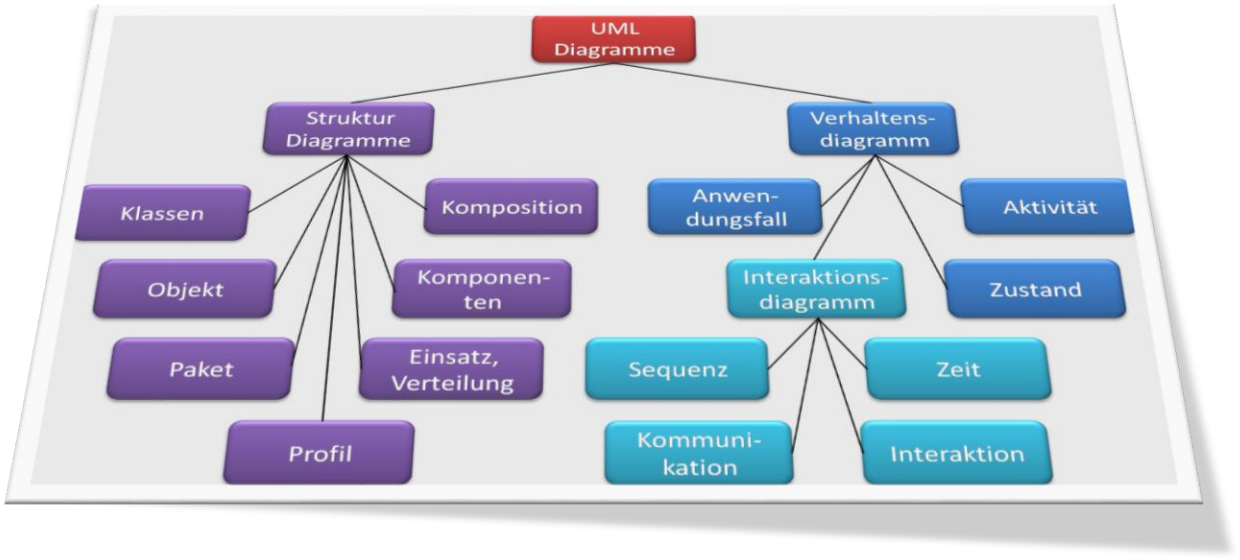
ITEN ENGINEERING
Software Development

UML

Einführung in die Konzepte der Unified Modeling Language (UML) und Kennenlernen der verschiedenen Struktur-, Verhaltens- und Interaktionsdiagramme.



Ausgabe 08.2017

Inhaltsverzeichnis

1. Einführung 4
1.1. Einleitung 4
1.2. Geschichte 4
1.3. Strukturierung 6
1.4. Profile 7
1.5. Meta Modellierung 7
2. Übersicht 9
2.1. Strukturdiagramme (engl. Structural diagrams) 10
2.2. Verhaltensdiagramme (engl. Behavioral diagrams) 11
2.3. Interaktionsdiagramme (engl. Interaction diagrams) 12
3. Allgemeine Elemente und Konzepte 13
3.1. Diagramme 13
3.2. Pakete (engl. Package) 14
3.3. Stereotypen (engl. Stereotype) 16
3.4. Notizen (engl. Notes) 18
3.5. Zusicherungen (engl. Constraint) 19
3.6. Eigenschaftswert (engl. Tagged Values) 20
4. Klassendiagramm 21
4.1. Klasse 21
4.2. Attribute 23
4.3. Operationen 24
4.4. Vererbung 25
4.5. Assoziation 26
4.6. Aggregation 29
4.7. Komposition 29
4.8. Abhängigkeit 30
4.9. Realisation 30
5. Objektdiagramm 31
6. Paketdiagramm 32
7. Komponentendiagramm 34
8. Einsatz- und Verteilungsdiagramm 36
9. Anwendungsfalldiagramm 37
10. Aktivitätsdiagramm 41
10.1. Aktivitätsdiagramm 41
10.2. Aktionsknoten / Aktionen 42
10.3. Aktivitäten 42
10.4. Kontrollflüsse 42
10.5. Kontrollknoten 43
10.6. Objektknoten und Objektflüsse 44
10.7. Partitionen (Verantwortlichkeitsbereiche) 45
10.8. Signale 46
11. Zustandsdiagramm 47
12. Sequenzdiagramm 51
13. Kommunikationsdiagramm 55
14. Zeitdiagramm 56
15. Übungen 57
15.1. Pflegeheim (Klassendiagramm) 57
15.2. Nationalliga (Klassen- und Objektdiagramm) 58
15.3. Store (Komponentendiagramm) 59
15.4. WebApp (Komponentendiagramm) 59
15.5. Reporting (Einsatz- / Verteildiagramm) 60
15.6. Coiffeur Salon (Anwendungsfall) 61
15.7. Patentverwaltung (Systemkontext) 61
15.8. Reisebüro (Aktivitätsdiagramm) 62

15.9. Bestellvorgang (Aktivitätsdiagramm) 62
15.10. Thread (Zustandsdiagramm) 63
15.11. Kreditvergabe (Zustandsdiagramm) 63
15.12. CD Sales Report (Sequenzdiagramm) 64
15.13. Mikrowelle (Sequenzdiagramm mit Nebenläufigkeit) 64
15.14. Zahlungsauftrag (Sequenzdiagramm mit alternativem Ablauf und Abbruch) 65
15.15. CD Sales Report (Kommunikationsdiagramm) 65
16. Literatur und Weblinks 66

1. Einführung

1.1. Einleitung

Die Abkürzung UML steht für Unified Modeling Language. UML ist keine Programmiersprache sondern eine grafische Sprache für die Modellierung von Software Systemen. Es ist handelt sich um eine Auswahl von Diagrammen mit denen ein Software System spezifiziert, visualisiert und dokumentiert werden kann.

Software Entwickler und Architekten verwenden UML um das Design der Software zu konstruieren und zu erläutern, genauso wie Architekten von Gebäuden, Zeichnungen und Modelle Ihrer Entwürfe erstellen. UML ist aber auch geeignet um Anforderungen für ein System zu erfassen oder Analysen durchzuführen.

Die von UML zur Verfügung gestellten Diagramme decken alle Phasen der Softwareentwicklung ab. Von der Erhebung der Anforderungen über Analyse, Design bis zur Codierung, dem Testen und Installieren der Software.

UML ist nicht zu verwechseln mit einem Software Entwicklungsprozess. UML ist einzig ein Set von Diagrammen welche innerhalb von Entwicklungsprozessen eingesetzt werden. UML kann in beliebigen Vorgehensmodellen wie zum Beispiel RUP (Rational Unified Process), XP (eXtreme Programming), V-Modell oder Hermes (dem Vorgehensmodell des Bundes) eingesetzt werden.

1.2. Geschichte

UML 1.x:

UML wurde in den 1990iger Jahren von den damals drei führenden Methoden Spezialisten Grady Booch, James Rumbaugh und Ivar Jacobson entwickelt. Die drei sind auch unter dem Spitznamen „die drei Amigos“ bekannt.

Jeder von Ihnen hatte bereits seine eigene Modellierungssprache entwickelt. Als Sie nun zusammen bei der „Firma Rational“ Software arbeiteten, hatten Sie das Ziel, die einzelnen, bereits vorhandenen, Methoden zu vereinen (engl. unify) und einen neuen Standard für die Software Modellierung zu erstellen.

Eine Vielzahl von unterschiedlichen Modellierungssprachen hatte dabei einen Einfluss auf die Konzeption von UML. Dies ist auch sehr gut aus der folgenden Grafik zur „Historie der objektorientierten Methoden und Notationen“ ersichtlich. Als Resultat dieser Bemühungen entstand die UML.

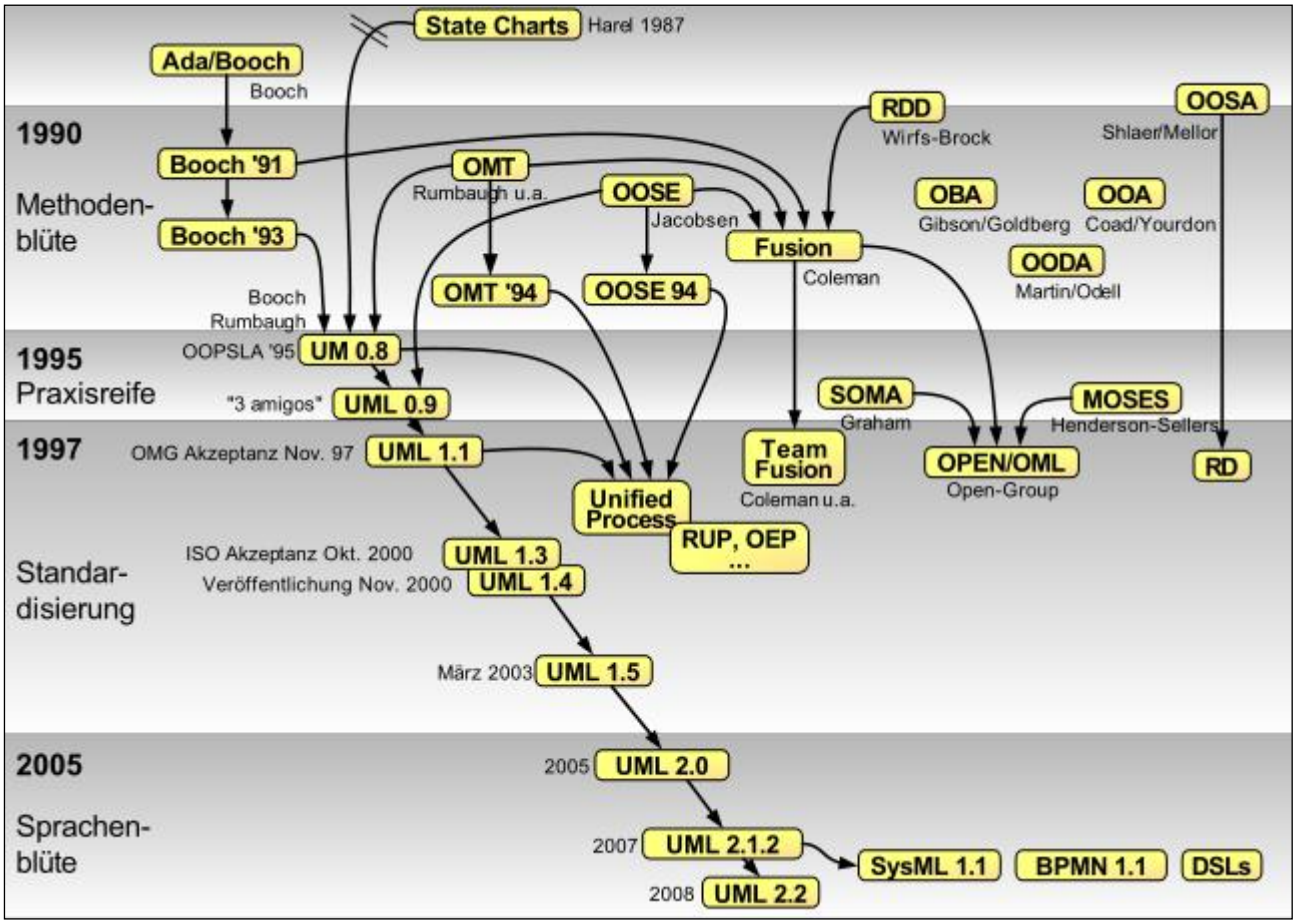
Die Standardisierung, Pflege und Weiterentwicklung der Sprache wurde später an die Object Management Group (OMG) übergeben, die die Sprache am 19. November 1997 als Standard akzeptierte.

Object Management Group:

Die Object Management Group (OMG) ist ein 1989 gegründetes Konsortium, das sich mit der Entwicklung von Standards für die Hersteller unabhängige systemübergreifende Objektorientierte Programmierung beschäftigt. Der OMG gehören u. a. Firmen wie Apple, IBM, Microsoft und Oracle an. Mittlerweile hat sie über 800 Mitglieder und entwickelt international anerkannte Standards.

Die bekanntesten Entwicklungen der OMG sind die Common Object Request Broker Architecture (CORBA), die das Erstellen von verteilten Anwendungen in heterogenen Umgebungen vereinfacht, sowie die Unified Modeling Language (UML), welche die Modellierung und Dokumentation von Objektorientierten Systemen in einer normierten Syntax erlaubt.

Historie der objektorientierten Methoden und Notationen:



Quelle: www.oose.de

Die Grafik zeigt gut die vorhandene Vielfalt während der Methodenblüte und die anschliessende Vereinigung zur Unified Modelling Language.

Im Weiteren entstanden in dieser Zeit auch der „Unified Process“ und daraus die beiden Vorgehensmodelle „Rational Unified Process“ der Firma Rational Software (gehört heute zu IBM) und „OOSE Engineering Process“ der Firma OOSE in Deutschland. Die beiden Vorgehensmodelle (neben vielen anderen) werden heutzutage in der Praxis eingesetzt.

UML 2.x:

UML ist mittlerweile die am meisten verwendete Modellierungssprache für die Erstellung von Software Systemen. Mit UML 2 werden über eine Dutzend verschiedene Diagrammtypen definiert, welche in Struktur-, Verhaltens- und Interaktionsdiagramme gegliedert sind.

Mit der Überarbeitung der UML von Version 1 auf 2 wurde unter anderem die Spezifikation neu strukturiert, mit dem Ziel, dass Sie einfacher zu implementieren, anzupassen und anzuwenden ist.

Die aktuellen Spezifikationen können von der entsprechenden Web Seite der Object Management Group (http://www.omg.org/uml) kostenlos bezogen werden.

1.3. Strukturierung

Die Spezifikation ist in folgende Teile gegliedert:

UML Infrastructure:

- Definiert die grundlegenden Sprachkonstruktionen, auf denen UML aufbaut.
- So werden zum Beispiel Konzepte wie Klasse, Assoziationen oder die Multiplizität eines Attributes spezifiziert.
- Dieser Teil ist für den Anwender der UML nicht direkt relevant, sondern richtet sich eher an die Hersteller von Modellierungswerkzeugen

Die UML Infrastructure soll

- Einen wieder verwendbaren Meta Sprachkern zur Verfügung stellen, mit dem die UML selbst definiert werden kann.
- Mechanismen zur Anpassung der Sprache zur Verfügung stellen.

UML Superstructure:

- Baut auf der UML Infrastructure auf und definiert die Anwender Konstrukte der UML.
- So werden zum Beispiel Konzepte wie Anwendungsfall, Aktivität oder Zustandsautomat spezifiziert.
- Dieser Teil definiert also diejenigen Elemente der Sprache, mit denen der Anwender direkt arbeitet.

Die UML Superstructure soll

- Eine bessere Unterstützung für die Komponenten basierte Entwicklung bieten.
- Die Konstrukte zur Spezifikation von Architekturen verbessern.
- Bessere Möglichkeiten zur Modellierung des Verhaltens bieten.

Object Constraint Language:

- Die Object Constraint Language (OCL) ist eine Sprache zur Spezifikation von Restriktionen.
- Die Syntax ist dabei an die Programmiersprache Smalltalk angelehnt.
- Wird zum Beispiel verwendet für die Formulierung von Bedingungen in Sequenzdiagrammen oder von Vor- und Nachbedingungen für Methoden.

UML Diagramm Interchange:

- Die UML Diagramm Interchange (UMLDI) Spezifikation ermöglicht den Austausch von UML Diagrammen (sogenannte UML Modelle) zwischen verschiedenen Software Tools.
- Dabei wird auch das Aussehen (Layout) der einzelnen Diagrammelemente berücksichtigt.
- Dieser Teile richtet sich an die Hersteller von UML Werkzeugen.

1.4. Profile

Profile bieten die Möglichkeit, den UML Standard für spezifische Bereiche oder Technologien anzupassen. Damit erhält man für eine spezifische Domäne einen genau zugeschnittenen Katalog.

Aktuelle sind folgende Profile definiert:

- Platform Independent Model (PIM) & Platform Specific Model (PSM) for Software Radio Components (also referred to as UML Profile for Software Radio)
- UML Profile for CORBA® and CORBA® Component Model (CCM) [This specification supersedes the separate profiles for CORBA® and the CORBA® Component Model.]
- UML Profile for Enterprise Application Integration (EAI)
- UML Profile for Enterprise Distributed Object Computing (EDOC)
- UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms
- UML Profile for Schedulability, Performance and Time
- UML Profile for System on a Chip (SoC)
- UML Profile for Systems Engineering (SysML)
- UML Testing Profile

Die aufgelisteten Profile sind auf der entsprechenden OMG Profil Katalog Seite unter http://www.omg.org/technology/documents/profile_catalog.htm verfügbar.

1.5. Meta Modellierung

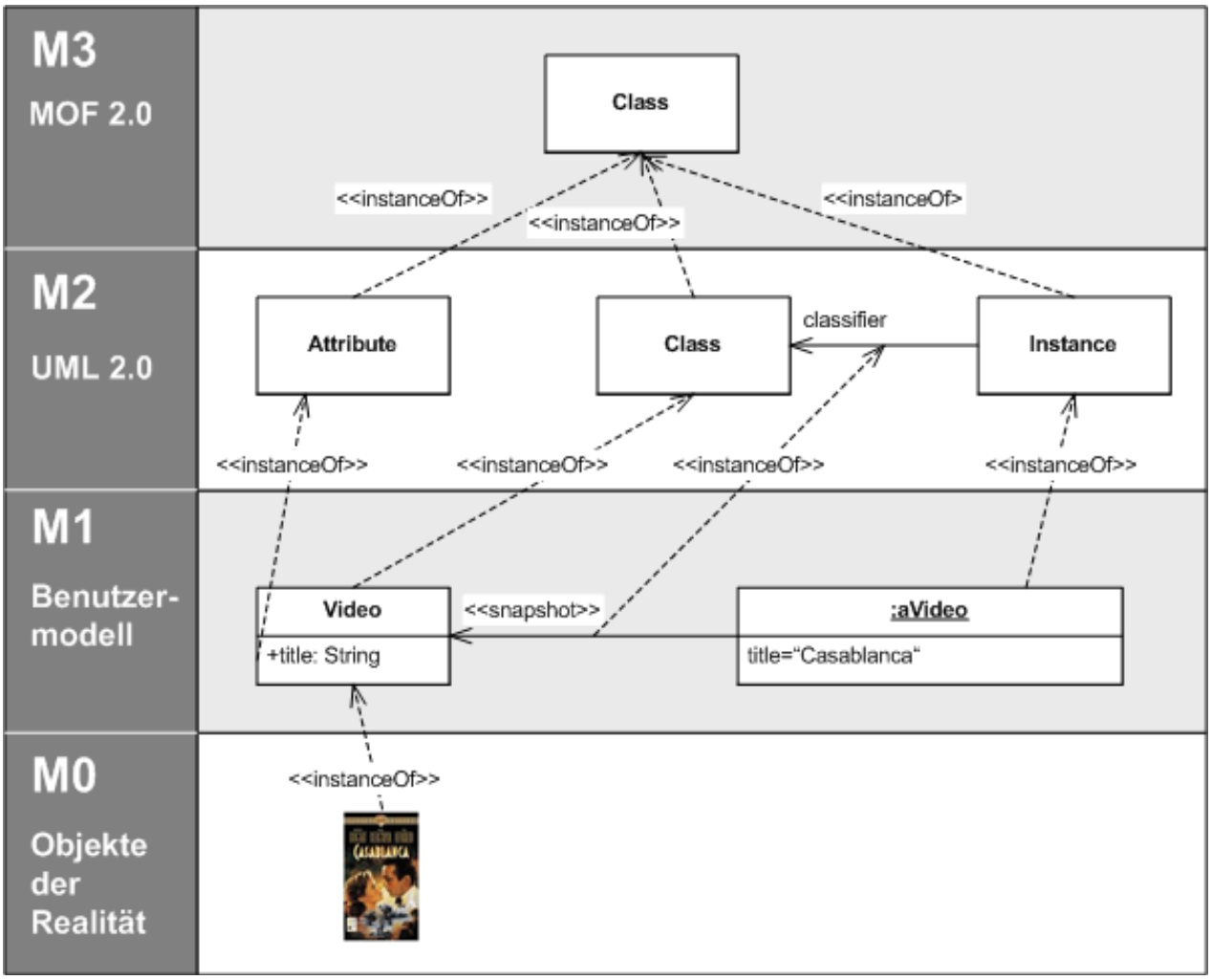
Ähnlich wie sich natürliche Sprachen in Lexika oder Grammatiken selbst beschreiben, wurde auch UML als ein Sprachwerkzeug konzipiert, das sich mit einigen Sprachbestandteilen selbst erklärt.

Hierzu wurde von der Object Management Group der Begriff Meta Object Facility (MOF) eingeführt. Dieser beschreibt eine spezielle Metadaten Architektur. Diese unterscheidet die folgenden vier Ebenen:

Ebene	Bezeichnung
M3	Meta-Metamodell Abstrakte Beschreibung der Modell-Notation
M2	Metamodell Abstrakte Beschreibung des Models
M1	Domain Modell Abstrakte Beschreibung der realen Welt.
M0	Instanzen, Objekte und Daten repräsentieren die reale Welt

Des Weiteren enthält die MOF-Spezifikation das XML Metadata Interchange Format (XMI) für den Austausch von Metadaten. Damit können solche Modelle auch zwischen verschiedenen Software Entwicklungswerkzeugen ausgetauscht werden.

Hierarchie der UML Meta-Modellierung:



Quelle: http://en.wikipedia.org

Die vier Ebenen bei der UML Meta Modellierung:

- M3: Die bereits erwähnte MOF stellt die oberste der vier Schichten dar. Es ist die Metasprache der Metasprachen von UML2 und beinhaltet deren grundlegende Elemente.
- M2: Die Metasprache von UML2 spezifiziert deren grundlegende Elemente mit ihren Eigenschaften.
- M1: Die in der Praxis verwendete UML befindet sich auf der Ebene 1. Dies sind die Diagramme die man als Anwender von UML erstellt.
- M0: Mit der untersteten Schicht werden die realen Objekte dargestellt.

