ Komposition

concrete class ⇒ Konkrete Klasse

concurrency ⇒ Nebenläufigkeit

constraint ⇒ Zusicherung

Containerklasse ⇒ Behälterklasse

CRC-Karten (Klassenkarte)

Karteikarten, auf denen der Name der Klasse (Class), ihre Aufgaben (Responsibilities) und ihre Beziehungen (Collaborations) beschrieben werden.

Datenabstraktion

Hierunter versteht man das Prinzip, nur die auf ein Objekt anwendbaren Operationen nach außen sichtbar zu machen. Die tatsächliche innere Realisierung der Operationen und die innere Struktur des Objektes werden verborgen, d.h. man betrachtet abstrakt die eigentliche Semantik und läßt die tatsächliche Implementierung außer acht.

Default-Implementierung ⇒ Standard-Implementierung

Degenerierte Assoziationsklasse

Die (namenlose) ⇒ Assoziationsklasse einer ⇒ attribuierten Assoziation.

Delegation

ist ein Mechanismus, bei dem ein Objekt eine Nachricht nicht (vollständig) selbst interpretiert, sondern an ein anderes Objekt weiterleitet (pro-pagiert).

dependency ⇒ Abhängigkeit

deployment diagram ⇒ Verteilungsdiagramm

derived element ⇒ Abgeleitetes Element

Design

Mit (objektorientiertem) Design werden alle Aktivitäten im Rahmen des Softwareentwicklungsprozesses bezeichnet, mit denen ein Modell logisch und physisch strukturiert wird und die dazu dienen zu beschreiben, wie das System die in der ⇒ Analyse beschriebenen Anforderungen erfüllt.

Diskriminator

Ein Diskriminator ist ein Unterscheidungsmerkmal für die Strukturierung der Modellsemantik in ⇒ Generalisierungs- bzw. Spezialisierungsbeziehungen.

domain ⇒ Problembereich

Domäne ⇒ Problembereich

Domänenmodell ⇒ Fachklassenmodell

Dynamische Bindung, Späte Bindung

Hierunter ist zu verstehen, daß eine ⇒ Nachricht erst zur Programmaufzeit einer konkreten ⇒ Operation zugeordnet wird, die diese Nachricht dann interpretiert.

Dynamische Klassifikation

Ein Objekt ist nacheinander Instanz unterschiedlicher Klassen einer Untertypenstruktur, d.h. es kann seine Klassenzugehörigkeit während seiner Lebenszeit ändern.

Eigenschaftswert (UML: *property, tagged value*)

Eigenschaftswerte sind benutzerdefinierte, sprach- und werkzeugspezifische Schlüsselwort-Wert-Paare (*tagged values*), die die Semantik einzelner Modellelemente um spezielle charakteristische Eigenschaften erweitern. Der Unterschied zum \Rightarrow Stereotyp besteht darin, daß durch ein Stereotyp das Metamodell um ein neues Element erweitert wird. Mit Eigenschaftswerten hingegen können einzelne Ausprägungen bestehender Modellelemente (z.B. eine bestimmte Operation) um bestimmte Eigenschaften erweitert werden.

Einfachvererbung \Rightarrow Vererbung

Bei der Einfachvererbung erbt eine Unterklasse nur von einer direkten Oberklasse.

Einsatzdiagramm \Rightarrow Verteilungsdiagramm**Einschränkung** \Rightarrow Zusicherung**Entwurfsmuster**

Entwurfsmuster sind generalisierte Lösungsideen zu immer wiederkehrenden Entwurfsproblemen. Sie sind keine fertig codierten Lösungen, sie beschreiben lediglich den Lösungsweg.

Ereignis

Ein Geschehen, das in einem gegebenen Kontext eine Bedeutung hat und sich räumlich und zeitlich lokalisieren läßt.

Exemplar \Rightarrow Objekt, \Rightarrow Instanz

Für den Hausgebrauch können Instanz, Objekt und Exemplar synonym betrachtet werden. In der UML 1.0 finden sich jedoch teilweise inkonsistente Akzentuierungen.

event \Rightarrow Ereignis*framework* \Rightarrow Rahmenwerk**Fachklasse** \Rightarrow Klasse

Gewöhnliche fachlich motivierte Klasse, die einen Begriff aus dem Problembereich repräsentiert. Wird zur Unterscheidung von \Rightarrow Businessklassen verwendet.

Fachklassenmodell \Rightarrow Domänenmodell

Ein Klassenmodell, das ausschließlich oder vorwiegend Fachklassen enthält.