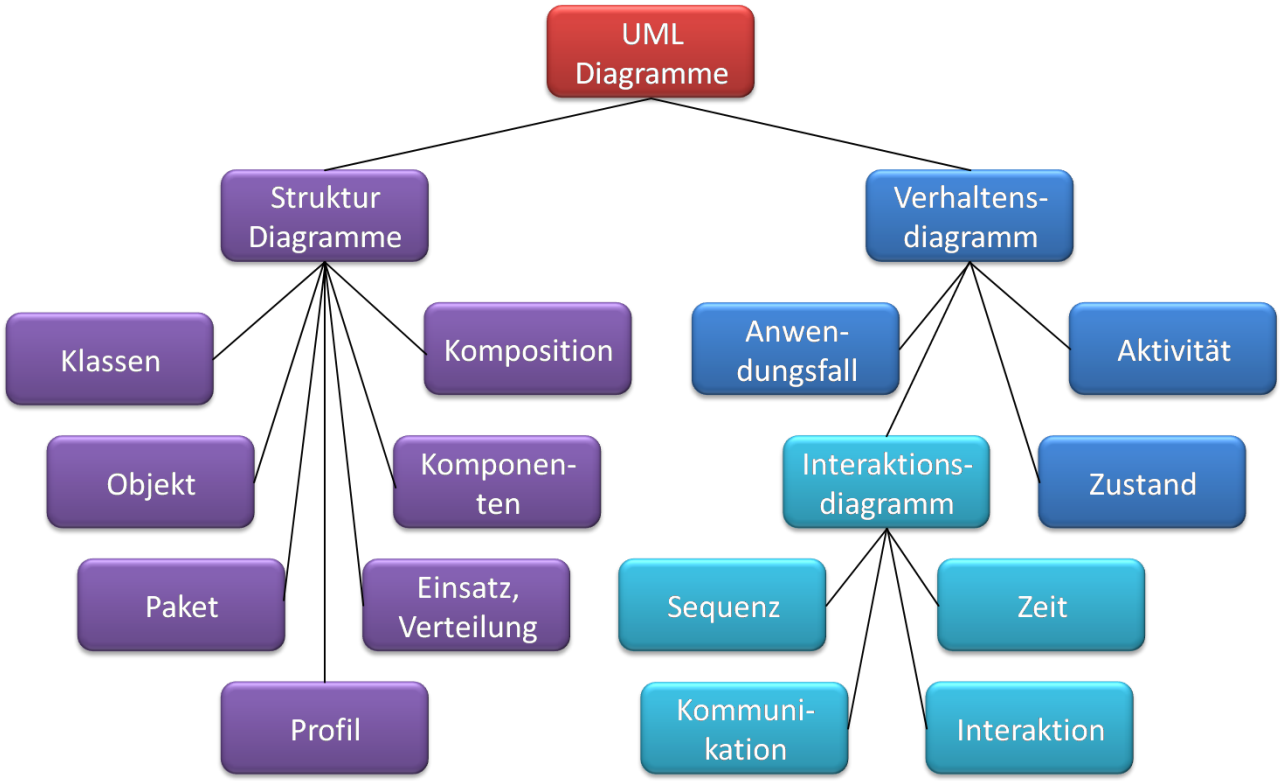
Wie in UML2.0, war auch UML1.x auf der dritten von vier Meta Modellierungsebenen eingeordnet. Zu UML 1.x besteht jedoch ein wesentlicher Unterschied: Die auf diesen vier Ebenen verwendeten Modellierungssprachen, besonders die Sprachen auf den Ebenen M2 und M3, teilen sich die gemeinsame Spracheinheit der Infrastrukturbibliothek (Infrastructure Library). Sie wird in der UML 2.0 Infrastructure definiert und bildet einen Kern der grundlegenden Modellierungselemente, der sowohl in der UML 2.0 Infrastructure als auch in der UML 2.0 Superstructure und in der MOF 2.0 eingesetzt wird.

2. Übersicht

Generell unterscheidet man in UML zwischen Struktur- und Verhaltensdiagrammen. Die erste Kategorie bietet eine statische Sicht auf das System während dem die zweite Gruppe dynamische Aspekte aufzeigt.

Innerhalb der Verhaltensdiagramme gibt es zudem die Untergruppe Interaktionsdiagramme. Diese zeigen Objekte und Ihren Beziehungen sowie die Nachrichten die untereinander ausgetauscht werden.

Die UML Diagramme im Überblick:



Nachfolgend werden die einzelnen Diagrammtypen und Ihr Verwendungszweck kurz aufgezeigt. Anschliessend werden die wichtigsten Diagrammtypen in Detail erläutert.

2.1. Strukturdiagramme (engl. Structural diagrams)

Diagramm	Beschreibung und Verwendungszweck
Klassen *engl. Class*	*Beschreibung:* Zeigt eine Auswahl von Klassen in einem System sowie die Beziehungen und Vererbungshierarchie unter den Klassen. Die Klassenelemente können auch die Attribute und Operationen der Klasse enthalten. *Verwendungszweck:* Das Klassendiagramm ist essentiell für das Aufzeigen der Struktur eines Systems oder Teile davon. Es zeigt auch was programmiert werden muss. Viele UML Entwicklungswerkzeuge (Case Tools) können basierend auf dem Klassendiagramm Programmcode erzeugen.
Objekt *engl. Object*	*Beschreibung:* Das Objektdiagramm zeigt spezifische Instanzen von Objekten mit deren gegenseitigen Verbindungen. Es zeigt Teile des Systems zu einem bestimmten Zeitpunkt (Snapshot). *Verwendungszweck:* Mit dem Objektdiagramm kann das Klassendiagramm verifiziert werden oder es können spezifische Punkte es Klassendiagramms weiter erläutert werden.
Paket *engl. Package*	*Beschreibung:* Ein Paket ist dabei eine Sammlung von logisch zusammengehörenden UML Elementen. Damit kann das Gesamtmodell in überschaubare Einheiten gegliedert werden. *Verwendungszweck:* Pakete werden für die verschiedensten Anwendungszwecke eingesetzt. So zum Beispiel für die Unterteilung der Software Komponenten bei mehrschichtigen Anwendungen oder die Aufteilung von fachlichen zusammengehörenden Einheiten (Module, Anforderungen, etc.)
Komponenten *engl. Component*	*Beschreibung:* Zeigt die wichtigsten Software Komponenten und deren Beziehungen untereinander. Komponentendiagramme können auch nicht objekt orientierte Software Komponenten (wie z. B. Legacy Systeme) beinhalten. *Verwendungszweck:* Aufzeigen der wichtigsten Software Komponenten und deren Integration im System. Bietet eine „high level“ Perspektive der Softwarestruktur des Systems.
Einsatz- und Verteilung *engl. Deployment*	*Beschreibung:* Zeigt die Hardware Komponenten (Nodes) und installierten Software Module eines Systems. *Verwendungszweck:* Das Einsatz- und Verteilungsdiagramm ist nützlich um die Konfiguration eines verteilten Systems aufzugeigen. Die Software Artefakte können innerhalb der Hardware Komponenten eingezeichnet werden um zu zeigen, wo diese installiert werden.

Kompositions-struktur *engl. Composite Structure*	*Beschreibung:* Das Kompositionsstrukturdiagramm zeigt die innere Zusammensetzung einer Komponente oder Klasse. *Verwendungszweck:* Aufzeigen von Einzelteilen und deren Verbindungen zu einem Ganzen.
Profil *engl. Profile*	*Beschreibung:* *Das Profil Diagramm zeigt die Definition von eigenen Stereotypen dar.* *Verwendungszweck:* *Das Profildiagramm ist nützlich, wenn eine Firma oder ein Projekt UML an seine eigenen Bedürfnisse anpassen will. Die Profildiagramme werden dazu eingesetzt, die Herleitung der eigenen Definitionen aufzuzeigen.*

2.2. Verhaltensdiagramme (engl. Behavioral diagrams)

Diagramm	Beschreibung und Verwendungszweck
Anwendungsfall *engl. Use Case*	*Beschreibung:* Zeigt die Akteure, Anwendungsfälle und Ihre Beziehungen. Ermöglicht einen raschen Überblick über die Verwendung und Prozesse des Systems. *Verwendungszweck:* Das Anwendungsfall Diagramm ist sehr wichtig für die Erhebung der Anforderungen und Analyse der Arbeitsabläufe (Workflow). Während der ganzen Entwicklungszeit sollte es möglich sein, die ausgeführten Arbeitsschritte einem Anwendungsfall zuzuordnen.
Aktivität *engl. Activity*	*Beschreibung:* Zeigt Aktivitäten, Objektzustände, Zustandsübergänge und Ereignisse. Es zeigt die Abfolge von Aktivitäten in einem Prozess oder Algorithmus. *Verwendungszweck:* Das Aktivitätsdiagramm eignet sich sehr gut für das Aufzeigen von Geschäftsprozessen während der Erhebung der Anforderungen und Analyse der der Arbeitsabläufe.
Zustand *engl. Statechart, State Machine*	*Beschreibung:* Zeigt die verschieden Zustände eines Objektes sowie die Ereignisse und Bedingungen für einen Zustandswechsel. *Verwendungszweck:* Das Zustandsdiagramm ist nützlich für das Aufzeigen und Nachvollziehen von signifikanten Statuswechseln eines Objekts.

2.3. Interaktionsdiagramme (engl. Interaction diagrams)

Innerhalb der Verhaltensdiagramme wird die folgende Untergruppe von Diagrammen als Interaktionsdiagramme bezeichnet:

Diagramm	Beschreibung und Verwendungszweck
Sequenz *engl. Sequence*	*Beschreibung:* Zeigt Objekte und Ihre Beziehungen inklusive ihres zeitlich geordneten Nachrichtenaustausches aus einer objekt orientierten Perspektive. Sequenzdiagramme visualisieren einen Ablauf (oder Teil davon) welcher im System stattfindet. *Verwendungszweck:* Die Sequenzdiagramme zeigen die Verantwortlichkeiten und Zusammenarbeit der einzelnen Klassen und kommen vor allem in der Analyse und Design Phase zum Einsatz.
Kommunikation *engl. Collaboration, Communication*	*Beschreibung:* Zeigt Objekte und Ihre Beziehungen inklusive ihres räumlichen geordneten Nachrichtenaustausches. Sowohl das Sequenz- als auch das Kollaborationsdiagramm zeigen Prozesse aus Sicht einer objekt orientierten Perspektive. Der Unterschied ist, dass das Kollaborationsdiagramm den Fokus mehr auf die Objekte anstatt die Abläufe legt. *Verwendungszweck:* *Der Einsatz erfolgt vorwiegend in der Design Phase und zeigt detailliert die Objekte und deren Rollen. Da Sequenzdiagramme einfacher zu lesen sind und die gleichen Informationen enthalten, werden diese den Kommunikationsdiagrammen oft vorgezogen.*
Zeit *engl. Timing*	*Beschreibung:* Ein Zeitdiagramm beschreibt die zeitlichen Bedingungen von Zustandswechseln mehrerer beteiligter Objekte. Es handelt sich dabei um die gleichen Elemente wie in Sequenz- und Kommunikationsdiagrammen. *Verwendungszweck:* Aufzeigen einer bestimmten Sicht auf dynamische Aspekte des Systems.
Interaktions-übersicht *engl. Interaction Overview*	*Beschreibung:* Die Interaktionsübersicht ist ein Aktivitätsdiagramm, in dem Teilabläufe durch referenzierte oder eingebettete Sequenzdiagramme repräsentiert werden. *Verwendungszweck:* Bei grossen komplexen Systemen können die einzelnen Sequenzen in einer übergeordneten Übersicht zusammengefasst werden. Die Sequenzen werden dabei in ihrer zeitlichen Abfolge gegliedert.

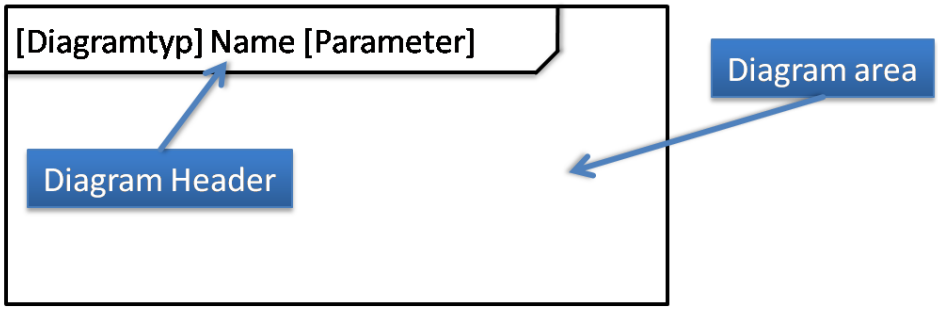
3. Allgemeine Elemente und Konzepte

UML Diagramme repräsentieren Konzepte, dargestellt als Symbole (Nodes) und Beziehungen (Links). Je nach Diagramm Typ werden die Symbole und Beziehungen spezialisiert. In einem Klassendiagramm zum Beispiel repräsentieren die Symbole die Klassen und die Beziehungen die Verbindungen und Vererbungshierarchien der Klassen untereinander.

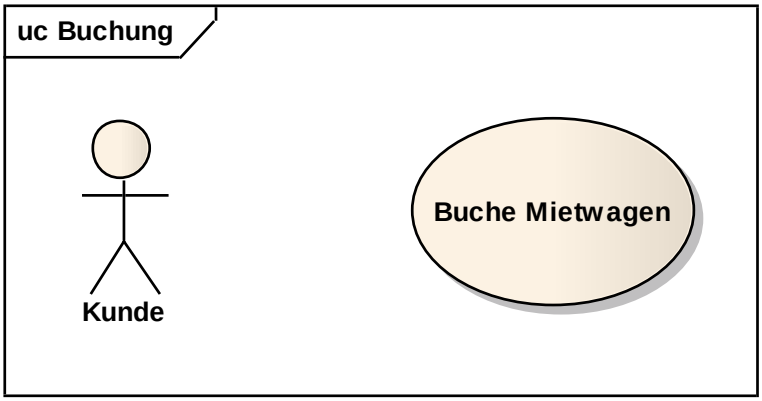
Zusätzlich gibt es weitere allgemeine Elemente, mit welchen die Diagramme ergänzt werden. Die gebräuchlichsten werden nachfolgend kurz aufgezeigt.

3.1. Diagramme

Die Basis Notation für ein UML Diagramm besteht aus einem Diagramm Kopf (engl. Header) und dem eigentlichen Bereich (engl. Area) mit den Modellelementen.



Beispiel:



Das Beispiel zeigt einen Anwendungsfall (engl. Use Case). Der Diagrammtyp wird dabei in abgekürzter Schreibweise (uc für Use Case) dargestellt. Parameter werden keine angegeben.

Notation:

Der Header besteht aus drei Teilen. Der Name wird immer angegeben, Diagrammtyp und Parameter sind optional. Der Diagramm Typ wird z. T. auch in abgekürzter Form angegeben. Also zum Beispiel uc für Use Case oder sd für Sequence Diagram.

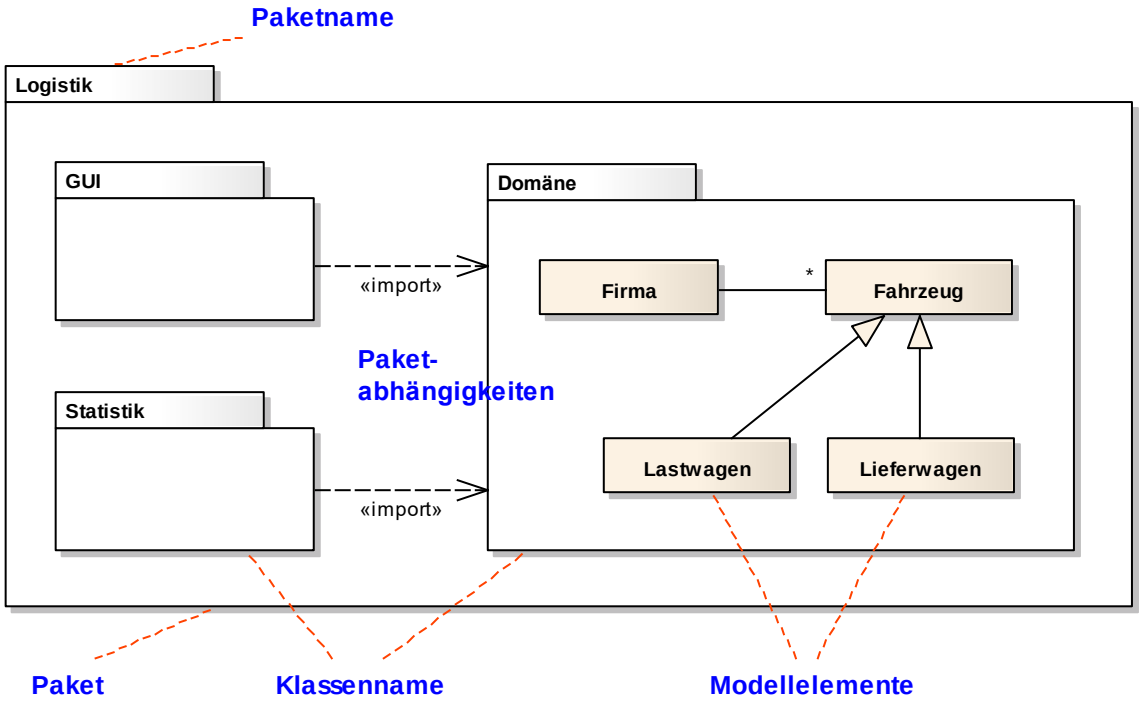
Mit den Parametern können die Eingangsparameter und der Rückgabewert des Diagramms angegeben werden. Der Header für eine Abfrage des Kontostandes könnte für ein Sequenzdiagramm zum Beispiel folgendermassen aussehen:

sd Balance Lookup (int accountNumber) : int

3.2. Pakete (engl. Package)

Mit Paketen können die Elemente des Models gruppiert werden. Pakete in UML sind ein generischer Mechanismus um eine logische Struktur erstellen zu können. Sie haben keine direkte Beziehung zu ähnlichen Konstruktionen der Programmiersprachen wie zum Beispiel den Packages von Java. Es ist aber natürlich möglich, mit UML Paketen Java Packages zu modellieren.

Beispiel Logistik:



Das Beispiel zeigt die Aufteilung einer Logistikapplikation in drei Bereiche. Die Geschäftsdomäne, das grafische User Interface und ein Statistik Modul.

Die Verbindungen zeigen die Abhängigkeiten der Pakete untereinander. Das GUI und Statistik Modul sind beide Abhängig von Domänen Modell. Zwischen GUI und Statistik besteht keine Abhängigkeit.

Notation:

Pakete werden als Aktenregister dargestellt, welches den Namen des Paketes enthält. Oberhalb des Paketnamens können Stereotypen notiert werden. Innerhalb des Paket können weitere Pakete und / oder Elemente dargestellt werden.

Wenn das Paket keine weiteren Elemente im Diagramm anzeigt, ist es auch möglich den Paketnamen direkt im Paket anstatt im Register anzugeben.

Beispiel Java:

Möchte man nun das gezeigte Modell mit Java realisieren und die aufgezeigte Paket Struktur für die Java Packages übernehmen, so würde das folgendermassen aussehen:

Klasse Fahrzeug:

```
package logistik.domaene;

public class Fahrzeug {

        // Deklarationen
}
```

Klasse FahrzeugAuswahl:

```
package logistik.gui;

import logistik.domaene.*;                    // Abhängigkeit zum Paket
Domäne

public class FahrzeugAuswahl {

        // Deklarationen
}
```

Klasse FahrzeugStatistikReport:

```
package logistik.statistik;

import logistik.domaene.*;                    // Abhängigkeit zum Paket
Domäne

public class FahrzeugStatistikReport {

        // Deklarationen
}
```

Das Package Domäne hat keine weiteren Imports, d. h. es ist nicht abhängig von Modulen in anderen Packages.

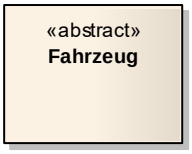
Die Packages GUI und Statistik importieren beide das Package Domäne, da Sie auf die Domänenobjekte zugreifen. Untereinader haben Sie aber keine Abhängigkeiten, d. h. das grafische User Interface und die Statistikapplikation sind unabhängig voneinander.

3.3. Stereotypen (engl. Stereotype)

Bei der Erstellung von UML wurde berücksichtigt, dass das UML Modell nicht alle aktuellen und zukünftigen Anforderungen der verschiedenen Programmiersprachen abdecken kann. Daher wurden verschiedene Mechanismen im UML eingebaut, um die Modelle mit einer eigenen Semantik ergänzen zu können.

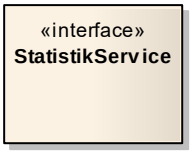
Eines dieser Hilfsmittel sind die Stereotypen. Mit diesen können den Elementen und Beziehungen eines Modells bestimmte Eigenschaften zugeordnet werden.

Beispiele:



Abstract:

Mit dem Stereotyp «abstract» wird angegeben, dass es sich bei der Klasse Fahrzeug um eine Abstrakte Klasse handelt.



Interface:

Mit dem Stereotyp «interface» wird angegeben, dass es sich bei der Klasse StatistikService um ein Interface handelt.

Mit den beiden Beispielen wird gut ersichtlich, dass man dank den Stereotypen einem bestehenden UML Element (in unserem Beispiel der Klasse) eine spezifische Bedeutung zuordnen kann, ohne dass das Meta Modell der UML Spezifikation mit einem neuen Element ergänzt werden muss.

Notation:

Stereotypen werden oberhalb oder vor den Elementnamen angegeben und mit den Zeichen «…» (guillements) umschlossen. Die beiden auch „angled quotes“ genannten Zeichen, sollten nicht mit den kleiner << und grösser als >> Zeichen verwechselt werden.

Ein UML Element kann auch gleichzeitig mit mehreren Stereotypen klassifiziert werden. Neben der Angabe bei Elementen können Stereotypen auch bei Attributen, Methoden oder Beziehungen verwendet werden.

Um zusätzliche Informationen anzugeben, können Stereotypen zudem mit eigenen Attributen ergänzt werden.

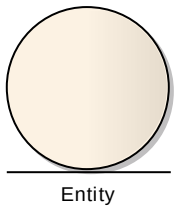
Beachte:

Es ist zu beachten, dass es sich nicht bei allen so angegebenen Bezeichnungen um einen Stereotyp handeln muss. Die in UML vordefinierten Schlüsselwörter werden ebenfalls mit guillements umschlossen.

Grafische Darstellung:

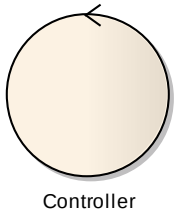
Es ist auch möglich, die grafische Darstellung des Elementes je nach Stereotyp zu variieren. Damit wird die Semantik der Elemente mit der grafischen Darstellung zusätzlich „unterstrichen“ und aussagekräftig dargestellt.

Gute Beispiele sind die drei Stereotypen «entity», «boundary» und «control» für die alternativ die folgenden Symbole verwendet werden können:



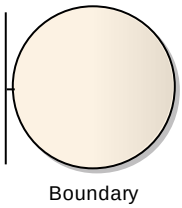
Entity:

Entitätsklassen repräsentieren fachlichen Sachverhalt oder Realweltgegenstand (Vertrag, Kunde, Adresse).



Control:

Steuerungsklassen dienen Ablauf-, Steuerungs- und Berechnungsvorgängen, meistens stark klassenübergreifend.



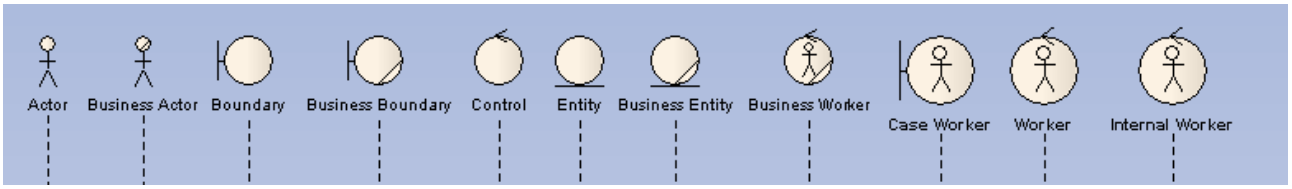
Boundary:

Schnittstellenobjekte bilden eine Zusammenstellung von Eigenschaften anderer Objekte, zum Beispiel zur Entkopplung im Sinne von Fassaden.

Herstellerspezifische Ergänzungen:

Zusätzlich erlaubt es die Spezifikation dass Hersteller von UML Werkzeugen weitere grafische Elemente definieren oder zum Beispiel mit verschiedenen farblichen Darstellungen die Bedeutung von Stereotypen hervorheben.

Beispiel Enterprise Architekt:



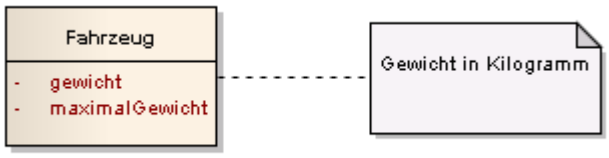
Die Grafik zeigt die vordefinierten Symbole des „Enterprise Architekt“ für die Darstellung von Geschäftsabläufen mit Hilfe von Sequenzdiagrammen.

3.4. Notizen (engl. Notes)

UML Diagramme können mit zusätzlichen Notizen ergänzt werden. Diese können sich auf einzelnen Elemente (Klassen, Attribute, Operationen, etc.), Beziehungen oder auch das ganze Modell beziehen.

In einer Notiz können z.B. Informationen über den Entwicklungsstand, verantwortliche Entwickler eines Modellelementes, Versionsnummer einer Klasse u.ä. stehen. Möglich ist auch Daten des Projektmanagements in einer Notiz zu speichern, beispielsweise der bisherige Aufwand und der geschätzte Restaufwand für ein Projekt.

Beispiel Fahrzeug:



Die Klasse Fahrzeug wird mit der Beschreibung der verwendeten Masseinheit für das Gewicht ergänzt.

Notation:

Notizen werden als Rechtecke mit einem Eselsohr dargestellt. Sie enthalten die zu notierenden Informationen und besitzen wahlweise eine Linie bzw. Abhängigkeitsbeziehung, die vom Rechteck zum zugehörigen Modellelement führt.

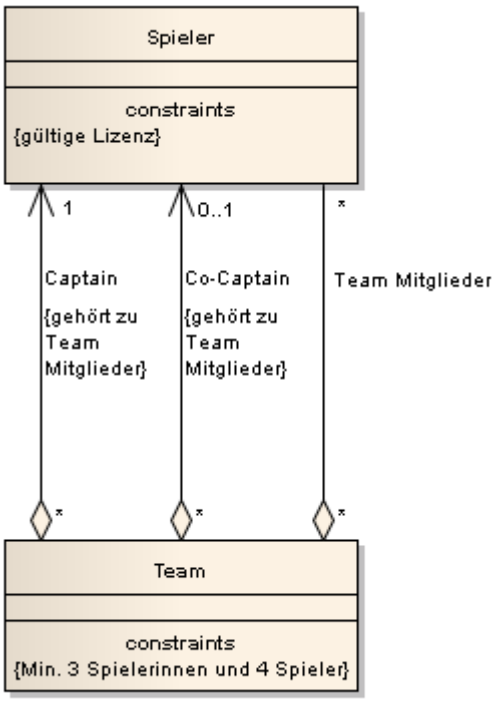
Die gestrichelte Linie im Beispiel zeigt zum Element auf das sich die Notiz bezieht. Wenn es keine gestrichelte Linie hat, so bezieht sich die Notiz auf das gesamte Model.

3.5. Zusicherungen (engl. Constraint)

Mit Einschränkungen / Zusicherungen können gewisse Bedingungen für ein Element oder eine Beziehung formuliert werden.

Eine Zusicherung beschreibt eine Bedingung oder Integritätsregel. So kann sie z.B. die zulässige Wertemenge eines Attributes einschränken, strukturelle Eigenschaften zusichern, zeitliche Bedingungen stellen und andere, ähnliche Bedingungen setzten. Sie fordert oder verbietet spezielle Eigenschaften. Zusicherungen können an beliebige Modellelemente angefügt werden, z.B. an Attribute, Operationen, Klassen, Assoziationen, usw. Sie repräsentieren zusätzliche semantische Informationen zu einem Modellelement.

Beispiel Spieler – Team:



Die Einschränkung bei den Spielern definiert, dass diese eine gültige Lizenz benötigen.

Die Einschränkung beim Team definiert, dass die Team Zusammensetzung aus mindestens drei weiblichen und 4 männlichen Spielern bestehen muss.

Bei der Beziehung des Captain und Co – Captain wird definiert, dass diese zu den Team Mitgliedern (für das aktuelle Team) gehören müssen. Es kann also nicht ein Spieler Captain eines Teams sein, bei dem er nicht mitspielt.

Notation:

Zusicherungen stehen zwischen geschweiften Klammern.

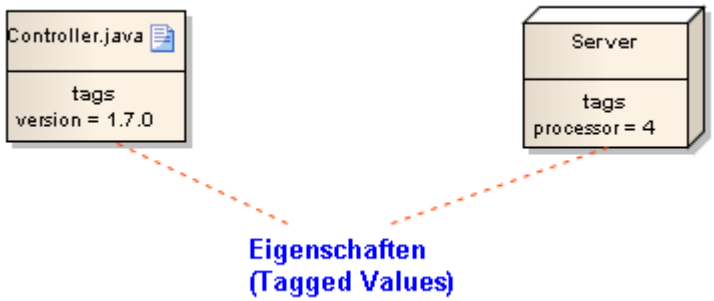
Zusicherungen, die eine direkte Abhängigkeit zweier Elemente definieren, werden durch eine gestrichelte Linie zwischen den entsprechenden Elementen notiert. Ist dabei ein Element vom anderen abhängig, so wird anstatt der Linie ein Pfeil in Richtung des abhängigen Elements notiert.

Falls eine Zusicherung mehrere Modellelemente betrifft, oder sie nicht direkt bzw. sinnvoll zugeordnet werden kann ist es möglich, die Zusicherung innerhalb einer Notiz zu beschreiben. Von dieser führt dann eine gestrichelte Linie zu den beteiligten Modellelementen.

3.6. Eigenschaftswert (engl. Tagged Values)

Eigenschaftswerte / Merkmale fügen vorhandenen, beliebigen Modellelementen bestimmte weitere charakteristische Eigenschaften hinzu und erweitern so deren Semantik.

Beispiele:



Die beiden Beispiele zeigen die Definition von Eigenschaftswerten für ein Java Artefakt und einen Server.

Die Darstellung erfolgt i.d.R. innerhalb von geschweiften Klammern anstelle der hier verwendeten Notation mit der Überschrift tags.

Notation:

Merkmale bestehen aus einem sogenannten Schlüsselwort-Wert-Paar. Schlüsselwort und zugehöriger Wert stehen in geschweiften Klammern und werden etikettenartig an jedes Modellelement angefügt.

Ist der Wert ein Boolean der true ist, kann er auch weggelassen werden, d.h. {transient=true} ist identisch mit {transient}.

Code Generierung:

Eigenschaftswerte sind strukturierter als eine reine Notiz und werden daher häufig eingesetzt, wenn aus Modellelementen Source Code generiert werden soll.

Dazu gibt es sogar eigens für die Code Generierung geschaffene Eigenschaften. Abstrakte Klassen und Operationen müssen auch im Code als abstrakt deklariert werden. Private Operationen und Attribute ebenso. Beispiele für solche Eigenschaften sind u.a.:

- *abstract:* für abstrakte Klassen und Operationen
- *readonly*: für Attribute die nur gelesen werden dürfen
- *private*: weist darauf hin, dass auf das Element von aussen nicht zugegriffen werden kann
- *deprecated*: das Element existiert nur noch zur Kompatibilität mit älteren Versionen

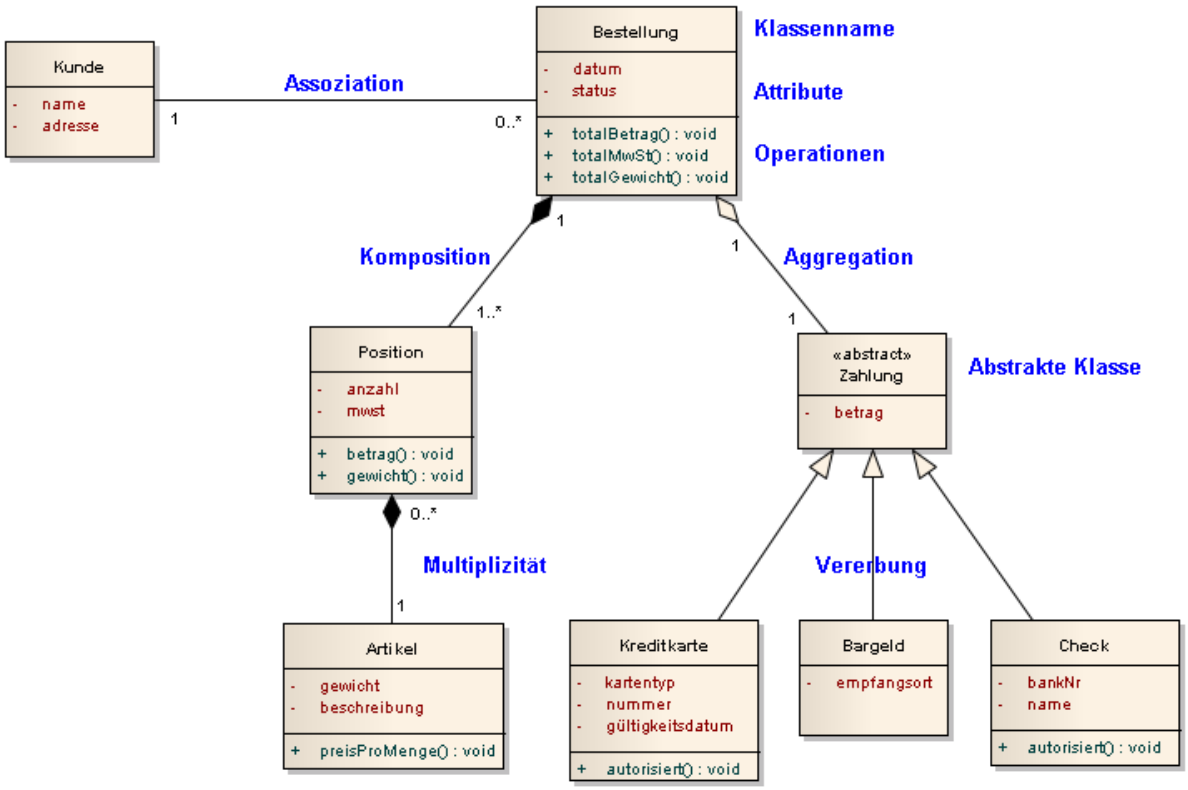
Eigenschaften sind somit eine mächtiges und bequemes Mittel für die Definition von semantischen Details!

4. Klassendiagramm

4.1. Klasse

Eine Klasse beschreibt dabei den Bauplan für Objekte mit gleicher Struktur (Attribute) und dem gleichen Verhalten (Operationen). Ein Klassendiagramm gibt einen Überblick eines Systems indem die wesentlichen Klassen und deren Beziehungen untereinander dargestellt werden. Klassendiagramme sind statisch, d.h. Sie zeigen welche Objekte miteinander interagieren aber nicht was passiert, wenn dies tatsächlich stattfindet.

Beispiel Bestellung:



Das Diagramm zeigt einen Bestellvorgang eines Kunden (z.B. von einem Versandkatalog). Die zentrale Klasse ist die Bestellung. Sie stellt die Verbindung mit den Kunden, den Details der Bestellung und dem Zahlvorgang her.

Die Bestelldetails bestehen aus einzelnen Positionen mit Artikeln und Angaben zum Betrag, Gewicht und Preis.

Eine Zahlung kann auf drei verschiedene Arten ausgeführt werden. Per Kreditkarte, Barzahlung oder mit einem Check.

Notation:

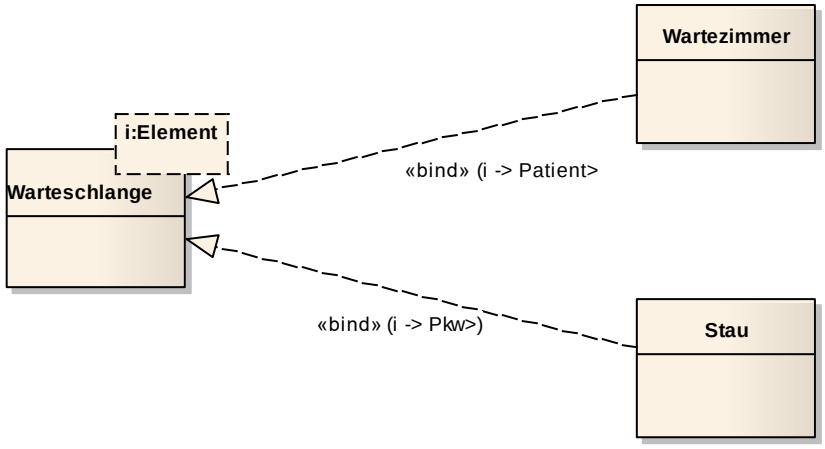
Eine **Klasse** wird durch ein, in drei Bereiche unterteiltes, Rechteck dargestellt. Zuerst folgt der Klassenname gefolgt von den Attributen und Operationen der Klasse. Die Angabe der Attribute und Operationen ist optional.

Bei abstrakten Klassen wird der Klassenname in *kursiver* Schrift dargestellt. Alternativ kann diese auch mit einem Stereotypen gekennzeichnet werden.

Parametrisierbare Klasse

Bei einer parametrisierbaren Klasse handelt es sich um eine Schablone zu Erzeugung von Klassen. In statisch typisierten Programmiersprachen, wie C++ oder Eiffel sind parametrisierbare Klassen ein wichtiges Hilfsmittel zur Erzeugung von wiederverwendbaren Code. Ab Java Version 5 und .Net Version 2.0 ist dieser Mechanismus (Generics) auch vorhanden.

Beispiel:



Notation:

Parametrisierbare Klassen werden wie Klassen dargestellt, sie erhalten aber zusätzlich in der rechten oberen Ecke in einem gestrichelten Rechteck die notwendigen Parameter.

Eine Klasse die durch eine parametrisierbare Klasse erzeugt wird muss mit einer Verfeinerungs-beziehung, die den Stereotyp **«**bind**»** erhält gekennzeichnet werden.

Aktive Klasse:

Aktive Klassen sind solche bei denen die jeweiligen Instanzen in einem eigenen Thread laufen. Solche Klassen werden vorwiegend für eingebettete Systeme und Echtzeitanwendungen eingesetzt.

Beispiel:

Notation:

Eine Aktive Klasse wird wie eine gewöhnliche Klasse notiert, jedoch befindet sich links und rechts an der Innenseite des Rechtecks eine weitere Linie. Alternativ kann die Angabe {active} verwendet werden.

4.2. Attribute

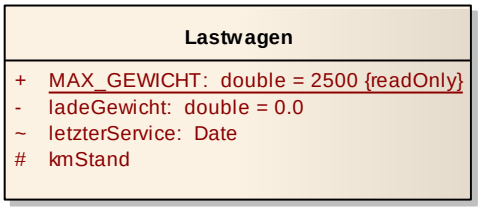
Attribute sind Informationen bzw. Daten die ein Element einer Klasse näher beschreiben. Sie werden mindestens durch ihren Namen beschrieben, können aber zusätzlich einen Initialwert und Zusicherungen besitzen.

- Durch Zusicherungen kann der Wertebereich bzw. die Wertemenge eines Attributes eingeschränkt werden.
- Mit Hilfe von Merkmalen können weitere besondere Eigenschaften von Attributen beschrieben werden, z.B. dass ein Attribut nur gelesen werden darf.

Neben normalen Attributen existieren noch sogenannte abgeleitete Attribute. Diese werden durch eine Berechnungsvorschrift automatisch berechnet. Sie sind innerhalb eines Objektes nicht durch einen physischen Wert repräsentiert und sie benötigen keinen Initialwert.

Eine weitere Ausprägung von Attributen sind Klassenattribute. Sie gehören nicht zu einem einzelnen Objekt, sondern zu einer Klasse. Alle Objekte dieser Klasse können ein solches Klassenattribut benutzen.

Beispiel:



Notation:

Innerhalb des Klassenrechteckes im Klassendiagramm werden die Attribute vom Klassennamen durch eine horizontale Linie getrennt und stehen somit in der zweiten Kategorie.

Eine Attributdefinition besteht aus fünf Elementen wobei nur der Attributname zwingend ist. Dieser beginnt mit einem Kleinbuchstaben. Anschliessend folgt der Typ sowie der Initialwert, die Merkmale und zum Schluss die Zusicherungen.

Name: Typ = Initialwert {Merkmal} {Zusicherung}

Abgeleitete Attribute werden mit einem vorangestellten Schrägstrich markiert. Klassenattribute werden unterstrichen und die Sichtbarkeitsangaben erfolgen durch die Angabe der folgenden Symbole:

Symbol	Sichtbarkeit	Zugriffsmodus
-	Private	Der Zugriff ist nur durch die eigene Klasse möglich.
~	Package private	Zugriff durch alle Klassen im gleichen Paket möglich.
#	Protected	Zugriff durch eigene und abgeleitete Klassen möglich.
+	Public	Zugriff durch alle Klassen möglich.

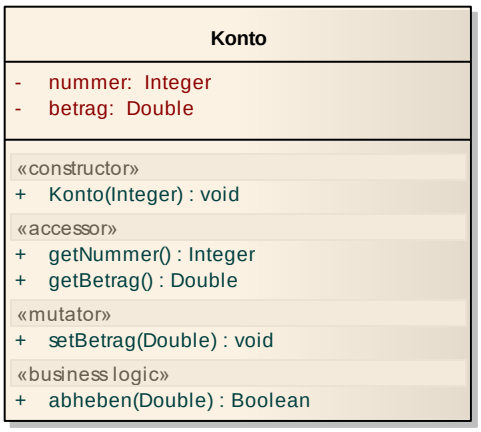
4.3. Operationen

Operationen sind Dienstleistungen, die von einem Objekt bereitgestellt werden und durch andere Objekte durch sogenannte Nachrichten aufgerufen werden können. Eine Nachricht besteht aus dem Selektor (ein Name) und einer Liste von Parametern. Sie wird an genau einen Empfänger gesendet.

Die Begriffe Operation und Nachricht werden oft synonym verwendet, was allerdings nicht richtig ist. Objekte kommunizieren untereinander mit Hilfe von Nachrichten. Ein Objekt kann aber nur eine solche Nachricht verstehen zu der es eine entsprechende Operation gibt.

Eine Operation setzt sich aus dem Namen der Operation, den Parametern (falls vorhanden) und einem eventuellen Rückgabewert zusammen. Eine Operation ist innerhalb einer Klassendefinition eindeutig identifizierbar und kann zudem mit Merkmalen und Zusicherungen versehen werden.

Beispiel:



Im Beispiel werden die Operationen zusätzlich mit Stereotypen klassifiziert. Das ist aber optional und soll im vorliegenden Fall einzig Auskunft über die Art der Methode liefern.

Notation:

Innerhalb des Klassenrechteckes werden Operationen im unteren Teil (nach den Attributen) aufgeführt. Die Signatur einer Operation sieht wie folgt aus:

Name (Argument:Typ=Standardwert,...) : Rückgabetyp {Merkmal} {Zusicherung}

Der Name einer Operation beginnt mit einem Kleinbuchstaben. Der Name des Argumentes beginnt ebenfalls mit einem Kleinbuchstaben. Das Argument wird durch Nennung seines Typs näher beschrieben, außerdem kann ein Initialwert angegeben werden. Argumentname und Argumenttyp werden durch einen Doppelpunkt getrennt. Merkmale und Zusicherungen stehen in geschweiften Klammern.

Abstrakte Operationen werden kursiv geschrieben oder durch das Merkmal abstrakt gekennzeichnet. Klassenoperationen werden durch Unterstreichung gekennzeichnet. Die äußere Sichtbarkeit von Operationen wird mit einem entsprechenden Symbol (analog den Attributen) angegeben.

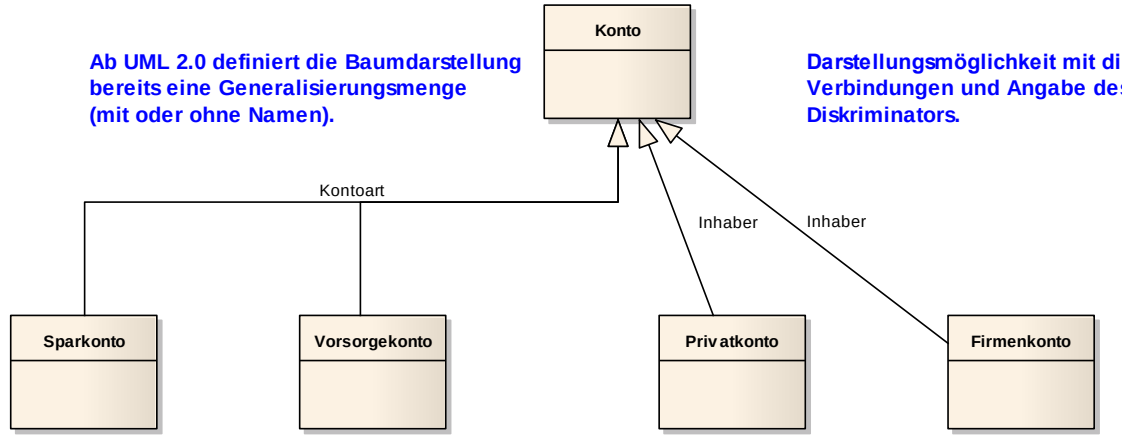
4.4. Vererbung

Mit Hilfe der Vererbung können Klassen hierarchisch strukturiert werden. Dabei werden Eigenschaften einer Oberklasse an die zugehörige Unterklasse weitergegeben. Eine Unterklasse verfügt folglich über ihre speziellen Eigenschaften und über die Eigenschaften ihrer Oberklasse(n).

Die Unterscheidung in Ober- und Unterklasse erfolgt anhand eines Unterscheidungsmerkmals, dem sogenannten Diskriminator. Dieser definiert den für die Strukturierung maßgeblichen Aspekt.

Bei der Darstellung werden zwei Varianten unterschieden. Die direkte Verbindung jeder einzelnen Unterklasse oder die baumartige Verbindung.

Beispiel:



Notation:

Die Vererbung wird mit einem nicht ausgefüllten Pfeil, der von der Unterklasse zur Oberklasse zeigt dargestellt. Die Pfeile von den Unterklassen können zu einer gemeinsamen Linie zusammengefasst werden oder direkt zur Oberklasse gezogen werden.

Zusammengefasste Pfeile stellen Gemeinsamkeiten der Unterklasse, nämlich dass sie Generalisierungen einer Oberklasse mit gemeinsamen Diskriminator sind, stärker dar.

Werden Generalisierungen mit gemeinsamen Diskriminator mit direktem Pfeil dargestellt, so sind die Pfeile entweder

- mit einer gestrichelten Linie, die den Namen des Diskriminator enthält zu verbinden
- oder alternativ jeder einzelne Pfeil mit dem Namen des Diskriminator zu versehen.