Fundamentalklasse \Rightarrow Fachklasse**Generische Klasse** \Rightarrow Parametrisierbare Klasse*generalization* \Rightarrow Generalisierung**Gebundenes Element** \Rightarrow Parametrisierte Klasse

Generalisierung (UML: generalization) ⇨ Spezialisierung /Konkretisierung

Generisches Design

Anwendung von Schablonen oder Makros zum Design (in CASE-Tools).

Generische Programmierung

Anwendung von Schablonen (engl. *templates*), ⇨ parametrisierbaren Klassen u.ä. bei der Programmierung.

Geordnete Assoziation ⇨ Assoziation

Assoziation, bei der die Objektverbindungen in bestimmter Weise geordnet sind.

Gerichtete Assoziation ⇨ Assoziation, ⇨ Navigation

Eine Assoziation, bei der von der einen beteiligten ⇨ Assoziationsrollen zur anderen direkt navigiert werden kann, **nicht aber umgekehrt**.

Geschäftsobjekt ⇨ Businessobjekt

Geschäftsprozeß ⇨ Workflow

Ein Geschäftsprozeß ist eine Zusammenfassung von organisatorisch evtl. verteilten, fachlich jedoch zusammenhängenden Aktivitäten, die notwendig sind, um einen Geschäftsvorfall (z.B. einen konkreten Antrag) ergebnisorientiert zu bearbeiten. Die Aktivitäten eines Geschäftsprozesses stehen gewöhnlich in zeitlichen und logischen Abhängigkeiten zueinander. Ein Geschäftsvorfall entsteht gewöhnlich durch ein Ereignis (z.B. Antragseingang).

Geschäftsvorfall (Vorgang)

Ein Geschäftsvorfall ist ein (Geschäfts-) Objekt (z.B. ein konkreter Vertrag), das durch ein Ereignis ausgelöst (z.B. Antragseingang) durch die innerhalb eines Geschäftsprozesses beschriebenen Aktivitäten bearbeitet wird.

GUI

Graphical User Interface, Grafische Benutzeroberfläche

Hilfsklasse (UML: *utility*)

Hilfsklassen sind Sammlungen von globalen Variablen und Funktionen, die zu einer Klasse zusammengefaßt und dort als Klassenattribute und -operationen definiert sind. Insofern sind Hilfsklassen keine echten Klassen. Das ⇨ Stereotyp «*utility*» kennzeichnet eine Klasse als Hilfsklasse.

Information Hiding

ist das bewußte Verbergen von Implementierungsdetails. Das heißt, nach außen wird eine Schnittstelle bereitgestellt, das Innere (z.B. einer Klasse) ist aber nicht sichtbar. Dadurch bleibt verborgen, wie die Schnittstelle intern bedient wird.

Identität ⇨ Objektidentität

inheritance ⇒ Vererbung

instance ⇒ Instanz

Instanz (UML: *instance*) ⇒ Exemplar

Instantiierung

ist das Erzeugen eines Exemplars aus einer Klasse.

interface ⇒ Schnittstelle

interaction diagram ⇒ Interaktionsdiagramm

Interaktionsdiagramm (UML: *interaction diagram*)

Sammelbegriff für ⇒ Sequenzdiagramm, ⇒ Kollaborationsdiagramm, ⇒ Aktivitätsdiagramm.

Invariante

Eine Eigenschaft oder ein Ausdruck, der über den gesamten Lebenszeitraum eines Elementes, bspw. eines Objektes gegeben sein muß.

Kardinalität ⇒ Multiplizität

Anzahl der Elemente.

Klasse (UML: *class*)

Eine Klasse ist die Definition der Attribute, Operationen und der Semantik für eine Menge von Objekten. Alle Objekte einer Klasse entsprechen dieser Definition.

Klassenattribut, Klassenvariable ⇒ Attribut

Klassenattribute gehören nicht einem einzelnen Objekt, sondern sind Attribut einer Klasse (gilt z.B. für Smalltalk).

Klassenoperation, Klassenmethode ⇒ Operation

Klassenoperationen sind Operationen, die nicht auf einem Objekt, sondern auf einer Klasse operieren (gilt z.B. für Smalltalk).

Klassenbibliothek

Eine Klassenbibliothek ist eine Sammlung von Klassen.

Klassendiagramm (UML: *class diagram*)

Ein Klassendiagramm zeigt eine Menge statischer Modellelemente, vor allem Klassen und ihre Beziehungen.

Klassenkarte ⇒ CRC-Karte

Klassenvorlage ⇒ Parametrisierbare Klasse

Klassenschablone ⇒ Parametrisierbare Klasse

Konfigurationsdiagramm ⇒ Einsatzdiagramm

Knoten (UML: *node*)

Ein Knoten ist ein physisches Laufzeit-Objekt, das über Rechnerleistung (Prozessor, Speicher) verfügt. Laufzeitobjekte und **Komponenten** können auf Knoten residieren.

Konkretisierung ⇒ Spezialisierung

Kollaboration (UML: *collaboration*) ⇒ Kollaborationsdiagramm

Eine Kollaboration ist der Kontext einer Menge von Interaktionen.

Kollaborationsdiagramm (UML: *collaboration diagram*)

Eine Kollaborationsdiagramm zeigt eine Menge von Interaktionen zwischen einer Menge **ausgewählter Objekte in einer bestimmten begrenzten Situation** (Kontext) unter Betonung der Beziehungen zwischen den Objekten und ihrer **Topographie**. **Ähnlich dem** ⇒ Sequenzdiagramm.

Komponente (UML: *component*)

Eine Komponente ist eine ausführbare und austauschbare Softwareeinheit mit definierten Schnittstellen und eigener Identität. Es sind Komponentendefinitionen (z.B. „Person“) und **Komponenteninstanzen** (z.B. „Gabi Goldfisch“) zu unterscheiden.

Komponentendiagramm (UML: *component diagram*)

Ein Komponentendiagramm zeigt die Organisation und Abhängigkeiten von ⇒ Komponenten.

Komposition (UML: *composite*) ⇒ Aggregation

Eine Komposition ist eine strenge Form der Aggregation, bei der die Teile vom **Ganzen existenzabhängig sind**. Sie beschreibt, wie sich etwas Ganzes aus Einzelteilen zusammensetzt und diese kapselt.

Konkrete Klasse (UML: *concrete class*)

Eine ⇒ Klasse, die ⇒ Objekte instantiiieren kann. Vgl. ⇒ Abstrakte Klasse.

Konsistenzsicherung ⇒ Zusicherung

Eine Zusicherung zwischen **mehreren Assoziationen**, die teilweise redundante Sachverhalte repräsentieren. Die Zusicherung gibt die Konsistenzbedingung an.

link ⇒ Objektbeziehung

Mehrfachvererbung ⇒ Multiple Vererbung

Mehrgliedrige Assoziation (UML: *n-ary association*)

Eine ⇒ Assoziation, an der **mehr als zwei** ⇒ Assoziationsrollen beteiligt sind.

message ⇒ Nachricht

meta class ⇒ Metaklasse

meta model ⇒ Metamodell

Metaklasse (UML: *meta class*)

Eine Metaklasse ist eine Klasse, deren Exemplare wiederum Klassen sind. Dieses Konzept existiert nur in einigen objektorientierten Sprachen (z.B. in Smalltalk).

Metamodell (UML: *meta model*)

Ein Modell, das die Sprache definiert, mit der ein Modell definiert werden kann.

Metatyp (UML: *powertype*)

Ein Metatyp ist ein Typ (eine Klasse), dessen Instanzen Untertypen (Unterklassen) eines anderen Typs (einer anderen Klasse) sind.

Merkmal ⇒ Eigenschaftswert

method ⇒ Methode

Methode (UML: *method*) ⇒ Operation

In Smalltalk werden Operationen Methoden genannt. In der UML wird eine Methode als Implementierung einer Operation definiert. Für die Praxis ist es unkritisch, Methode und Operation synonym zu verwenden.

Multiple Klassifikation

Ein Objekt ist zur gleichen Zeit Instanz mehrerer Klassen (nicht möglich in C++, Java und Smalltalk)

Multiple Vererbung

Eine Klasse hat mehrere direkte Oberklassen (nicht möglich in Java und Smalltalk).

Multiplizität

Bereich erlaubter ⇒ Kardinalitäten.

n-ary association ⇒ Mehrgliedrige Assoziation

Nachbedingung

Eine Nachbedingung beschreibt einen Zustand, der nach Abschluß einer Operation o.ä. gegeben sein muß.