Wird die Angabe des Diskriminator weggelassen, so ist nicht mehr ersichtlich, ob es sich bei den Unterklassen um eigenständige Spezialisierungen handelt oder ob sie durch einen gemeinsamen Diskriminator entstanden sind.

Mehrfachvererbung, Delegation und Interface:

Siehe hierzu die Ausführungen aus dem ersten Teil „OO Grundlagen“.

4.5. Assoziation

Assoziationen beschreiben Verbindungen zwischen Klassen. Sie sind notwendig damit die Objekte miteinander kommunizieren können. Häufig sind diese Beziehungen fachlich motiviert.

Die konkrete Beziehung zwischen zwei Objekten wird Objektverbindung (engl. Link) genannt. Sie ist damit eine Instanz einer Assoziation.

Beispiel:

Eine Firma besteht aus einer Menge von Mitarbeitern und ein Mitarbeiter ist genau bei einer Firma tätig. Dies wird durch die entsprechende Multiplizität festgehalten.

Der Beziehungsname zeigt, dass die Firma die Mitarbeiter beschäftigt. Dies wird durch die Rollenbezeichnungen Arbeitgeber für die Firma und Arbeitnehmer für die Mitarbeiter noch zusätzlich unterstrichen.

Notation:

Assoziationen werden durch eine Linie zwischen den beteiligten Klassen dargestellt. Die Linie wird mit einem Namen versehen, der beschreibt, worin und warum diese Beziehung besteht. Der Name der Assoziation kann mit einem Stereotypen ergänzt werden sowie durch Zusicherungen und Merkmale genauer beschreiben werden.

Mit der Angabe von Rollennamen, Multiplizität oder Richtungsangaben kann die Art der Beziehung näher definiert werden.

Die nachfolgende Tabelle zeigt die möglichen Angaben für die **Multiplizität**. Dabei sind neben den gebräuchlichen Angaben auch Aufzählungen oder Bereichsdefinitionen möglich:

Multiplizität	Bedeutung
0..1	Keine oder eine Instanz. Die Notation n..m bedeutet: n bis m Instanzen.
0..* oder *	Eine beliebige Anzahl von Instanzen (auch keine).
1	Genau eine Instanz.
1..*	Eine beliebige Anzahl Instanzen aber mindestens eine.
n..m	Bereich (z. B. 3..5, 1..7, etc.)
n1, n2, n3 …	Aufzählung (z.B. 5,10,15)
n1, n2…n3	Aufzählung und Bereiche gemischt (z.B. 7, 10..15, 20, 25)

Gerichtete Assoziation

Wird eine Verbindungslinie mit einem Pfeil ergänzt, spricht man von einer **gerichteten Assoziation**. Damit wird angegeben, in welcher Richtung die Assoziation navigiert oder abgefragt werden kann. Assoziationen ohne Richtungsangabe sind bidirektional.

Beispiel:

Eine Bestellposition kann über Ihren Artikel abgefragt werden und nicht umgekehrt.

Der Pfeil gibt auch an, wer Besitzer der Beziehung ist. In unserem Fall besitzt eine Bestellposition einen Artikel.

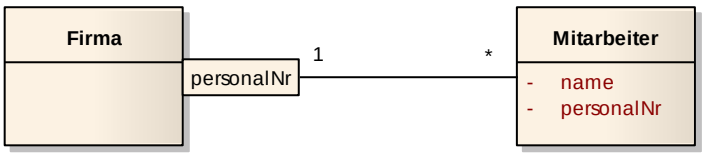
Qualifizierte Assoziation:

Beziehungen, bei denen ein Objekt viele (*) Objekte der gegenüberliegenden Seite assoziieren kann, werden durch Behälterobjekte implementiert.

Möchte man nun im Design angeben, über welche Schlüsselbegriffe (engl. Qualifier) der Zugriff auf ein spezifisches Element erfolgt, so kann dies explizit mit einer qualifizierten Assoziation angegeben werden. Dabei sind auch mehrere Schlüsselbegriffe erlaubt.

Qualifizierte Assoziationen können nur für binäre Beziehungen definiert werden.

Beispiel:



Eine Firma verfügt über mehrere Mitarbeiter, die mit Hilfe Ihrer Personalnummer eindeutig identifiziert werden können.

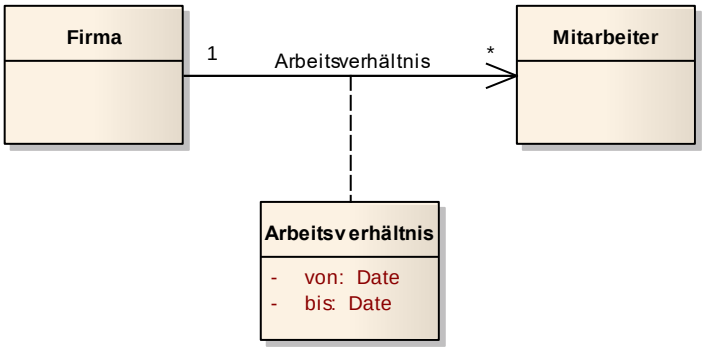
Im Beispiel wird diese Nummer nun für den Zugriff auf einen spezifischen Mitarbeiter verwendet.

Attributierte und mehrgliedrige Assoziation:

Attributierte Assoziation:

Es existiert auch eine Form in der die Assoziation selbst über Attribute verfügt. Diese Assoziationsattribute sind dann existenzabhängig von der Assoziation. Man spricht von sogenannten attributierten Assoziationen, bei der alle Eigenschaften der Klasse der Beziehung zugeordnet werden. Klasse und Beziehung verwenden dabei den gleichen Namen.

Beispiel:

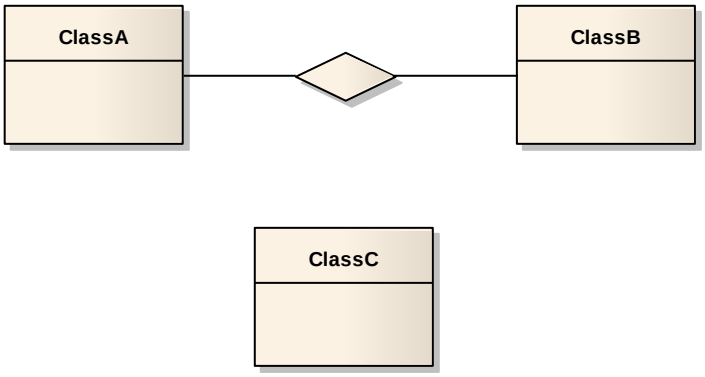


Die Beziehung Firma Mitarbeiter wird mit Informationen zum Arbeitsverhältnis erweitert. Die Attribute gehören weder zur Firma noch zum Mitarbeiter. Für die Umsetzung muss die Assoziationsklasse in eine richtige Klasse umgewandelt werden.

Mehrgliedrige Assoziationen:

Neben den Zweierbeziehungen gibt es abgesehen von den attributierten Assoziationen noch drei- oder mehrgliedrige Assoziationen an denen drei oder mehr Klassen (resp. Assoziationsrollen) beteiligt sind.

Beispiel:



Für die Umsetzung des Beispiels müsste diese Assoziation in drei binäre Assoziationen umgewandelt werden.

Hinweis:

Attributierte und mehrgliedrige Assoziationen sollten wenn möglich vermieden werden, da sie nicht den objektorientierten Prinzipien entsprechen und in der Praxis fehleranfällig sind.

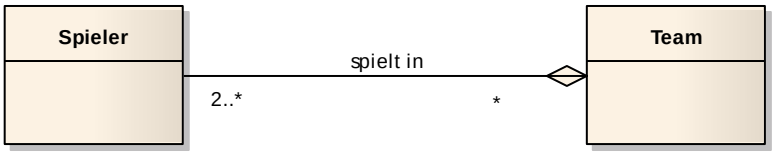
4.6. Aggregation

Eine Aggregation ist die Zusammensetzung eines Objektes aus einer Menge von Einzelteilen und eine Spezialform der Assoziation. Die Lebensdauer der Einzelteile kann dabei länger sein als die Lebensdauer der Aggregate

In der Aggregation nimmt das Ganze stellvertretend für seine Teile Aufgaben wahr. Die Aggregatklasse kann Operationen enthalten, die keine unmittelbare Wirkung im Aggregat selbst erzeugen, sondern die entsprechenden Nachrichten an seine Teile weiterleiten.

Die beteiligten Klassen führen keine gleichberechtigte Beziehung. Stattdessen bekommt die Aggregatklasse eine verantwortliche und führende Rolle. In einer Aggregationsbeziehung muss an einem Ende das Aggregat stehen und am anderen Ende die zugehörigen Teile.

Beispiel:



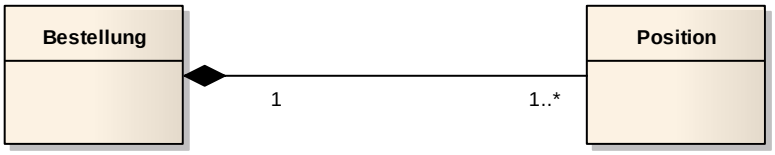
Notation:

Die Aggregation wird als Linie zwischen zwei Klassen dargestellt, und zusätzlich mit einer Raute versehen. Die Raute steht auf der Seite des Aggregats (des Ganzen) und symbolisiert das Behälterobjekt, in dem die Teile gesammelt sind.

4.7. Komposition

Eine Komposition ist eine Aggregation mit existenzabhängigen Teilen. Bei einer Komposition kann ein Teil immer nur in genau einem zusammengesetzten Objekt enthalten sein. Die Lebensdauer des zusammengesetzten Objekts entspricht immer der Lebensdauer seiner Komponenten.

Beispiel:



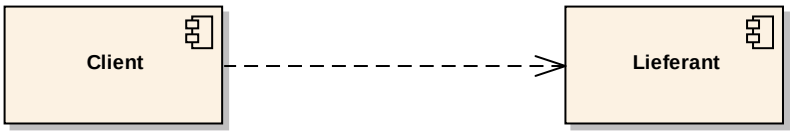
Notation:

Die Komposition wird als Linie zwischen zwei Klassen gezeichnet. Die Linie wird auf der Seite des Aggregates mit einer ausgefüllten Raute versehen. Mehrere Kompositionsbeziehungen zu einem Ganzen können baumartig zusammengefasst werden

4.8. Abhängigkeit

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 bewirkt. Die Abhängigkeit bezieht sich dabei auf die Modellelemente selbst und nicht auf eventuelle Instanzen dieser Elemente.

Beispiel:



Notation:

Eine Abhängigkeit wird durch einen gestrichelten Pfeil, der auf das unabhängige Element zeigt, dargestellt. Die Gründe für die Abhängigkeit können mit Hilfe von Stereotypen, wie zum Beispiel mit den folgenden, näher beschreiben werden:

- **«**call**»** Aufruf von Operationen des unabhängigen Elements.
- **«**create**»** Erstellung von Instanzen des Lieferanten (engl. Supplier)
- **«**derive**»** Der Client kann vom Lieferant abgeleitet werden.

4.9. Realisation

Eine Realisierungsbeziehung ist eine Beziehung zwischen einem Element, das eine Anforderung beschreibt und einem Element, das diese Anforderung umsetzt.

Beispiel:

Ein Beispiel ist die Implementation einer Schnittstelle (engl. Interface).

Das Interface definiert den Vertrag (engl. Contract) und das angegebene Element stellt die entsprechende Funktionalität zur Verfügung.

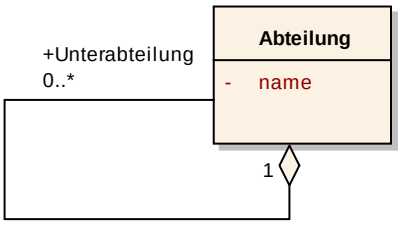
Notation:

Die Realisierungsbeziehung wird als Abhängigkeit mit einer gestrichelten Linie und offenem Pfeil dargestellt. In früheren UML Versionen wurde zusätzlich der Stereotyp **«**realize**»** angegeben.

5. Objektdiagramm

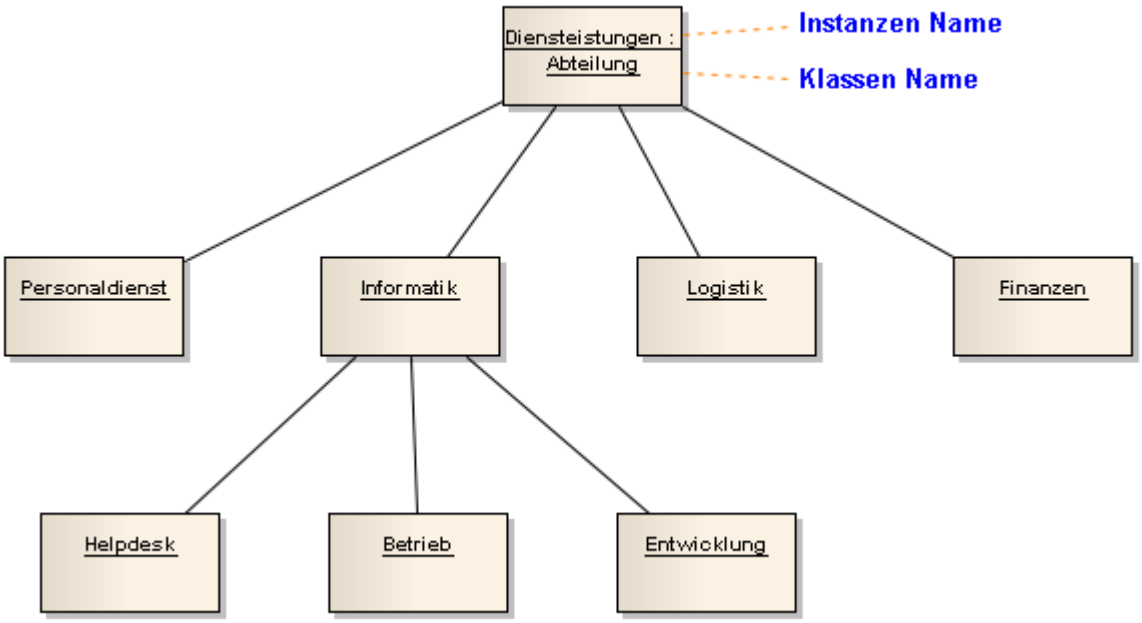
Objektdiagramme zeigen Instanzen anstelle von Klassen. Objektdiagramme sind nützlich, um kleine Teile mit komplexen Sachverhalten (wie zum Beispiel rekursive Beziehungen) aufzuzeigen.

Beispiel Abteilung:



Das obige **Klassendiagramm** zeigt eine Abteilung die ihrerseits wiederum eine beliebige Anzahl Unterabteilungen enthalten kann.

Das folgende **Objektdiagramm** zeigt nun eine konkrete Anwendung mit Instanzen, welche dem gezeigten Klassendiagramm entsprechen:



Die Abteilung Dienstleistungen besteht aus vier Unterabteilungen. Die Informatik Ihrerseits ist wiederum in weitere Unterabteilungen gegliedert.

Notation:

Jedes Rechteck des Objekt Diagramms repräsentiert eine einzelne Instanz. Die Instanz Namen werden in UML unterstrichen dargestellt. Klassen oder Instanzen Namen können weggelassen werden, sofern die Bedeutung des Diagramms klar bleibt.

6. Paketdiagramm

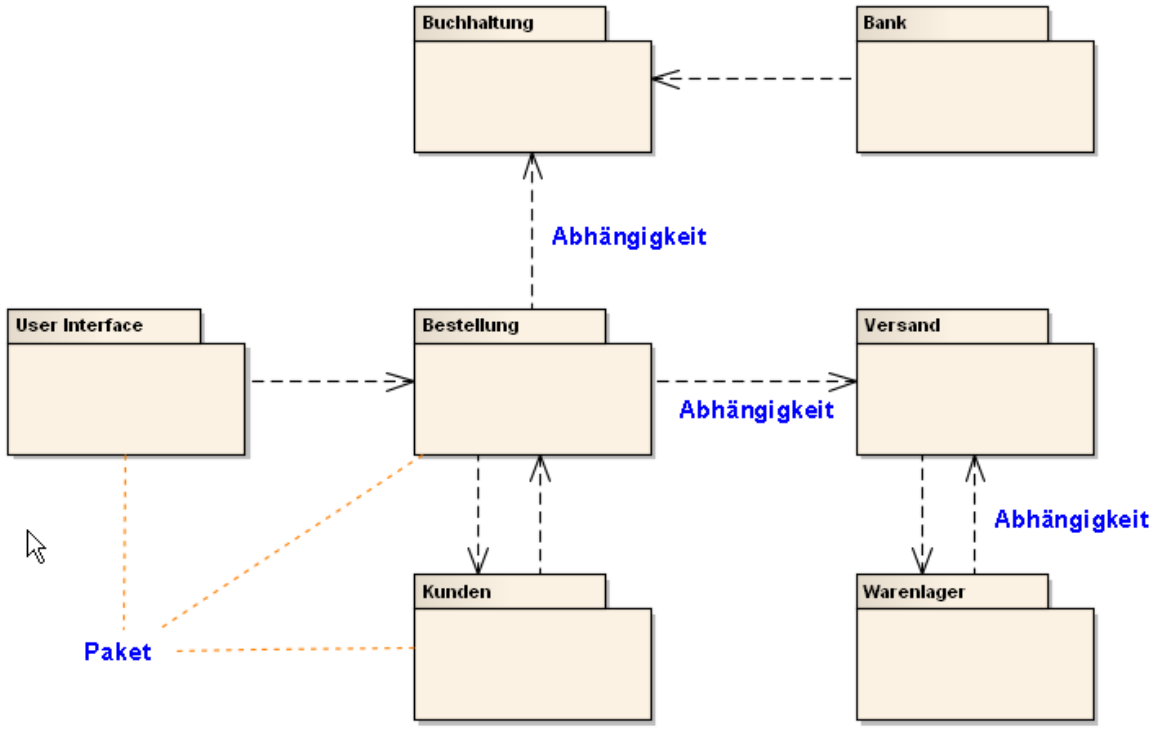
Zur Vereinfachung von komplexen Diagrammen, können diese in sogenannte Pakete unterteilt werden. Ein Paket ist dabei eine Sammlung von logisch zusammengehörenden UML Elementen (z.B. Klassen oder Anwendungsfälle).

Pakete können hierarchisch gegliedert werden, also ihrerseits wieder Pakete enthalten. Pakete werden aufgrund physischer oder logischer Zusammenhänge gebildet. Vorhandene Bibliotheken, Untersysteme und Schnittstellen bilden jeweils eigene Pakete. Wird das eigentliche Modell zu groß, kann es ebenfalls nach logischen Gesichtspunkten in Pakete gegliedert werden.

Ein Paket definiert einen Namensraum und gruppiert UML Elemente. So gehört eine Klasse zum Beispiel immer zu einem bestimmten Pakt. Wird sie nun in Klassen von anderen Paketen referenziert erfolgt dies über Ihren qualifizierten (Paketname::Klassenname) Namen.

Dadurch entstehen Abhängigkeiten zwischen den Paketen, beispielsweise nutzt eine Klasse Klassen anderer Pakete. Modelliert man eine gute Architektur des Gesamtsystems, führt das zu wenigen Abhängigkeiten zwischen den Paketen.

Beispiel Versandfirma:



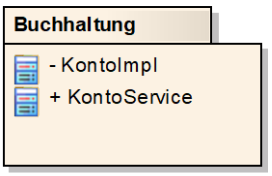
Das Beispiel zeigt das Geschäftsmodell einer Versandfirma welches in logisch sinnvolle Einheiten unterteilt ist.

Notation:

Pakete werden als Rechteck mit einem Register am oberen Rand dargestellt. Der Paketname wird im Register oder in Rechteck selber angegeben. Vor oder oberhalb des Namens können Stereotypen notiert werden. Die gestrichelten Linien geben die Abhängigkeiten der Pakete untereinander an.

Sichtbarkeit:

Mit der Sichtbarkeit wird angegeben, welche Elemente des Pakets von aussen sichtbar sind.

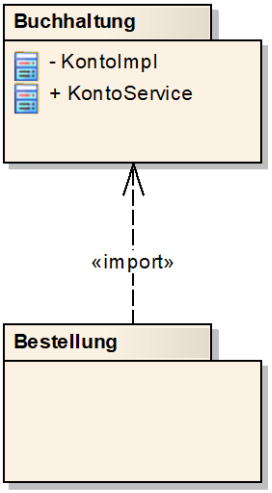


Das Interface KontoService ist public und wird von aussen über den qualifizierten Namen Buchhaltung::KontoService referenziert.

Die Implementation KontoImpl ist private und von aussen nicht sichtbar.

Import:

Mit einem Paket Import werden alle Namen der öffentlichen Elemente eines Pakets dem importierenden Paket als **öffentlich** hinzugefügt.



Im Paket Bestellung wird mit der Import Beziehung der Name vom KontoService Interface hinzugefügt.

Innerhalb der Bestellung kann nun der Kontoservice folgendermassen referenziert werden:

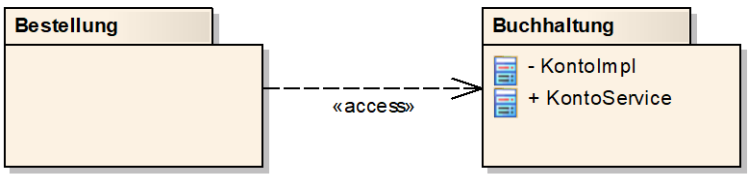
- ohne «import» → Buchhaltung::KontoService
- mit «import» → KontoService

Mit «import» ist KontoService ein öffentlicher Name von Bestellung.

Klassen von anderen Paketen die auf das Paket Bestellung zugreifen, haben jetzt auch, über den qualifizierten Namen Bestellung::KontoService, Zugriff auf den KontoService.

Access:

Mit einem Paket Access werden alle Namen der öffentlichen Elemente eines Pakets dem importierenden Paket als **privat** hinzugefügt. Ein Zugriff via Bestellung::KontoService ist somit ausgeschlossen.



Beachte:
Der Paket Zugriff via «access» ist der einzige, der sowohl von Java als auch C# unterstützt wird.

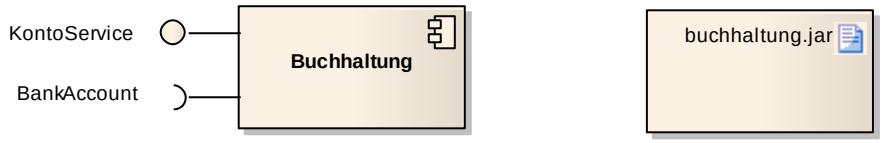
7. Komponentendiagramm

Komponente

Eine Komponente repräsentiert einen modularen Bestandteil von einem ganzen System. Die interne Struktur kann aus einigen wenigen Klassen bis zu grossen Softwaresystemen reichen und wird gegen aussen gekapselt. Dazu definiert eine Komponente die zur Verfügung gestellten Schnittstellen sowie die benötigten Schnittstellen anderer Komponenten.

Die Darstellung kann wie im Beispiel gezeigt als Blackbox oder auch in Tabellenform erfolgen. Es können logische Komponenten oder auch konkrete Artefakte wie z.B. eine Java Archiv Datei als Komponente Dargestellt werden.

Beispiel:



Notation:

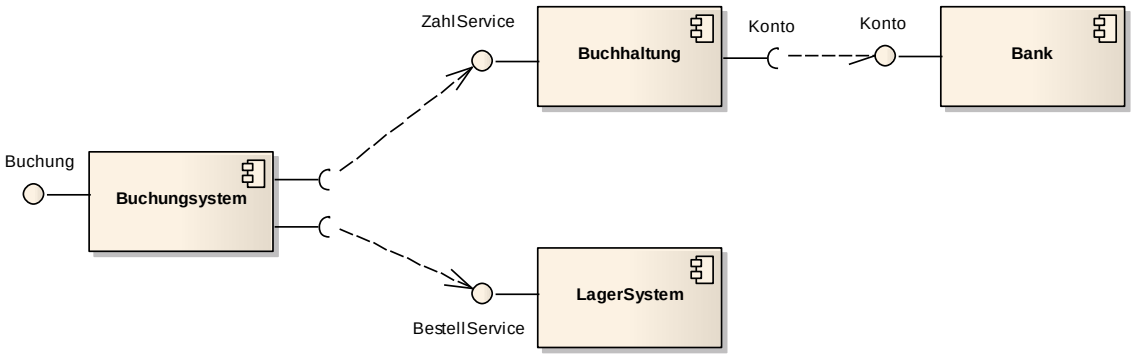
Bei der Kompakten Notation wird eine Komponente als Rechteck mit dem Namen dargestellt und mit dem Stereotyp «component» gekennzeichnet. Alternativ kann auch ein visuelles Objekt in der rechten oberen Ecke zur Kennzeichnung verwendet werden. Zusätzlich können die zur Verfügung gestellten und benötigten Schnittstellen mit den entsprechenden Symbolen angegeben werden.

Alternativ kann die Notation auch in Tabellenform erfolgen. Die Schnittstellen werden dann über die beiden Stereotypen «provided interfaces» und «required interfaces» innerhalb der Komponente dargestellt. Zudem stehen zusätzliche Stereotypen für die Angabe von Klassen Artefakten zur Verfügung.

Komponentendiagramm:

Mit dem Komponentendiagramm wird die Aufteilung / Gruppierung des Systems in verschiedene Komponenten sowie deren Abhängigkeiten dargestellt.

Beispiel:



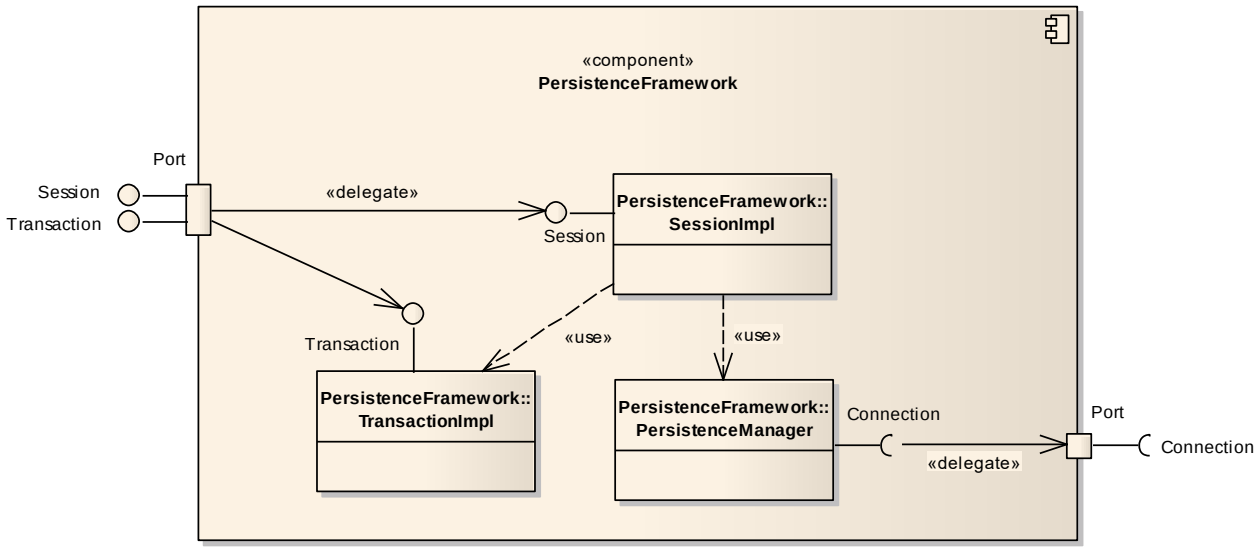
Notation:

Die Abhängigkeiten zwischen den einzelnen Komponenten werden durch gestrichelte Pfeile symbolisiert.

Konnektor:

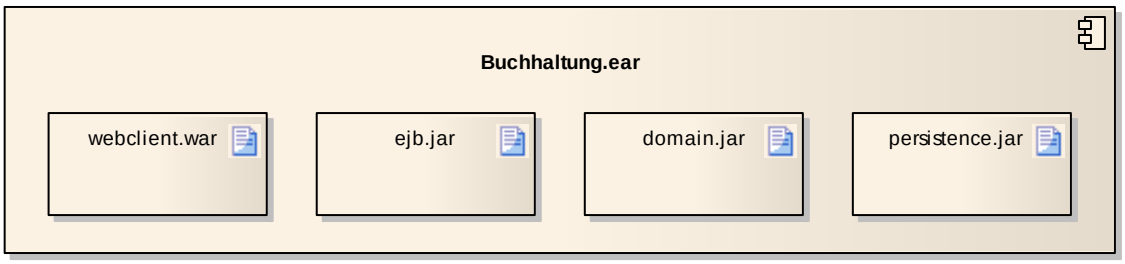
Mit einem Konnektor (engl. Assembly Connector) wird eine Verbindung zwischen zwei Bauteilen dargestellt. Ein Bauteil ist dabei ein Element eines Komponentendiagrammes das Schnittstellen oder Ports besitzen kann.

Mit Hilfe eines sogenannten Delegationskonnektor (engl. Delegation Connector) wird die Verbindung dargestellt. Im nachfolgenden Beispiel werden somit die Ports mit den externen Schnittstellen mit den inneren Bestandteilen der Komponente verbunden.



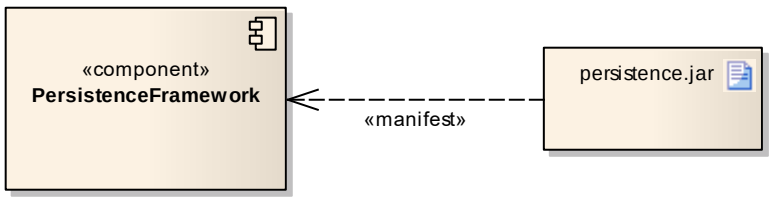
Artefakt und Manifest:

Mit einem Artefakt wird eine physische Informationseinheit symbolisiert, die bei der Softwareentwicklung erstellt wird (z.B. Source Code, Archive, Konfiguration, Dokumente).



Manifest:

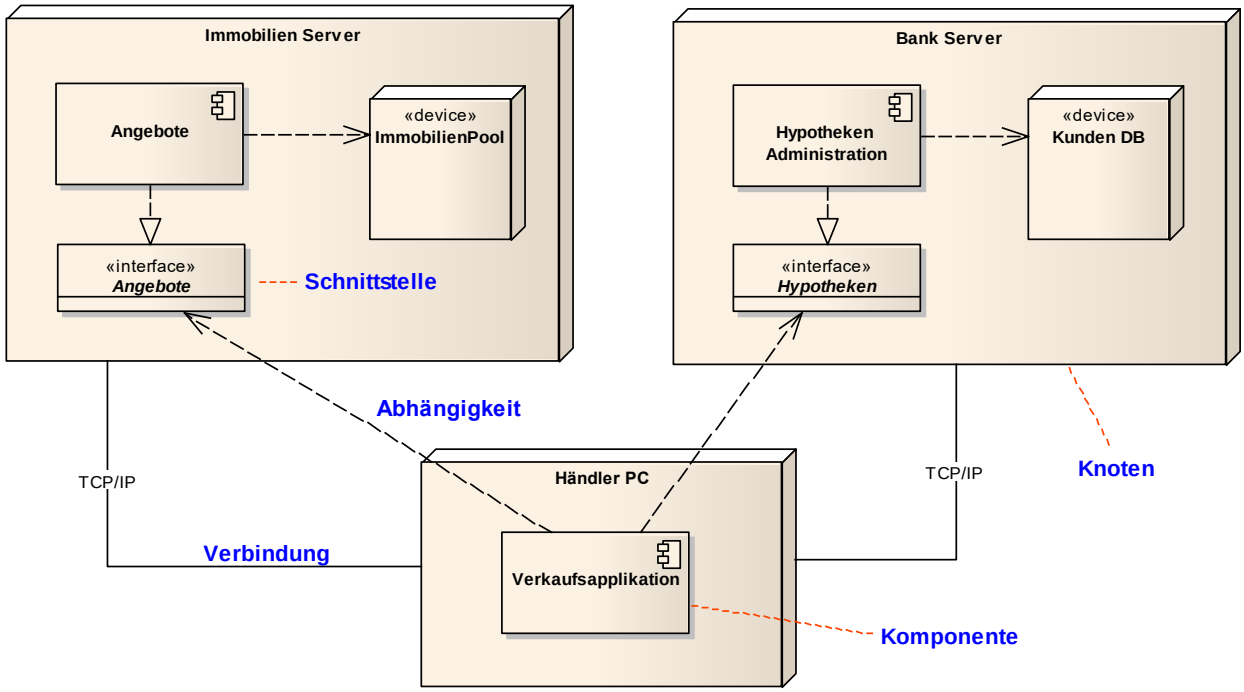
Mit der Manifest Beziehung wird ein Modellelement mit einem Artefakt verbunden, dass die Komponente realisiert.



8. Einsatz- und Verteilungsdiagramm

Mit einem Einsatz- / Verteilungsdiagramm wird die physikalische Konfiguration der Soft- und Hardware aufgezeigt. Es zeigt welche Komponenten auf welchen Knoten installiert sind und welche Abhängigkeiten bestehen.

Beispiel Immobilienverkauf:



Das Beispiel zeigt die Soft- und Hardwarekomponenten für den Verkauf von Immobilien.

Notation :

Die sogenannten Knoten (engl. Nodes) zeigen die Hardware. Die Komponenten zeigen die involvierten Software Module. Die Knoten werden als dreidimensionale Rechtecke (Quader) dargestellt.

In den Knoten können die dort ablaufenden Komponenten dargestellt werden, wobei auch Schnittstellen und Abhängigkeitsbeziehungen zwischen den Elementen erlaubt sind. Knoten die miteinander kommunizieren werden durch Linien miteinander verbunden.

Mit Hilfe von Stereotypen können die verschiedenen Arten von Knoten näher beschrieben werden, wie zum Beispiel mit «device» für eine Physische Recheneinheit, «execution environment» für eine Ausführungseinheit wie zum Beispiel den Tomcat Server oder «mobile device» für mobile Endgeräte.

9. Anwendungsfalldiagramm

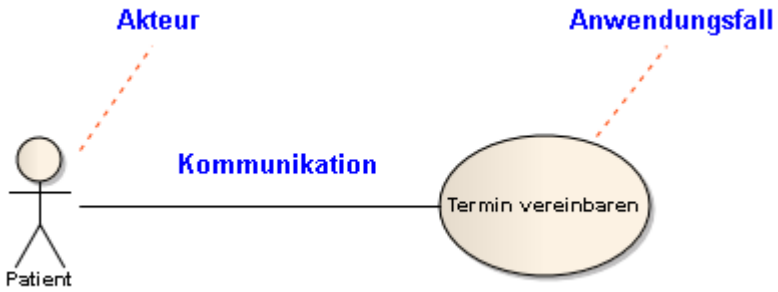
Anwendungsfalldiagramme beschreiben was ein System aus Sicht eines externen Beobachters macht. Der Fokus liegt auf dem Was und nicht auf dem Wie.

Ein Anwendungsfall Diagramm zeigt die Übersicht von Szenarien eines Systems die durch einen Akteur ausgelöst werden. Akteure sind dabei einfache Rollen die entweder durch Leute oder Systeme wahrgenommen werden. Die Verbindung zwischen Akteur und Anwendungsfall wird Kommunikationsverbindung oder einfach Kommunikation genannt.

Anwendungsfalldiagramme sind neben der Beschreibung eines Ablaufs als Anforderung, u. a. für folgende Bereiche hilfreich:

- **Ermittlung von neuen Anforderungen**
  Neue Anwendungsfälle die während der Analyse auftreten führen oft zu neuen Anforderungen.
- **Kommunikation mit dem Kunden**
  Die einfache und übersichtliche Darstellung der Anwendungsfälle eignet sich sehr gut für Besprechungen zwischen Kunden, Business Analysten und Entwicklern.
- **Erstellung von Testfällen**
  Die Übersicht von Szenarios kann ein guter Anhaltspunkt für Gruppierung von Testfällen sein.
- **Dokumentation**
  Anwendungsfälle können auch gut Bestandteil von Prozess- und Ablaufdokumentationen sein und auch wertvolle Informationen für Benutzerdokumentationen liefern.
- **Projektmanagement**
  Je nach Vorgehen, können anhand der Anwendungsfälle Aufwandschätzungen, die Planung von Iterationen und Risikoidentifikationen gemacht werden.

Beispiel Termin vereinbaren:



Ein Patient ruft in der Zahnarztpraxis an um den jährlichen Kontrolltermin zu vereinbaren. Der Empfang sucht den nächsten freien Termin im Kalender und reserviert diesen für den Patienten.

Notation:

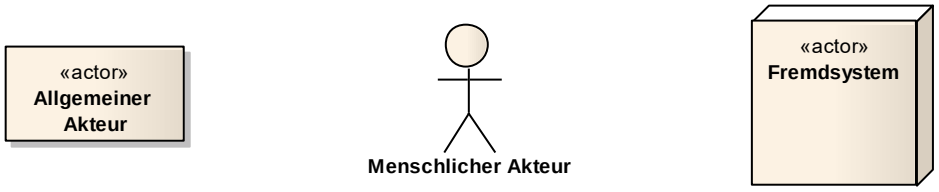
Ein Anwendungsfalldiagramm enthält eine Menge von Anwendungsfällen und Akteuren die daran beteiligt sind. Die Anwendungsfälle werden als Ellipse dargestellt, die Akteure könnten textuell mit einem Rechteck oder mit verschiedenen visuellen Symbolen dargestellt werden.

Die Anwendungsfälle werden durch Linien mit den Akteuren verbunden. Ein einzelner Anwendungsfall kann dabei auch mehrere Akteure haben. Die Systemgrenzen werden mit einem Rahmen um die Anwendungsfälle symbolisiert.

Akteure:

Akteure stellen nicht konkrete Personen dar sondern Rollen, d.h. ein Akteur ist eine Rolle, die im Kontext des jeweiligen Anwendungsfalles eine Aktion auslöst.

Wie bereits erwähnt gibt es dafür verschiedene Darstellungsmöglichkeiten:



Akteure können untereinander Generalisierungs- / Spezialisierungsbeziehungen haben. Damit lassen sich hierarchische Gliederungen und Abstraktionen zwischen den Rollen abbilden.

Multiplizität:

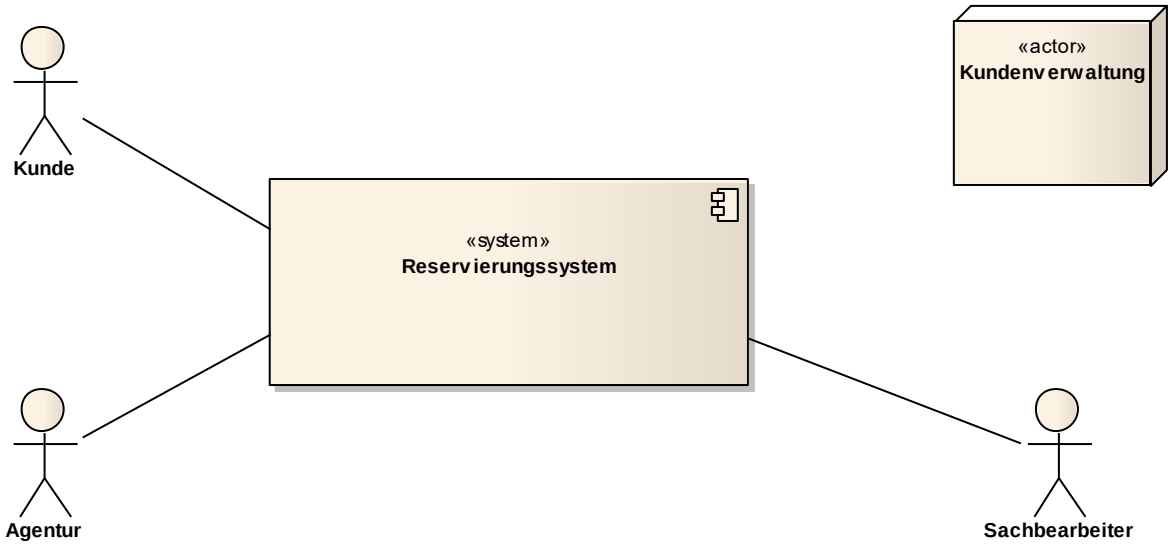
Bei der Assoziation zwischen Akteur und Anwendungsfall kann auch die Multiplizität festgelegt werden.

Die Angabe auf Seite des **Anwendungsfalles** gibt an, wie oft dieser Anwendungsfall vom Akteur **gleichzeitig** ausgeführt werden darf. Fehlt die Angabe, so ist die Multiplizität 0..1.

Auf Seite des **Akteurs** bedeutet die Multiplizität, wie viele Akteure der angegebenen Rolle am Anwendungsfall **beteiligt** sein müssen bzw. können. Fehlt die Angabe, so ist die Multiplizität 1.

Systemkontext:

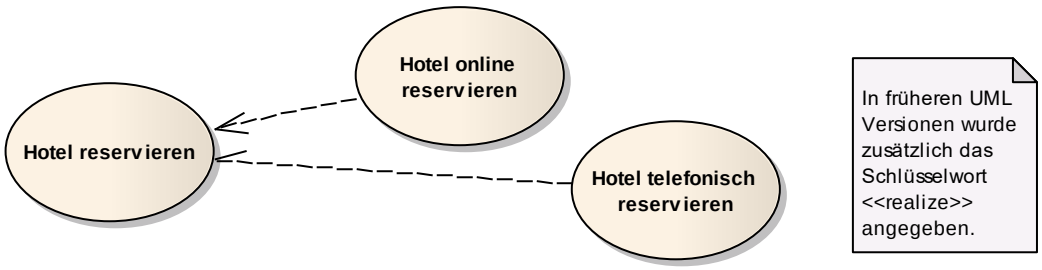
Im UML gibt es dazu kein eigener Diagrammtyp. Stattdessen verwendet man ein Anwendungsfalldiagramm mit Elementen der Klassen- Komponenten- und Verteil-Diagramme. Die Klasse oder Komponente repräsentiert dabei das zu erstellende System und alle Nutzer des Systems werden als Akteure mit einer Beziehung zum System modelliert.



Realisierung:

Eine Realisierungsbeziehung ist eine Beziehung zwischen einem Element, das eine Anforderung beschreibt und einem Element, das diese Anforderung umsetzt. Die Realisierungsbeziehung wird als Abhängigkeit mit einer gestrichelten Linie und offenem Pfeil dargestellt.

Realisierungen kann man auch im Falle von Anwendungsfalldiagrammen einsetzen. Im folgenden Beispiel gibt es zwei unterschiedliche Realisationen für den Anwendungsfall „Hotel reservieren“.

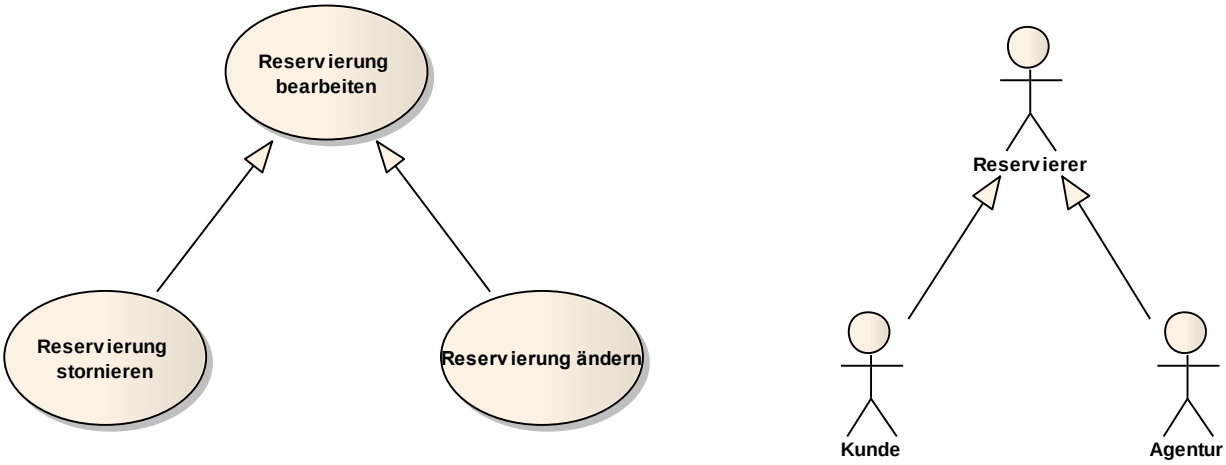


Spezialisierung:

Spezialisierung und Generalisierung sind Abstraktionsprinzipien zur hierarchischen Strukturierung der Semantik von Modellen.

Die Spezialisierung ist eine Beziehung zwischen dem allgemeinen und speziellen Element, wobei das spezielle Element weitere Eigenschaften hinzufügt und sich kompatibel zum allgemeinen Element verhält.