Nachricht (UML: *message*) ⇒ Operation, ⇒ Methode

Nachrichten sind ein Mechanismus, mit dem Objekte untereinander kommunizieren können. Eine Nachricht überbringt einem Objekt die Information darüber, welche Aktivität von ihm erwartet wird, d.h. eine Nachricht fordert ein Objekt zur Ausführung einer Operation auf. Eine Nachricht besteht aus einem Selektor (einem Namen), einer Liste von Argumenten und geht an genau einen Empfänger. Der Sender einer Nachricht erhält ggf. ein Antwort-Objekt zurück. Durch ⇒ Polymorphismus kann eine Nachricht zum Aufruf einer von mehreren gleichlautenden ⇒ Operationen führen.

navigability ⇨ Navigierbarkeit

Navigation, Navigierbarkeit ⇨ Navigationsangaben

Navigation ist die Betrachtung von Zugriffsmöglichkeiten auf Objekte (und ihre Attribute und Operationen) innerhalb eines Objektnetzes. *Direkt navigierbar* werden solche Zugriffe genannt, die ohne Umwege möglich sind.

Navigationsangaben

Navigationsangaben sind Spezifikationen zur Navigation, d.h. Beschreibungen von Zugriffspfaden und -einschränkungen und der daraus resultierenden Zugriffsergebnisse (beispielsweise mit Hilfe der ⇨OCL)

Nebenläufigkeit

Zwei oder mehr Aktivitäten werden zeitgleich (parallel) ausgeführt.

node ⇨ Knoten

Notiz (UML: *note*)

Kommentare bzw. Annotationen zu einem Diagramm oder einem oder mehreren beliebigen Modellelementen ohne semantische Wirkung.

Oberklasse, Superklasse (UML: *superclass*) ⇨ Generalisierung

Eine Oberklasse ist eine Verallgemeinerung ausgewählter Eigenschaften ihrer ⇨Unterklassen(n).

Oberzustand (UML: *superstate*) ⇨ Zustand

Ein Oberzustand enthält andere Zustände bzw. Unterzustände.

object diagram ⇨ Objektdiagramm

Objekt (UML: *object*)

Ein Objekt ist eine konkret vorhandene und agierende Einheit mit eigener Identität und definierten Grenzen das Zustand und Verhalten kapselt. Der Zustand wird repräsentiert durch die ⇨Attribute und ⇨Beziehungen, das Verhalten durch ⇨Operationen bzw. ⇨Methoden. Jedes Objekt ist Exemplar (Synonym: Instanz) einer Klasse. Das definierte Verhalten gilt für alle Objekte einer Klasse gleichermaßen, ebenso die Struktur ihrer Attribute. Die Werte der Attribute sind jedoch individuell für jedes Objekt. Jedes Objekt hat eine eigene, von seinen Attributen u.a. unabhängige, nicht veränderbare Identität.

Objektbasiert

Eine Programmiersprache oder Datenbank wird als objektbasiert bezeichnet, wenn sie das Konzept der Datenabstraktion unterstützt, weitergehende Konzepte wie Klassen, Vererbung, Polymorphie etc. aber teilweise oder vollständig fehlen.

Objektbeziehung (UML: *link*)

Eine konkrete Beziehung zwischen zwei Objekten, d.h. die Instanz einer \Rightarrow Assoziation. Ein Objekt hat eine Beziehung zu einem anderen Objekt, wenn es eine Referenz darauf besitzt. Implementiert werden diese Referenzen gewöhnlich durch \Rightarrow Attribute, was für die Modellierung jedoch unerheblich ist.

Objektdiagramm (UML: *object diagram*)

Ein Diagramm, das Objekte und ihre **Beziehungen** untereinander zu einem bestimmten Zeitpunkt zeigt. Gewöhnlich ein \Rightarrow Kollaborationsdiagramm oder eine spezielle Variante des \Rightarrow Klassendiagramms.

Objektidentität

ist eine **Eigenschaft**, die ein Objekt von allen anderen unterscheidet, auch wenn es möglicherweise die gleichen Attributwerte besitzt.

Objektorientierte Programmiersprache

Objektorientierte Programmiersprachen erfüllen folgende **Basiskonzepte**:

- Objekte sind abstrakte Einheiten,
- Objekte sind Exemplare einer Klasse, d.h. von einer Klasse abgeleitet,
- die Klassen vererben ihre Eigenschaften und bilden so eine **Vererbungshierarchie**,
- auf Objekte wird dynamisch verwiesen, d.h. die Bindung ist dynamisch und ermöglicht so **Polymorphie**.

OCL, Object Constraint Language

Die OCL definiert eine Sprache zu Beschreibung von \Rightarrow Zusicherungen, \Rightarrow Invarianten, \Rightarrow Vor- und Nachbedingungen und \Rightarrow Navigation innerhalb von UML-Modellen.

Oder-Zusicherung \Rightarrow Zusicherung

Eine Zusicherung zwischen **Assoziationen**, die alle von einer gemeinsamen Klasse zu verschiedenen anderen führen. Es wird definiert, daß die Objekte der **gemeinsamen Klasse stets nur** \Rightarrow Objektverbindungen zu genau einer der übrigen Klassen (präziser formuliert: \Rightarrow Assoziationsrolle) unterhalten (exklusives Oder).

OO

ist die Abkürzung für Objektorientierung.

operation \Rightarrow Operation**Operation** (UML: *operation*) \Rightarrow Methode, \Rightarrow Nachricht

Operationen sind Dienstleistungen, die von einem Objekt mit einer \Rightarrow Nachricht angefordert werden können, um ein bestimmtes Verhalten zu bewirken. Sie werden implementiert durch \Rightarrow Methoden. In der Praxis werden Operation und Methode häufig synonym verwendet.

Ordnungszusicherung ⇒Zusicherung

Eine Zusicherung zu einer Assoziation, die angibt, daß ihre Elemente (⇒Objektverbindungen) in bestimmter Weise geordnet sind.

package ⇒Paket

Paket (UML: *package*)

Pakete sind Ansammlungen von Modellelementen beliebigen Typs, mit denen das Gesamtmodell in kleinere überschaubare Einheiten gegliedert wird. Ein Paket definiert einen Namensraum, d.h. innerhalb eines Paketes müssen die Namen der enthaltenen Elemente eindeutig sein. Jedes Modellelement kann in anderen Paketen referenziert werden, gehört aber zu genau einem (Heimat-) Paket. Pakete können wiederum Pakete beinhalten. Das oberste Paket beinhaltet das gesamte System.

parameter ⇒Parameter

Parameter (UML: *parameter*)

Ein Parameter ist die Spezifikation einer Variablen, die Operationen, Nachrichten oder Ereignissen mitgegeben, von diesen verändert oder zurückgegeben wird. Ein Parameter kann aus einem Namen, einem Typ (einer Klasse) und einer Übergaberichtung (in, out, inout) bestehen.

parameterized class ⇒Parametrisierbare Klasse

Parametrisierbare Klasse (UML: *parameterized class*)

Eine parametrisierbare Klasse ist eine mit generischen formalen Parametern versehene Schablone, mit der gewöhnliche (d.h. nicht-generische) Klassen erzeugt werden können. Die generischen Parameter dienen als Stellvertreter für die aktuellen Parameter, die Klassen oder einfache Datentypen repräsentieren.

Parametrisierte Klasse ⇒ Parametrisierbare Klasse

Als parametrisierte Klasse wird die Instanz einer ⇒Parametrisierbaren Klasse bezeichnet, d.h. das Ergebnis einer konkreten Parametrisierung.

Parameterliste

Aufzählung der Namen von Argumenten sowie ggf. ihres Typs, Initialwertes u.ä.

Partition ⇒Diskriminator

Eine Partition ist die Gesamtheit der Subklassen, die auf dem selben Diskriminator beruhen.

pattern, design pattern ⇒Entwurfsmuster

Persistentes Objekt

Persistente Objekte (persistent: lat. „anhaltend“) sind solche, deren Lebensdauer über die Laufzeit einer Programmsitzung hinausreicht. Die Objekte werden hierzu auf nichtflüchtigen Speichermedien (z.B. Datenbanken) gehalten.

Polymorphismus

Polymorphismus (Vielgestaltigkeit) heißt, daß gleichlautende Nachrichten an kompatible Objekte unterschiedlicher Klassen ein unterschiedliches Verhalten bewirken können. Beim dynamischen Polymorphismus wird eine Nachricht nicht zur Compilierzeit, sondern erst beim Empfang zur Programmlaufzeit einer konkreten Operation zugeordnet. Voraussetzung hierfür ist das dynamische Binden.

powertype ⇒ Metatyp

Problembereich ⇒ Domäne

Anwendungsgebiet bzw. Problembereich, innerhalb dessen die fachliche Modellierung stattfindet. Als Problembereichsmodell (Domänenmodell) wird in der Regel der Teil des Gesamtmodells verstanden, der sich auf den eigentlichen fachlichen Problembereich bezieht (auch fachliches Modell genannt). Technische, querschnittliche u.ä. Aspekte gehören nicht dazu. Im Kontext von Anwendungsarchitektur ist zumeist das fachliche Klassenmodell gemeint (d.h. ohne Framework-, GUI-, Controller- u.ä. Klassen).