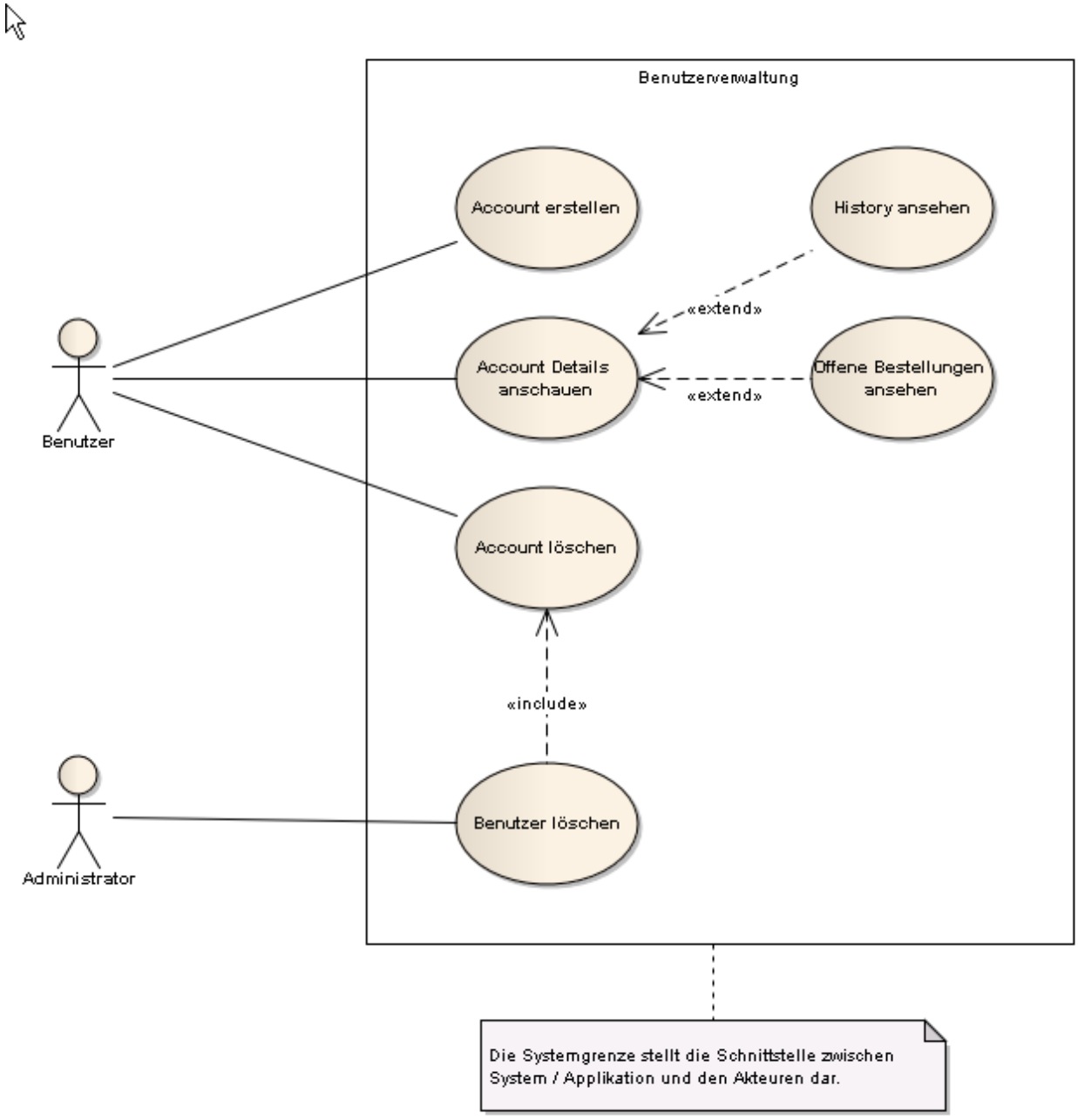
Bei den Anwendungsfalldiagrammen kann die Spezialisierung sowohl für Anwendungsfälle als auch (wie bereits erwähnt) für Akteure eingesetzt werden.



Enthält- und Erweiterungsbeziehung:

Mit einer Enthältbeziehung (engl. include) wird ein Anwendungsfall in einen anderen Anwendungsfall mit eingebunden und logischer Teil von diesem.

Mit einer Erweiterungsbeziehung (engl. extend) hingegen lässt sich ausdrücken, dass ein Anwendungsfall unter bestimmten Umständen an einer bestimmten Stelle (dem sogenannten Extension Point) ggf. mit einem anderen Anwendungsfall erweitert wird.



Ein Benutzer kann einen Account erstellen, seine Details anschauen und den Account auch wieder löschen. Die beiden Anwendungsfälle „History ansehen“ und „offene Bestellungen ansehen“ bilden dabei eine Erweiterung vom Anwendungsfall „Account Detail anschauen“.

Zusätzlich kann der Systemadministrator Benutzer löschen. Der entsprechende Anwendungsfall „Benutzer löschen“ beinhaltet u. a. auch den Anwendungsfall „Account löschen“.

10. Aktivitätsdiagramm

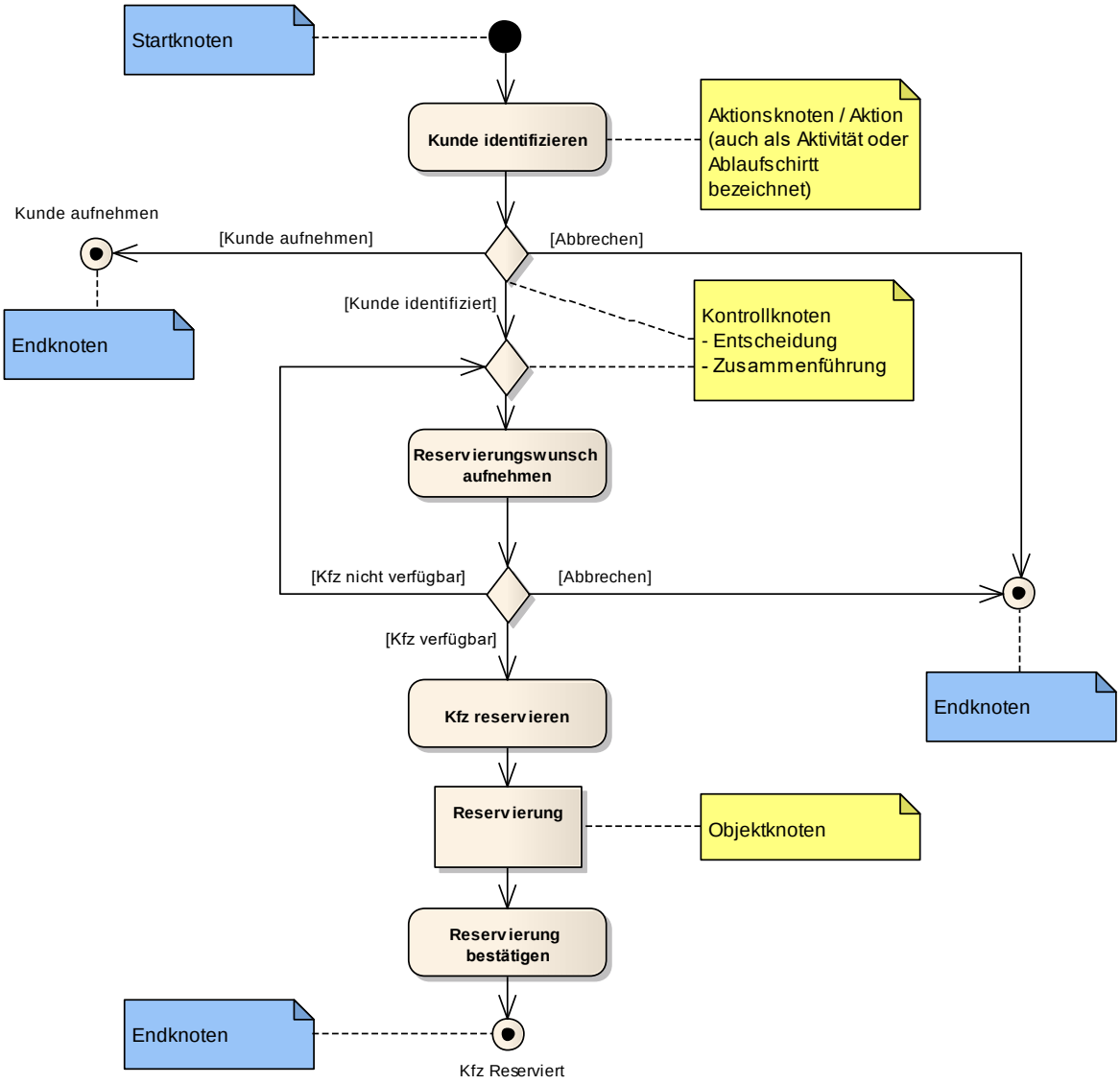
10.1. Aktivitätsdiagramm

Das Aktivitätsdiagramm beschreibt einen Ablauf und wird definiert durch verschiedene Arten von Knoten, die durch Objekt- und Kontrollflüsse miteinander verbunden sind. Dabei werden Aktions-, Objekt- und Kontrollknoten unterschieden.

Das Aktivitätsdiagramm ist verwandt mit dem Zustandsdiagramm, legt den Fokus aber nicht auf die verschiedenen Zustände eines Objektes, sondern auf Verlauf von Aktivitäten innerhalb eines Prozesses oder Algorithmus.

Das Aktivitätsdiagramm eignet sich sehr gut für das Aufzeigen von Geschäftsprozessen während der Erhebung der Anforderungen und Analyse der der Arbeitsabläufe.

Beispiel:



Notation:

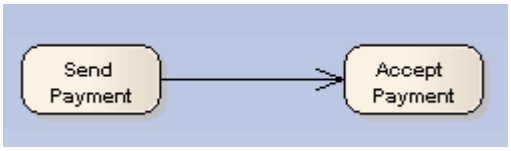
Ein Aktivitätsdiagramm besteht aus Start- und Endknoten, einer Reihe von Aktion-, Objekt- oder Kontrollknoten sowie Objekt- und Kontrollflüssen, die die einzelnen Knoten verbinden.

10.2. Aktionsknoten / Aktionen

Eine Aktion beschreibt einen einfachen Prozess oder eine Transformaton die innerhalb des Systems stattfindet. Aktionen können als Kind Elemente von Aktivitäten angesehen werden.

Beide (Aktionen und Aktivitäten) repräsentieren einen Prozess, während dem Aktivitäten aber aus weiteren Teilprozessen oder Aktivitäten bestehen können, kann eine Aktion nicht weiter unterteilt werden.

Beispiel:

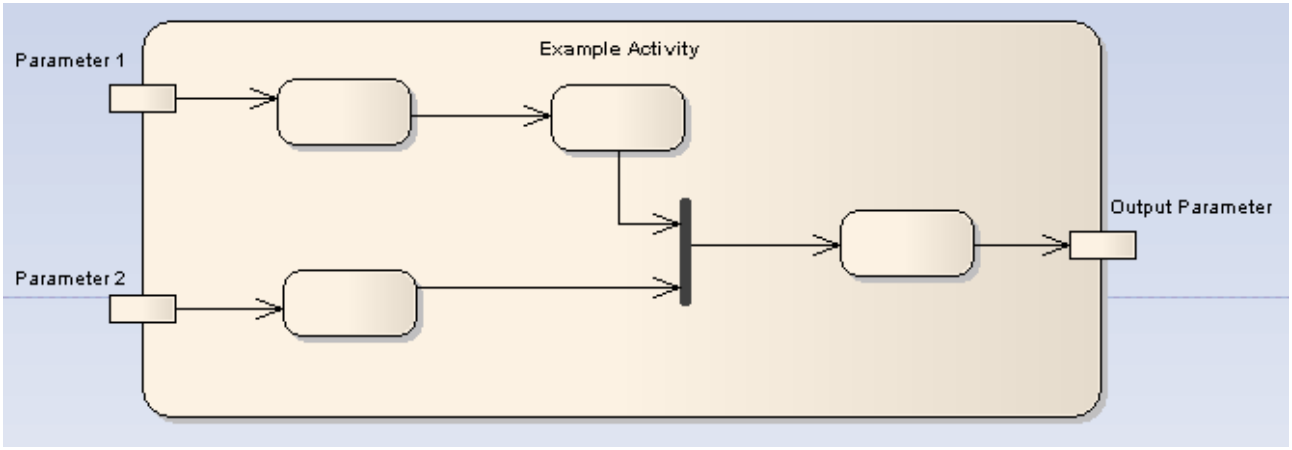


Für eine Aktion können Vor- und Nachbedingungen (engl. Pre and Post Conditions) definiert werden. Die Aktion wird dann erst ausgeführt, wenn die Vorbedingung erfüllt ist. Nach der Ausführung geht der Kontrollfluss erst weiter, wenn die Nachbedingung erfüllt ist.

10.3. Aktivitäten

Eine Aktivität organisiert oder spezifiziert weitere Sub-Aktivitäten oder Aktionen und den dazugehörigen Kontrollfluss. Sie dienen zur Strukturierung des Workflows und zur Gruppierung von zusammengehörenden Einheiten.

Beispiel:



Aktivitäten können mit einem Namen gekennzeichnet werden und Ein- und Ausgabeparameter definieren. Es können zudem gleich wie bei der Aktion auch für eine ganze Aktivität Vor- und Nachbedingungen formuliert werden.

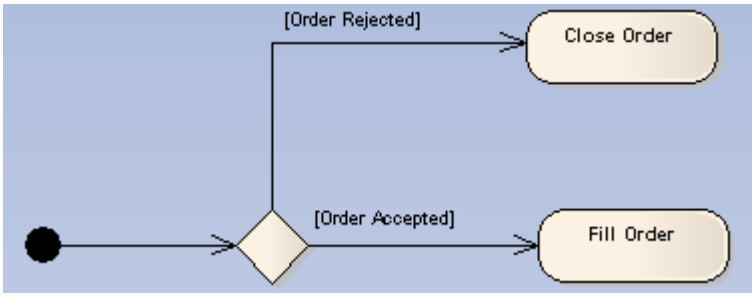
10.4. Kontrollflüsse

Kontrollflüsse können mit Konnektoren versehen werden, um lange, quer durch ein Diagramm laufende Linien zu vermeiden oder Kontrollflüsse auf einem anderen Blatt fortzuführen.

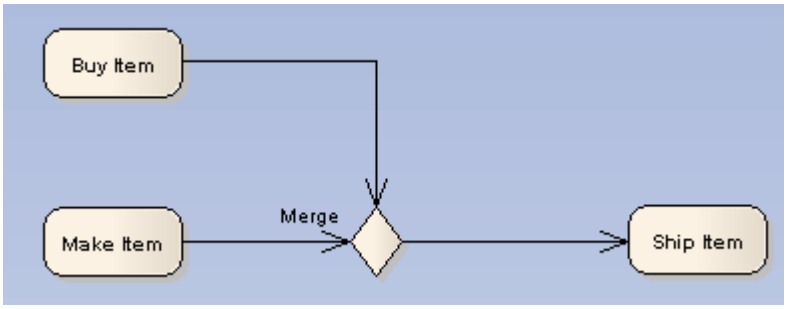
10.5. Kontrollknoten

Anbei einige Beispiel zu möglichen Kontrollknoten:

***Entscheidung**:*

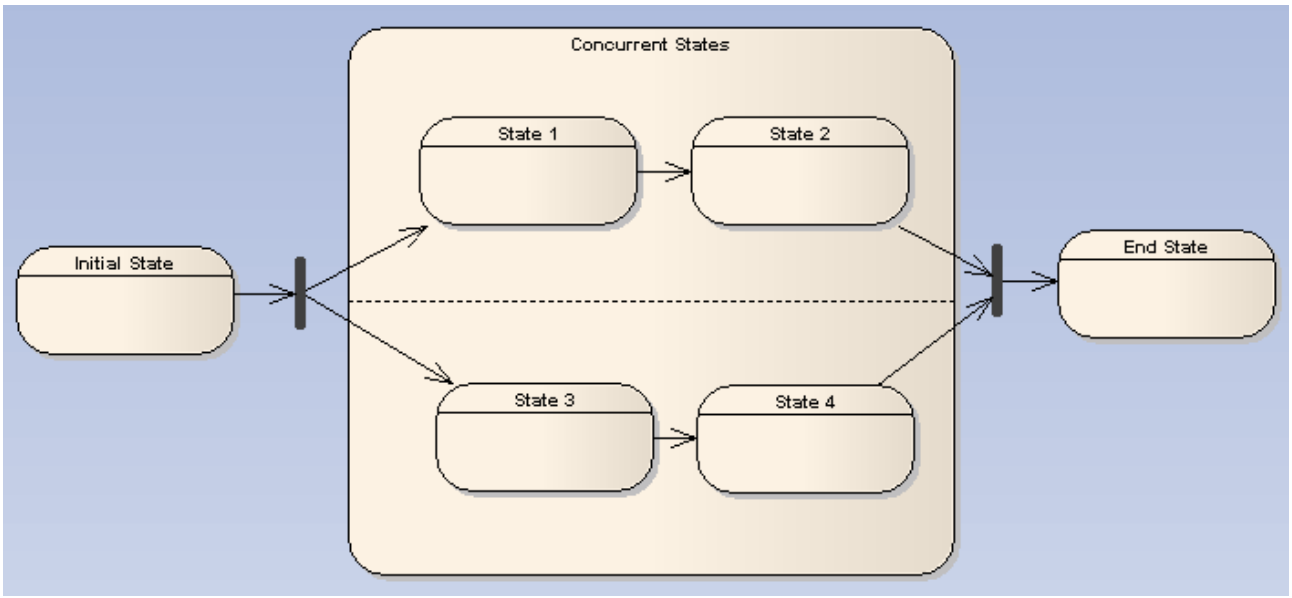


***Vereinigung**:*



Vergabelung und Synchronisation:

Um Nebenläufige Prozess (Parallelbetrieb) zu modellieren, kann eine Vergabelung (engl. Fork) eingesetzte werden. Das Gegenstück bildet die Synchronisation (engl. Join), die die konkurrierenden Prozesse wieder synchronisiert.



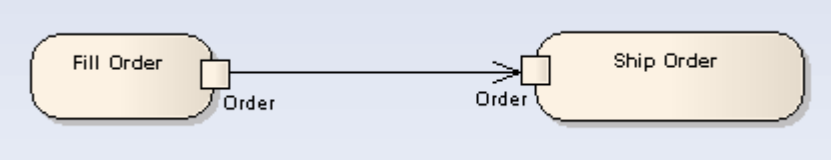
Es sind auch Mischformen möglich, so dass zum Beispiel mehrere Abläufe synchronisiert werden und anschliessend gleich wieder parallele Abläufe ausgelöst werden.

Im Weiteren ist es möglich, eine Synchronisation nach spezifischen Bedingungen durchzuführen. Beispiel für drei eingehenden Kontrollflüssen A, B und C: {joinSpec = (A and B) or (A and C) }

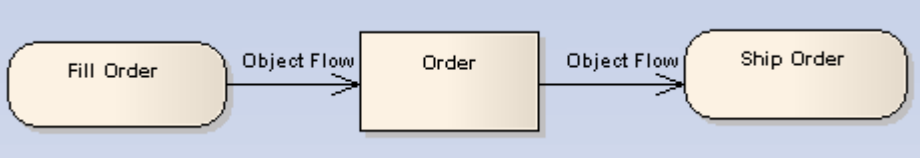
10.6. Objektknoten und Objektflüsse

Ein Objektknoten gibt an, das ein oder mehrere Objekte existieren. Sie können als ein- und ausgehende Parameter in Aktivitäten verwendet werden.

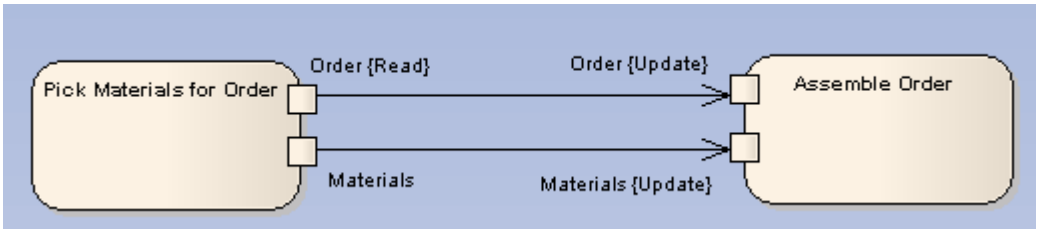
Objektknoten werden durch sogenannte Pins dargestellt. Dies sind kleine Rechtecke am Ein- beziehungsweise Ausgang der Aktionsknoten.



Alternativ kann anstatt der Pins auch ein Objektknoten (dargestellt durch ein Rechteck) verwendet werden:



Das nachfolgende Beispiel zeigt die Notation bei mehreren Objekten:

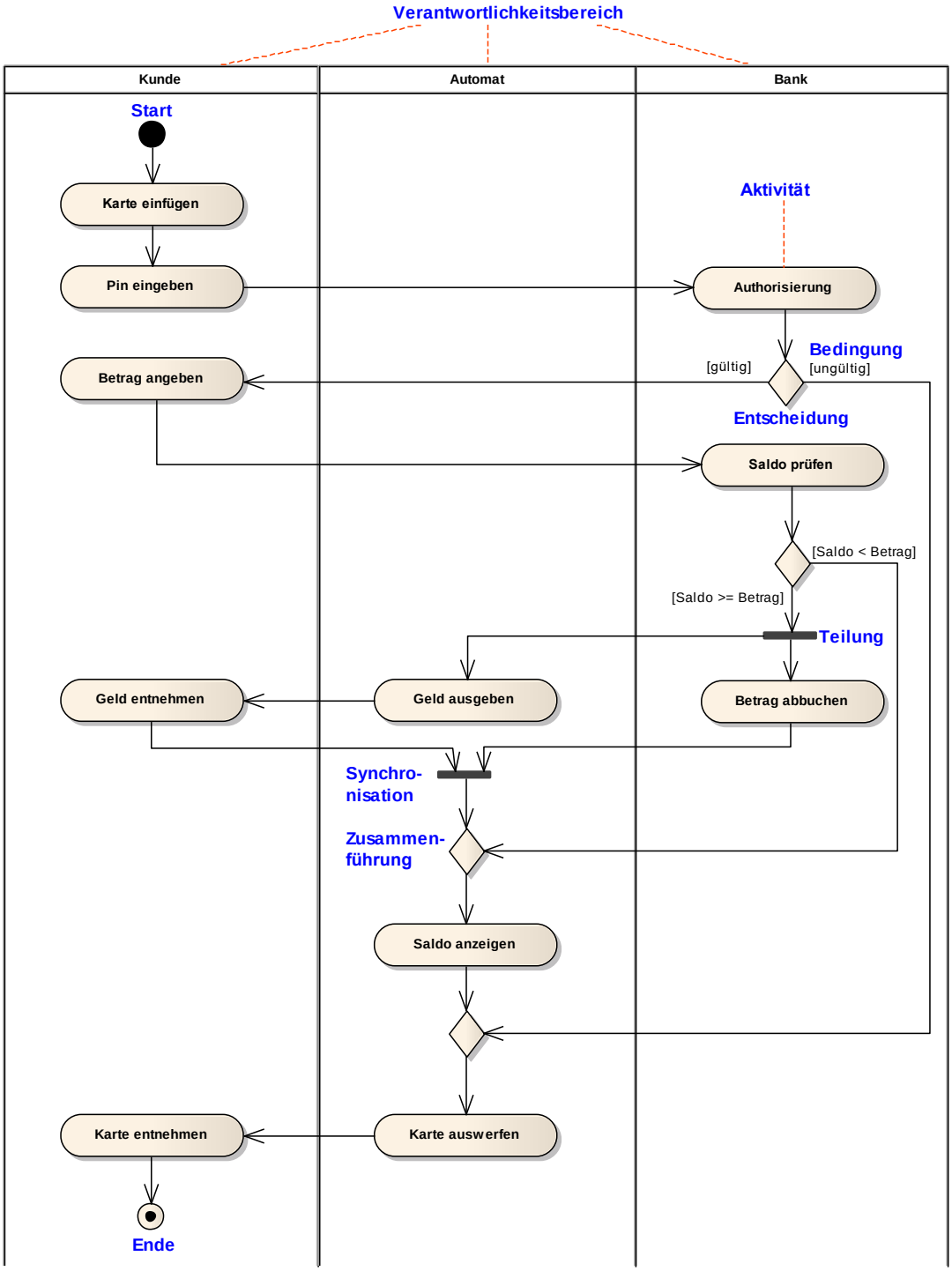


Mehrere Pins könne auch zu sogenannten **Parametergruppen** zusammengefasst werden. Innerhalb einer (eingehenden) Gruppe müssen nun alle Objekte bereitstehen, bevor die Aktivität startet. Umgekehrt müssen bei einer ausgehenden Parametergruppe alle Objekte vorhanden sein, bevor der Objektfluss weiter geht.