Propagation ⇒ Delegation

Ausdehnung der Eigenschaften einer Klasse durch Verwendung von Operationen anderer Klassen.

property ⇒ Eigenschaftswert

Protokoll

Eine Menge von Signaturen.

qualifier ⇒ Qualifizierendes Attribut

Qualifizierendes Attribut (UML: *qualifier*) ⇒ Qualifizierte Assoziation

Das Attribut, über welches in einer Assoziation der Zugriff auf die gegenüberliegende Seite erfolgt. Das qualifizierende Attribut ist definiert als Teil der Assoziation, jedoch muß in der Klasse, auf die darüber zugegriffen wird, dieses Attribut definiert sein.

Qualifizierte Assoziation ⇒ Assoziation, ⇒ Qualifizierendes Attribut

Eine qualifizierte Assoziation ist eine Assoziation, bei der die referenzierte Menge der Objekte durch qualifizierende Attribute in Partitionen unterteilt wird, wobei vom Ausgangsobjekt aus betrachtet jede Partition nur einmal vorkommen kann.

Rahmenwerk (UML: *framework*)

Ein Rahmenwerk ist eine Menge kooperierender Klassen, die unter Vorgabe eines Ablaufes („*Don't call the framework, the framework calls you*“) eine generische Lösung für eine Reihe ähnlicher Aufgabenstellungen bereitstellen.

Referentielle Integrität

Regel, die die Integrität von Objektbeziehungen beschreibt, vor allem für den Fall, daß eines der beteiligten Objekte oder die Objektverbindung selbst gelöscht werden sollen.

refinement ⇒ Verfeinerungsbeziehung

relationship ⇒ Beziehung

Rolle ⇒ Assoziationsrolle

Sammlung (UML: *collection*)

Sammlungen sind Objekte, die eine Menge anderer Objekte referenzieren und die Operationen bereitstellen, um auf diese Objekte zuzugreifen.

scenario ⇒ Szenario

Schablone ⇒ Parametrisierbare Klasse

Schnittstelle (UML: *interface*) ⇒ Schnittstellenklassen

Schnittstellen beschreiben einen ausgewählten Teil des extern sichtbaren Verhaltens von Modellelementen (hauptsächlich von Klassen und Komponenten), d.h. eine Menge von Signaturen.

Schnittstellenklassen

Schnittstellenklassen sind ⇒ abstrakte Klassen (genauer: Typen), die ausschließlich ⇒ abstrakte Operationen definieren. Schnittstellenklassen sind Klassen, die mit dem ⇒ Stereotyp «*interface*» gekennzeichnet sind. Sie sind Spezifikationen des extern sichtbaren Verhaltens von Klassen und beinhalten eine Menge von ⇒ Signaturen für Operationen, die Klassen, die diese Schnittstelle bereitstellen wollen, implementieren müssen.

Schnittstellenvererbung

Innerhalb einer ⇒ Spezialisierungsbeziehung wird lediglich eine ⇒ Schnittstelle vererbt.

Selbstdelegation

Zur Ausführung einer Operation wird eine Teilaufgabe an eine andere Operation der selben Klasse delegiert (d.h. ein Objekt sendet sich selbst eine Nachricht).

Self

Self (Smalltalk) und *this* (Java, C++) sind vordefinierte Programmiersprachen-Schlüsselwörter. Mit *this* bzw. *self* kann sich ein Objekt selbst eine Nachricht senden, d.h. es ruft eine andere seiner eigenen Methoden auf. Nachrichten, die ein Objekt mit *this* bzw. *self* an sich selbst sendet, werden genauso behandelt, wie solche von außen. Ebenso kann damit auf Attribute der eigenen Klasse zugegriffen werden.

sequence diagram ⇒ Sequenzdiagramm

Sequenzdiagramm (UML: *sequence diagram*)

Eine Sequenzdiagramm zeigt eine Menge von Interaktionen zwischen einer Menge ausgewählter Objekte in einer bestimmten begrenzten Situation (**Kontext**) unter **Betonung der zeitlichen Abfolge**. Ähnlich dem ⇒ Kollaborationsdiagramm. Sequenzdiagramme können in generischer Form existieren (Beschreibung aller möglichen Szenarien) oder in Instanzform (Beschreibung genau eines speziellen ⇒ Szenarios).