Eine weitere Möglichkeit bietet die sogenannten **Ausnahmeparameter** (engl. Exception). Tritt eine Ausnahme ein, so geht der Kontrollfluss unabhängig von allen anderen Kontroll- und Objektflüssen über den Ausnahmeparameter weiter.

10.7. Partitionen (Verantwortlichkeitsbereiche)

Aktivitätsdiagramme können optional in Verantwortungsbereiche unterteilt werden. Dies erfolgt durch die sogenannten „swimlanes“ welche entlang der Objektegrenzen verlaufen. Die Verbindungslinien stellen den Übergang von einer zur nächsten Entität dar.

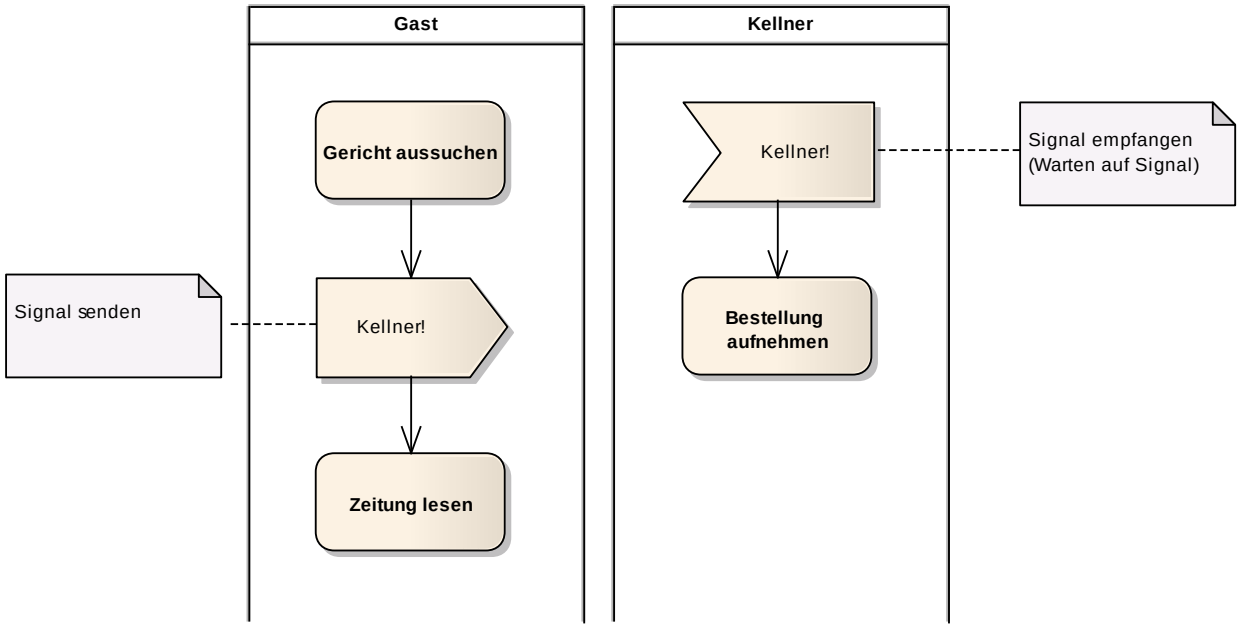


Das Beispiel zeigt die Aktivitäten für das Abheben von Geld mit Hilfe eines Automaten. Die drei involvierten Objekte sind der Kunde, der Automat und die Bank.

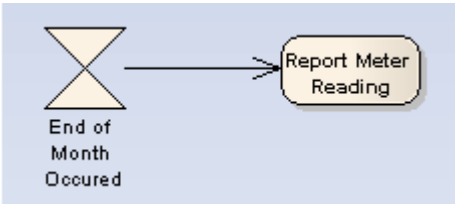
10.8. Signale

Während eines Kontrollflusses können Signale gesendet und empfangen werden. Damit ist es möglich, nebenläufige Prozesse zu synchronisieren oder auf äussere asynchrone Ereignisse zu reagieren.

Beispiele:



Es sind auch zeitliche Ereignisse möglich, die ein Signal auslösen:



Signale eignen sich ausgezeichnet für den asynchronen Datenaustausch und die Synchronisation von nebenläufigen Prozessen.

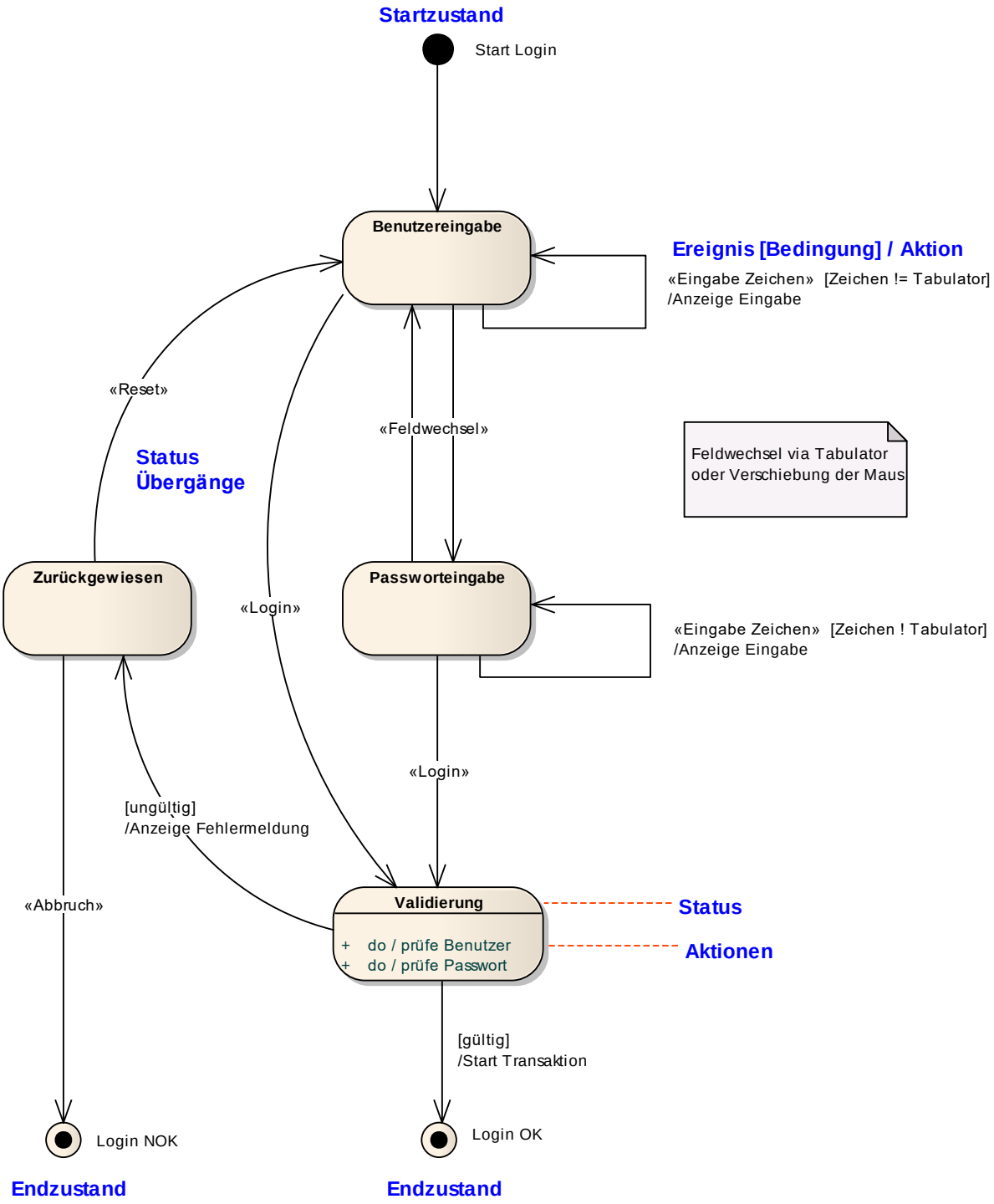
Ein Signal von mehreren Aktionen aus gesendet werden, jedoch nur eine Aktion kann das Signal empfangen.

11. Zustandsdiagramm

Mit einem Zustandsdiagramm werden die möglichen Zustände eines Objektes und die Bedingungen für einen Statusübergang aufgezeigt. Zustandsdiagramme oder Zustandsautomaten (engl. State Machine) basieren auf dem Konzept von extremistischen, endlichen Automaten.

Zustandsdiagramme eignen sich zum Beispiel für die Modellierung der Lebenszyklen von Business Objekten oder für die Darstellung des Verhaltens von Objekten auf Ereignisse.

Beispiel Login:



Das Beispiel zeigt eine Login Sequenz. Hierzu wird ein Benutzername und Passwort benötigt. Anschliessend wird durch Betätigung der Login Taste die Validierung durchgeführt.

Die Login Sequenz kann in vier verschiedene, sich nicht überlappende, Zustände unterteilt werden. Für jeden Zustand werden die Bedingungen definiert, die für einen Status Übergang notwendig sind.

Notation:

Zustände werden durch Rechtecke mit gerundeten Ecken dargestellt. Sie können in bis zu drei Bereiche geteilt werden.

- Im ersten Teil steht der Name des Zustandes

- Der mittlere Bereich enthält die Zustandsvariablen. Da sie Attribute der Klasse sind, werden sie folgendermassen notiert:
  variable : Klasse = Initialwert { Merkmal } { Zusicherung }

- Im unteren Bereich werden mögliche innere Ereignisse, Bedingungen und daraus resultierende Operationen definiert. Für sie gilt folgendes Format:
  Ereignis / Aktionsbeschreibung

  Die Aktionsbeschreibung kann ein Operationsname sein oder frei formuliert werden. Sie kann Zustandsvariablen, Attribute einer Klasse oder Parameter der eingehenden Transition enthalten.

Mit den **Verbindungen** werden die Statusübergänge markiert. Diese werden mit dem auslösenden Ereignis bezeichnet. Dazu können noch Bedingungen und Folgeaktionen angegeben werden.

Es sind auch **Übergänge** auf den gleichen Status möglich, wie im Beispiel mit der Eingabe von Benutzer und Passwort Daten dargestellt.

Der **Startpunkt** wird durch einen ausgefüllten Kreis gekennzeichnet. Der **Endzustand** enthält zusätzlich einen zweiten nicht ausgefüllten Kreis. Zudem gibt es das sogenannte **Terminator** Symbol (X). Diese beendet den gesamten Zustandsautomaten.

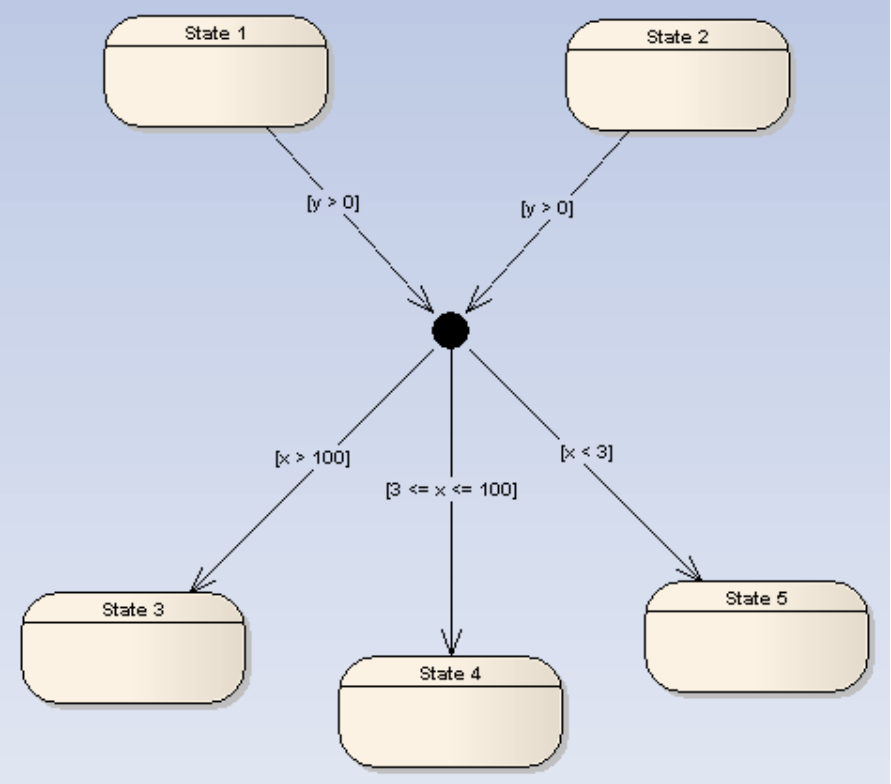
Die **Aktionen** die Ausgeführt werden aufgrund eines Ereignisses oder einer Aktion, können mit einem Schrägstrich mit anschliessender Bezeichnung der Aktion dargestellt werden.

Es gibt auch Aktionen die nicht durch ein externes Ereignis ausgelöst werden sondern durch eine Aktion „**innerhalb**“ des Status. Ein Beispiel dazu ist die Validierung, die je nach Resultat (gültig oder ungültig) der internen Prüfung einen Statuswechsel bewirkt. Für die interne Aktion kann entweder beim Eintreten in den Zustand, während dem Zustand und vor dem Verlassen ausgeführt werden. Hierzu stehen die folgenden Ereignisse zur Verfügung:

- entry → Ausführung beim Eintritt in den Zustand
- do → Ausführung während dem Zustand
- exit → Ausführung vor dem Verlassen des Zustandes

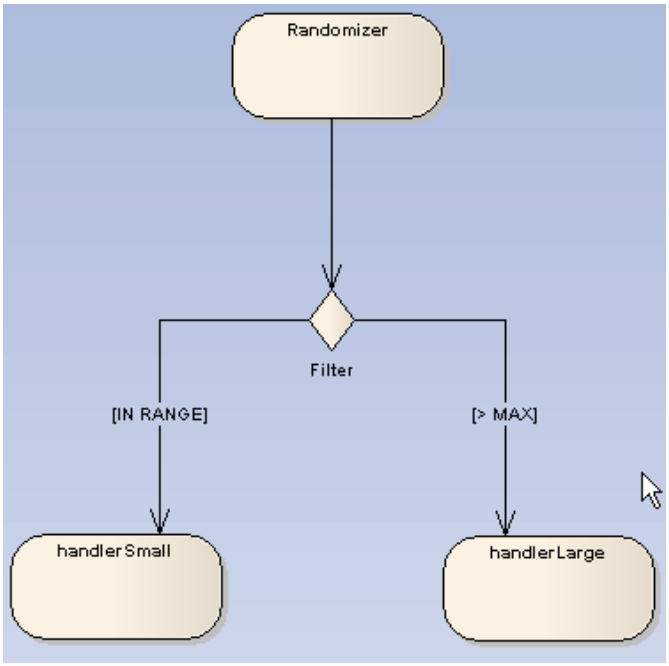
Kreuzung:

Eine Kreuzung ermöglicht es, mehrere Transitionen zur Auswahl zu stellen. Die Entscheidung wird beim Verlassen des Ausgangszustandes (statisch) getroffen.



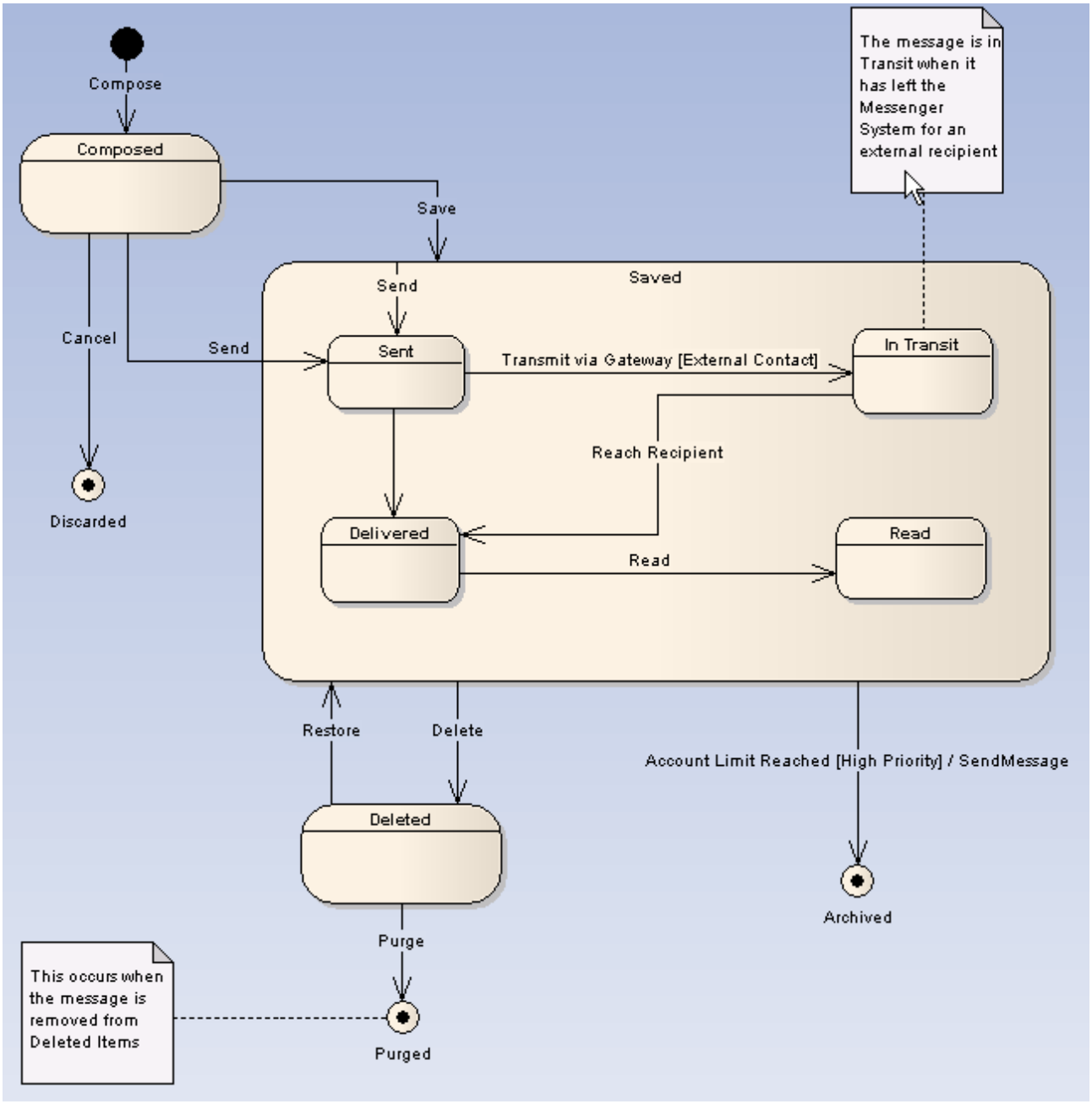
Entscheidung:

Eine Entscheidung ermöglicht es, wie bei einer Kreuzung, mehrere Transitionen zur Auswahl zu stellen. Die Entscheidung wird hier aber dynamisch an der Entscheidung gefällt.



Zusammengesetzter Zustand:

Innerhalb eines Statusdiagramms können auch ein oder mehrere zusammengesetzte Zustände (engl. Composite State) dargestellt werden. Die Status Elemente innerhalb des Composite State werden als Unterzustände (engl. Substates) bezeichnet.



Das Beispiel zeigt den zusammengesetzten Status Saved mit den Unterzuständen Sent, In Transit, Delivered und Read.

Regionen:

Mit Regionen können zusammengesetzte Zustände oder ganze Zustandsautomaten in parallele Zustandsräume aufgeteilt werden.

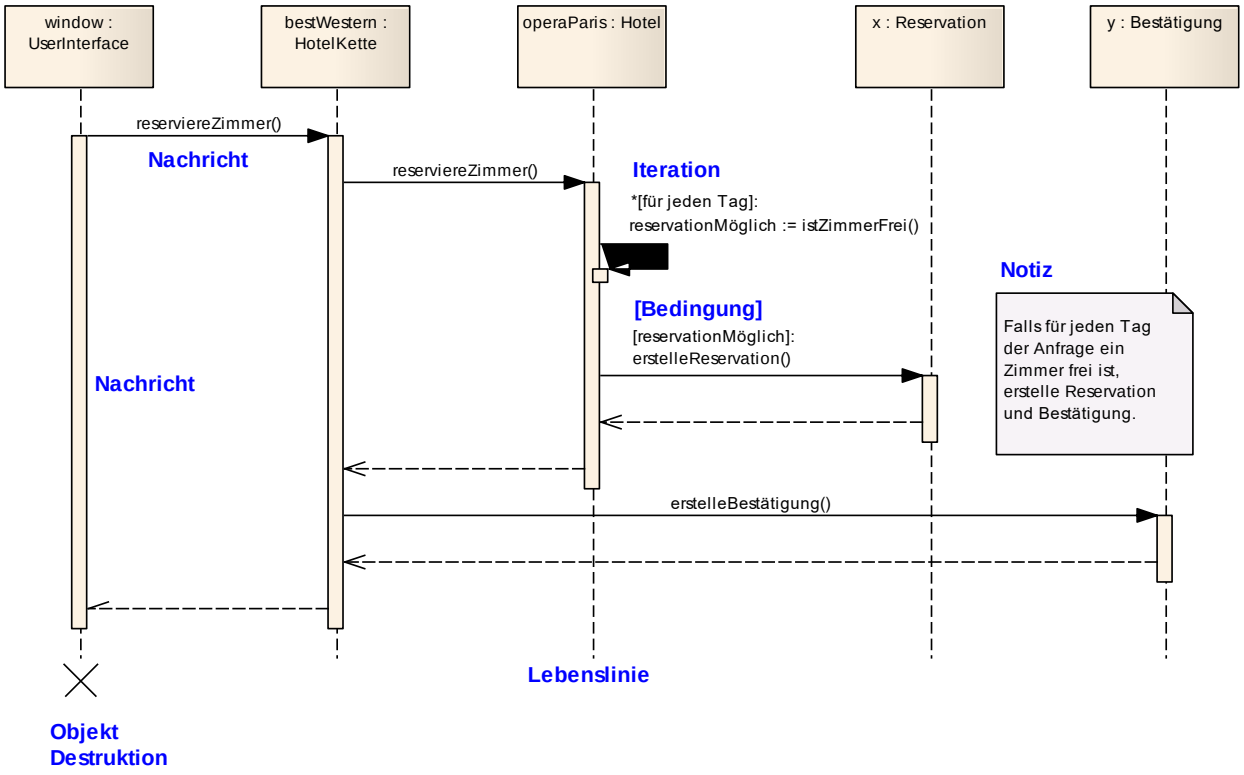
12. Sequenzdiagramm

Das Sequenzdiagramm zeigt den zeitlichen Verlauf von Nachrichten (Operationen) zwischen verschiedenen Interaktionspartnern (Objekte) für eine zeitlich begrenzte Situation.