Spezialisierung, Generalisierung ⇒ Vererbung

Eine Generalisierung (bzw. Spezialisierung) ist eine taxonomische Beziehung zwischen einem allgemeinen und einem speziellen Element (bzw. umgekehrt), wobei **das speziellere weitere Eigenschaften hinzufügt**, die Semantik erweitert und sich kompatibel zum allgemeinen verhält. Generalisierung und Spezialisierung sind Abstraktionsprinzipien zur hierarchischen Strukturierung der Modellsemantik unter einem diskriminierenden Aspekt (⇒ Diskriminator).

Sichtbarkeitskennzeichen

schränken die **Zugreifbarkeit** von Attributen und Operationen ein (*private, protected, public* etc.).

Signatur

Die Signatur einer Operation setzt sich zusammen aus dem **Namen der Operation**, ihrer Parameterliste und der Angabe eines evtl. Rückgabetypes.

Standard-Implementierung

Konkrete Implementierung einer eigentlich abstrakten Operation, um für Subklassen ein Standardverhalten bereitzustellen.

state ⇒ Zustand

state diagram ⇒ Zustandsdiagramm

Statische Klassifikation

Ein Objekt ist und bleibt Instanz genau einer Klassen, d.h. es kann seine Klassenzugehörigkeit während seiner Lebenszeit nicht ändern. Vgl. ⇒ Dynamische Klassifikation.

stereotype ⇒ Stereotyp

Stereotyp (UML: *stereotype*)

Stereotypen sind projekt-, **unternehmens-** oder methodenspezifische Erweiterungen vorhandener Modellelemente des UML-Metamodells. Entsprechend der mit der Erweiterung definierten Semantik wird das Modellierungselement, auf das es angewendet wird, direkt semantisch beeinflusst. **In der Praxis geben Stereotypen vor allem die möglichen Verwendungszusammenhänge einer Klasse, einer Beziehung oder eines Paketes an.** Andere erweiternde Mechanismen in der UML sind \Rightarrow Eigenschaftswerte und \Rightarrow Zusicherungen. (Duden: *das Stereotyp*).

subclass \Rightarrow Unterklasse

Subklasse (UML: *subclass*) \Rightarrow Unterklasse

Subsystem \Rightarrow Komponente

Ein Subsystem ist ein Teil eines Gesamtsystems, daß **nach außen durch eine Schnittstelle** definiert ist und das die Struktur und die Zusammenarbeit seiner Einzelteile versteckt.

super

Super (Smalltalk, Java) ist ein Programmiersprachen-Schlüsselwort. Es bewirkt, daß die Nachricht immer an die nächsthöhere Klasse geht, die über die genannte **Operation** verfügt.

superclass \Rightarrow Oberklasse

Superklasse (UML: *superclass*) \Rightarrow Oberklasse

superstate \Rightarrow Superzustand

swimlane \Rightarrow Verantwortungsbereich

Szenario (UML: *scenario*)

Ein Szenario ist eine spezifische Folge von Aktionen. Beispielsweise ein konkreter **Ablaufpfad in einem Anwendungsfall** (sozusagen eine Instanz des Anwendungsfalls). Vgl. \Rightarrow Sequenzdiagramm.

tagged value \Rightarrow Eigenschaftswert

template \Rightarrow Parametrisierbare Klasse

Ternäre Assoziation \Rightarrow Mehrgliedrige Assoziation

Eine Assoziation, an der drei Assoziationsrollen beteiligt sind.

this \Rightarrow self

type \Rightarrow Typ

Typ (UML: *type*)

Definition einer Menge von Operationen und Attributen. Andere Elemente sind **typkonform**, wenn sie über die durch den Typen definierten Eigenschaften verfügen. Wird in der Praxis häufig **gleichgesetzt** mit der Beschreibung von \Rightarrow Schnittstellen.

Transition (UML: *transition*)

Eine Transition ist ein Zustandsübergang (\Rightarrow Zustand), häufig ausgelöst durch ein \Rightarrow Ereignis.

Ungerichtete Assoziation \Rightarrow Bidirektionale Assoziation**Unidirektionale Assoziation** \Rightarrow Gerichtete Assoziation**Unterklasse, Subklasse** (UML: *subclass*)

Eine Unterklasse ist die Spezialisierung einer Oberklasse und erbt alle Eigenschaften der Oberklasse.

Untermengenzusicherung \Rightarrow Zusicherung

Eine Zusicherung/Abhängigkeit zwischen zwei Assoziationen. Die Elemente (\Rightarrow Objektverbindungen) der einen Assoziationen müssen Teil der Elemente der anderen Assoziation sein.

use case \Rightarrow Anwendungsfall**utility** \Rightarrow Hilfsklasse**Verantwortlichkeit**

umfasst die Attribute und die interpretierbaren Nachrichten eines Objektes.

Verantwortlichkeitsbereich (UML: *swimlane*)

Durch Linien getrennte Bereiche in \Rightarrow Aktivitätsdiagrammen, die die Verantwortlichkeit der im Diagramm enthaltenen Elemente beschreiben.

Vererbung (UML: *inheritance*) \Rightarrow Einfachvererbung, \Rightarrow Multiple Vererbung, \Rightarrow Multiple Klassifikation, \Rightarrow Dynamische Klassifikation

Vererbung ist ein Programmiersprachenkonzept für die Umsetzung einer Relation zwischen einer Ober- und einer Unterklasse, wodurch Unterklassen die Eigenschaften ihrer Oberklassen mitbenutzen können. Vererbung implementiert normalerweise \Rightarrow Generalisierungs- und Spezialisierungsbeziehungen. Alternativen: \Rightarrow Delegation, \Rightarrow Aggregation, \Rightarrow generische Programmierung, \Rightarrow generisches Design.

Verfeinerungsbeziehung (UML: *refinement*)

Verfeinerungsbeziehungen sind Beziehungen zwischen gleichartigen Elementen unterschiedlichen Detaillierungs- bzw. Spezifikationsgrades. Verfeinerungsbeziehungen sind Stereotypisierungen von \Rightarrow Abhängigkeitsbeziehungen.

Verteilungsdiagramm (UML: *deployment diagram*)

Ein Diagramm, welches die Konfiguration der zur Laufzeit vorhandenen (eingesetzten) \Rightarrow Knoten und ihrer \Rightarrow Komponenten, Prozesse und Objekte zeigt.

Virtuelle Klasse \Rightarrow Abstrakte Klasse**Virtuelle Operation** \Rightarrow Abstrakte Operation

Vorbedingung

Eine Vorbedingung beschreibt einen Zustand, der vor dem Ablauf einer Operation o.ä. gegeben sein muß.

Workflow ⇨ Geschäftsprozeß

Ein Workflow ist die computergestützte Automatisierung und Unterstützung eines Geschäftsprozesses oder eines Teils davon.

Workflow-Engine

Die Workflow-Engine ist eine Software, die Workflows steuert. Sie erzeugt, aktiviert, suspendiert und terminiert Workflow-Instanzen (d.h. die computergestützte Manifestation eines Geschäftsvorfalles).

Workflow-Instanz

Eine Workflow-Instanz ist die computergestützte Manifestation eines Geschäftsvorfalles; sie wird durch eine Workflow-Engine gesteuert.

Zustand (UML: *state*)

ist eine Abstraktion der möglichen Attributwerte eines Objektes. Ein Zustand gehört zu genau einer Klasse und stellt eine Abstraktion bzw. Zusammenfassung einer Menge von möglichen Attributwerten dar, die die Objekte dieser Klasse einnehmen können. In der UML ist ein Zustand eine Bedingung bzw. Situation im Leben eines Objektes, während der eine bestimmte Bedingung erfüllt ist, Aktivitäten ausgeführt werden oder auf ein Ereignis gewartet wird.

Zustandsdiagramm (UML: *state diagram, state machine*)

Ein Zustandsdiagramm zeigt eine Folge von Zuständen, die ein Objekt im Laufe seines Lebens einnehmen kann und aufgrund welcher Stimuli Zustandsänderungen stattfinden. Ein Zustandsdiagramm beschreibt eine hypothetische Maschine (Endlicher Automat), die sich zu jedem Zeitpunkt in einer Menge endlicher Zustände befindet. Sie besteht aus:

- einer endlichen, nicht-leeren Menge von Zuständen;
- einer endlichen, nicht-leeren Menge von Eingabesymbolen (Ereignissen);
- Funktionen, die den Übergang von einem Zustand in den nächsten beschreiben;
- einen Anfangszustand und
- einer Menge von Endzuständen.

Zusicherung (UML: *constraint*)

Eine Zusicherung ist ein Ausdruck, der die möglichen Inhalte, Zustände oder die Semantik eines Modellelementes einschränkt und der stets erfüllt sein muß. Bei dem Ausdruck kann es sich um ein ⇨Stereotyp oder ⇨Eigenschaftswerte handeln, um eine freie Formulierung (⇨Notiz) oder um eine ⇨Abhängigkeitsbeziehung. Zusicherungen in Form reiner boolescher Ausdrücke werden auch ⇨*assertions* genannt.