Das Sequenzdiagramm kann die gleichen Sachverhalte wie ein Kommunikationsdiagramm zeigen, jedoch aus einer anderen Perspektive. Beim Kommunikationsdiagramm steht der Zusammenhang der Rollen im Vordergrund, Sequenzdiagramme hingegen sind auf den zeitlichen Ablauf ausgerichtet.

In der Analysephase eigenen sich Sequenzdiagramme sehr gut für die Darstellung von Nachrichtenflüssen in Geschäftsprozessen oder Abläufen von Anwendungsfällen. Beim Design können u.a. Interaktion einzelner Systemteile, Funktionsweise von Design Pattern, Abläufe von wichtigen Systemteilen oder die Interaktion von Benutzern mit dem User Interface aufgezeigt werden. Während der Implementation- / Testphase können Abläufe von Testszenarien oder einzelnen Testfällen dargestellt werden.

Beispiel Hotel Reservation:



Das Beispiel zeigt die Ablauf Sequenz einer Hotel Reservation. Startpunkt ist dabei das Reservationsfenster der Client Applikation. Diese sendet eine reserviereZimmer() Nachricht zur Best Western Hotel Kette. Falls ein Hotelzimmer für die angefragte Periode frei ist, wird eine Reservation ausgeführt und eine Bestätigung erstellt.

Dazu wird u.a. auch ein Selbstaufruf eingesetzt, mit dem in einer Schlaufe geprüft wird ob ein Zimmer für die gewünschten Tage frei ist. Die Iteration wird dabei mit dem Stern '*' Zeichen dargestellt. Die Ausdrücke in den eckigen Klammern […] stellen eine Bedingung dar.

Notation:

Die **Objekte** werden durch Rechtecke visualisiert. Von Ihnen aus gehen die senkrechten Lebenslinien, dargestellt durch gestrichelte Linien, ab. Die involvierten Objekte werden von links nach rechts aufgelistet und zwar (wenn möglich) in der Reihenfolge Ihre Teilnahme an der Ablaufsequenz.

Die **Nachrichten** erden durch waagerechte Pfeile zwischen den Objektlebenslinien beschrieben. Der Pfeil führt dabei vom Sender zum Anfang des Ausführungsbalkens des Empfängers. Der Ausführungsbalken repräsentiert dabei die Dauer der Nachrichtenverarbeitung. Auf diesen Pfeilen werden die Nachrichtennamen in der Form: nachricht(argumente) notiert. Nachrichten, die als Antworten deklariert sind erhalten die Form: antwort:=nachricht().

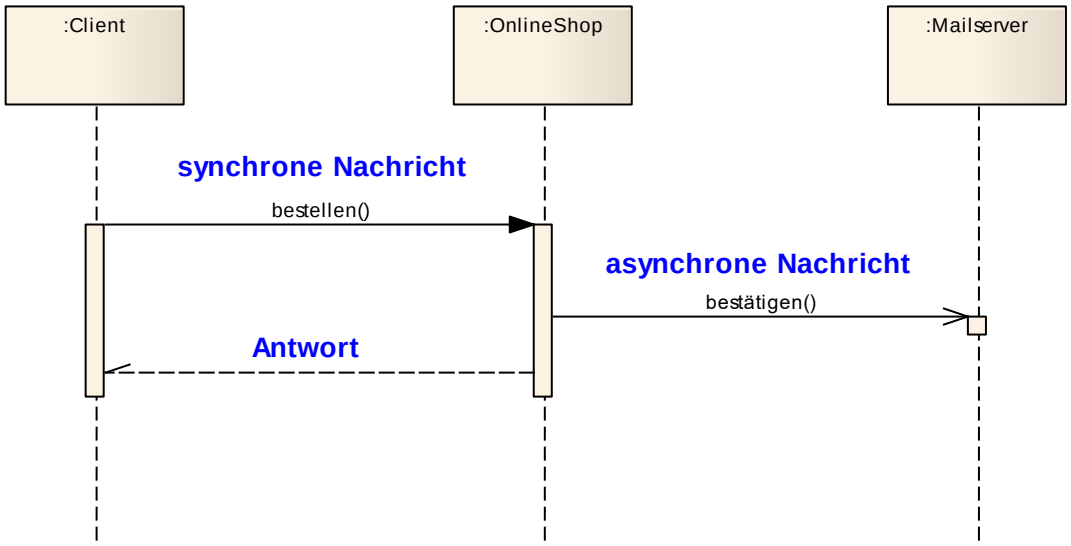
Nachrichten können **Bedingungen** der Form: [bedingung] nachricht() zugewiesen bekommen.

Iterationen von Nachrichten werden durch ein Sternchen "*" vor dem Nachrichtennamen beschrieben. Objekte, die gerade aktiv an Interaktionen beteiligt sind, werden durch einen Balken auf ihrer Lebenslinie zu kennzeichnen.

Objekte können während des zeitlichen Ablaufes des begrenzten Kontextes **erzeugt** und gelöscht werden. Ein Objekt wird erzeugt indem ein Pfeil mit der Aufschrift neu() auf ein neues Objektsymbol trifft und **zerstört** indem seine Lebenslinie in einem Kreuz endet.

Synchrone und asynchrone Nachrichten:

Nachrichten können synchron und asynchron ausgetauscht werden. Der Unterschied in der Darstellung wird durch die Pfeilspitze ausgedrückt.



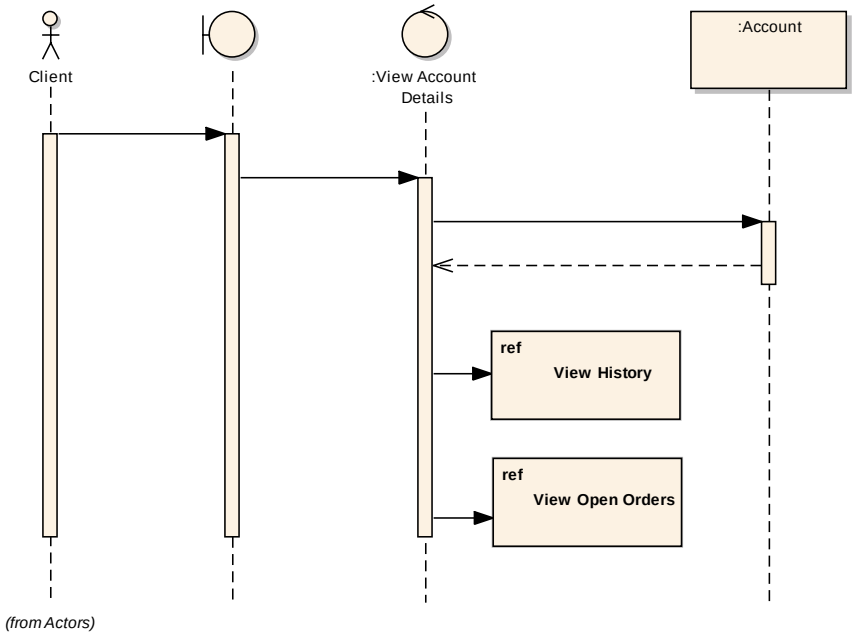
Fragmente:

Ein Sequenzdiagramm kann auch aus verschiedenen Fragmenten zusammengesetzt werden. Ein Fragment wird dabei mit einem Interaktionsrahmen umgeben und über einen Operator (sowie je nach Operator mit weiteren Angaben) beschreiben.

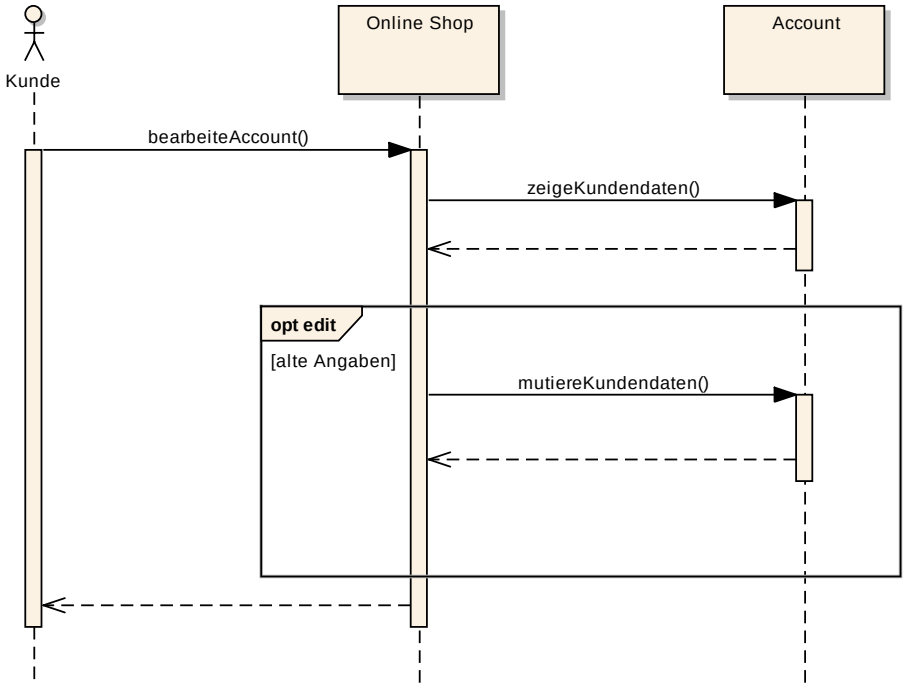
Die nachfolgende Tabelle zeigt einige gebräuchliche Operatoren:

Bezeichnung	Parameter / Bedingungen	Beschreibung
alt	[condition 1] [condition n] [else]	Auswahl von alternativen Abläufen. Es wird derjenige Teilablauf ausgeführt, für den die Bedingung zutrifft. Falls keine Bedingung zutrifft wird der [else] Zweig (falls vorhanden) ausgeführt.
loop	[condition]]	Der Block wird als Schleife wiederholt solange die angegebene Bedingung wahr ist.
break	[condition]	Trifft die Bedingung zu, so wird die Teilsequenz des Break ausgeführt und anschliessend wird die laufende Sequenz verlassen, d.h. die nachfolgenden Nachrichten werden nicht mehr ausgeführt. Trifft die Bedingung nicht zu, so wird die Teilsequenz übersprungen und mit den nachfolgenden Nachrichten weitergefahren. Break kann auch zum beenden von Schleifen verwendet werden.
opt	[condition]	Ausführung einer optionalen Teilsequenz falls die Bedingung wahr ist.
par		Parallele Ausführung von Teilsequenzen.
ref		Referenz auf ein weiteres Sequenzdiagram. Damit können Sequenzdiagramme verschachtelt werden.

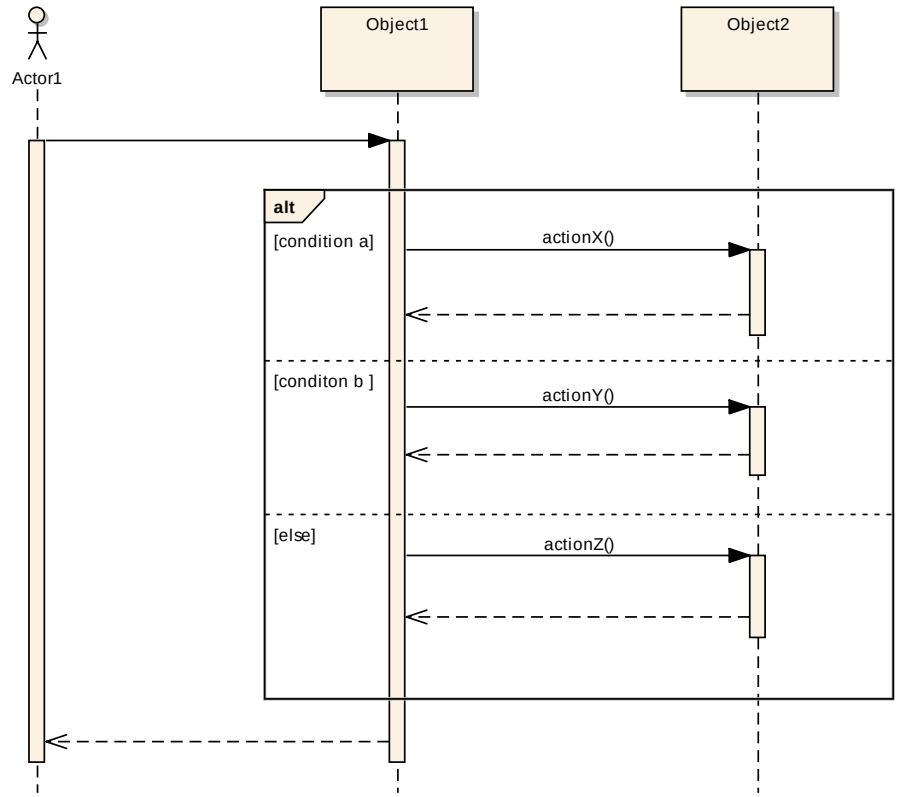
Beispiel mit Referenz:



Beispiel mit optionaler Sequenz:



Beispiel mit alternativen Sequenzen:



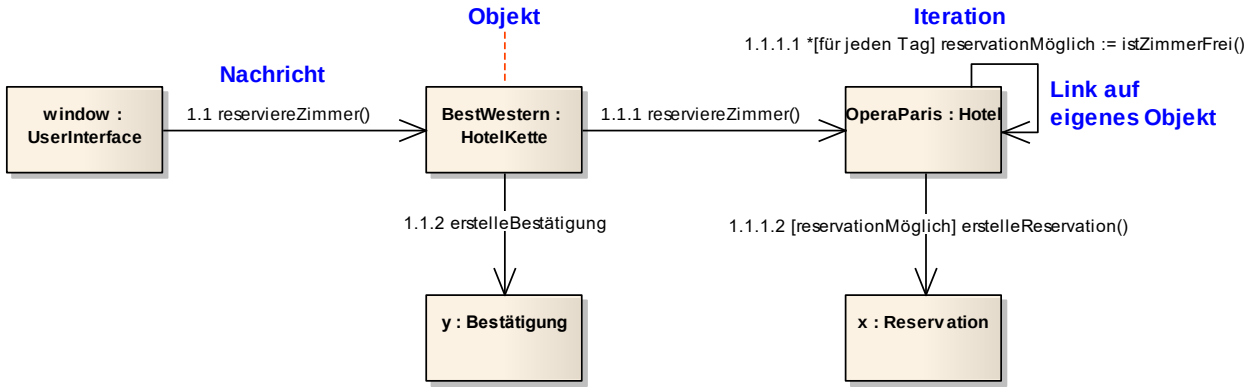
13. Kommunikationsdiagramm

Das Kommunikations- / Kollaborationsdiagramm visualisiert die einzelnen Objekte und ihre Zusammenarbeit untereinander. Dabei steht, im Vergleich zum Sequenzdiagramm, der zeitliche Ablauf dieser Interaktionen im Hintergrund, vielmehr werden die für den Programmablauf und das Verständnis des selbigen wichtigen kommunikativen Aspekte zwischen den einzelnen Objekten ereignisbezogen dargestellt.

Der zeitliche Verlauf der Interaktionen wird lediglich durch eine Nummerierung der Nachrichten symbolisiert. Die einzelnen Objekte können Nachrichten austauschen, ein Objekt kann sich jedoch auch stets selbst Nachrichten zusenden, ohne dass eine Assoziation vorhanden sein müsste.

Das Kollaborationsdiagramm kann für die Darstellung von Entwurfs-Sachverhalten benutzt werden und, in etwas detaillierter Form, von Realisierungssachverhalten. Es beinhaltet stets kontextbezogene begrenzte Projektionen des Gesamtmodells.

Beispiel Hotel Reservation:



Notation:

Die Objekte werden mit dem Objekt oder Klassennamen oder beidem (getrennt durch einen Doppelpunkt) gekennzeichnet. Diese werden durch Assoziationslinien mit einander verbunden, auf denen dann die Nachrichten (mit Name, möglichen Parametern und Antwort) angegeben werden.

Jede Nachricht im Kommunikationsdiagramm hat eine Sequenznummer, so dass die Ablaufreihenfolge ersichtlich ist. Die „Top Level“ Nachricht erhält die Nummer 1. Jede Nachricht auf der gleichen Stufe (während eines Aufrufes) hat dieselbe Vorziffer gefolgt von der Nachziffer 1, 2, etc.) entsprechend der Aufrufreihenfolge.

Beispiel:

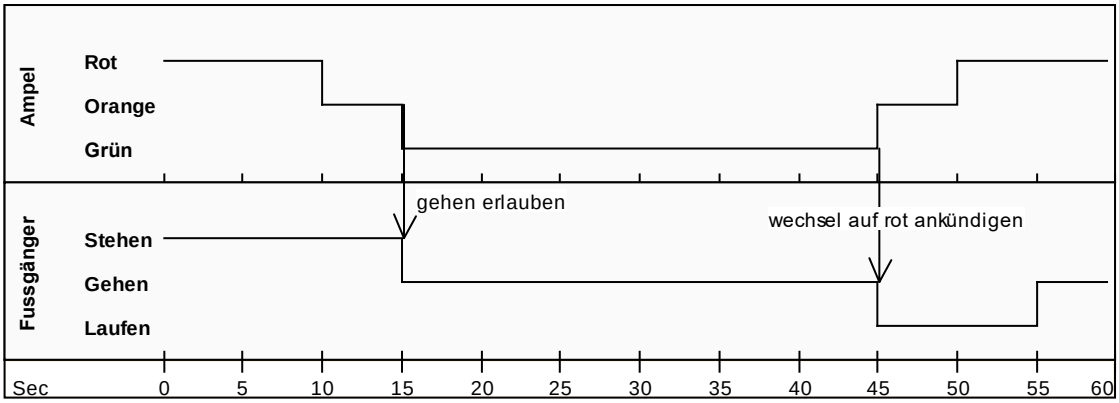
Vom der BestWestern Hotel Kette gibt es die beiden Aufrufe. 1.1.1 reserviereZimmer() und 1.1.2 erstelleBestätigung(). Die gemeinsame Vorziffer lautet in diesem Fall 1.1.

14. Zeitdiagramm

Mit einem Zeitdiagramm wird das zeitliche Zusammenspiel von mehreren beteiligten Objekten und deren Zuständen modelliert. Der Fokus liegt auf den zeitlichen Abhängigkeiten im Kontext von Zustandswechseln.

Ähnliche Diagramme sind vor allem in der Elektrotechnik und Elektronik verbreitet. Für die UML wurden diese Diagramme adaptiert, so dass der Bezug von Objekten und Objektzuständen möglich ist.

Beispiel Ampel:



Notation:

Ein Zeitdiagramm besteht aus einer horizontalen Achse auf der die Zeit linear dargestellt wird. Auf der vertikalen Achse werden nun die Objekte sowie deren Zustände dargestellt. Für jedes Objekt werden nun die Zustandsübergänge aufgezeigt. Das zeitliche Zusammenspiel wird mit Hilfe von vertikalen Linien dargestellt. Hierzu können auch noch Bedingungen angegeben werden.

Beispiel:

Das Beispiel zeigt ein Zeitdiagramm für die Interaktion einer Ampel und einem Fussgänger.

- *Das Zeitdiagramm besteht aus den beiden Lebenslinien Ampel und Fussgänger.*
- *Die Ampel hat die Zustände Rot, Orange und Grün.*
- *Der Fussgänger hat die Zustände Stehen, Gehen und Laufen.*
- *Die Zeitskala zeigt einen Bereich von 0..60 Sekunden an.*
- *Die Ampel ist zuerst 10 Sekunden rot, dann 5 Sekunden Orange und anschliessend 30 Sekunden Grün. Nach der Grünphase wechselt sie wieder für 5 Sekunden auf Orange und anschliessend auf Rot.*
- *Der Fussgänger beginnt zu gehen, sobald die Ampel auf Grün ist. Wenn Sie anschliessend auf Orange wechselt beginnt er zu laufen.*

15. Übungen

15.1. Pflegeheim (Klassendiagramm)

Übung:

Modellieren Sie folgenden Sachverhalt mittels eines Klassendiagramms. Das Klassendiagramm beinhaltet alle Klassen, Attribute und Methoden, die aus dem Text hervorgehen.

Weiter sind die Beziehungen der Klassen mit den richtigen Typen und eventuell Kardialitäten zu finden (Komposition, Aggregation, einfache Assoziation, Vererbung, Realisation).

Alle Methoden und Attribute, welche nicht aus dem Text hervorgehen sind zu vernachlässigen.
Tipp: verwenden Sie abstrakte Klassen und Interfaces wo dies Sinn macht

Aufgabe:

1) Ein Pflegeheim hat Patienten und Mitarbeiter
2) Ein Mitarbeiter kann ein Arzt oder Betreuer sein
3) Ein Patient hat Angehörige (Angehörige können auch anderen Patienten zugeordnet sein)
4) Mitarbeiter haben eine Personalnummer, Patienten werden mit der AHV Nummer identifiziert und Angehörige haben einen Namen und Vornamen
5) Das Pflegeheim bietet Services an, es wird zwischen internen und externen Services unterscheiden

    a. externe Services können Taxi Dienst und Lichttherapie sein
    b. interne Services können PKW Dienst, Kinderbetreuung und Gesprächsgruppen sein
    c. alle Services haben einen Preis der über die Methode getPreis verfügbar sein soll

6) Jeder Betreuer hat einen Einsatzplan
7) Der Einsatzplan enthält Einsätze mit Startzeit und Endzeit sowie um welchen internen Service es sich handelt (nur interne Services werden von Betreuern bestritten) und welcher Patient den Einsatz braucht
8) Zu Gesprächsgruppen kann zusätzlich ein Arzt beigezogen werden (ohne Einsatzplan)

15.2. Nationalliga (Klassen- und Objektdiagramm)

Übung:

Es soll ein Klassendiagramm für eine Fussballliga erstellt werden. Basierend darauf soll anschliessend anhand von einigen Beispieldaten ein entsprechendes Objektdiagramm erstellt werden.

Aufgabe:

1) Erstellen Sie ein Klassendiagramm für eine Fussballliga, so dass folgende Informationen abgebildet werden können:
   - Es gibt eine Liga mit 1..* Divisionen und 6..* Teams pro Division.
   - Die Liga beinhaltet als Information das Land.
   - Jede Division hat einen Namen.
   - Ein Team beinhaltet die Daten des aktuellen Tabellenstandes (Rang, Spiele, etc.) gemäss den Tabellen von Aufgabe 2.

2) Erstellen Sie ein Objektdiagramm für die Nationalliga der Schweiz, mit folgenden Angaben:

NLA – Axpo Super League

Rang	*Mannschaft*	*Spiele*	*Punkte*	*Torverhältnis*
1	Young Boys Bern	34	77	77:40
2	FC Basel	34	74	85:46
3	Grasshoppers Zürich	34	62	61:41

NLB – Challenge League

Rang	*Mannschaft*	*Spiele*	*Punkte*	*Torverhältnis*
1	Lugano	29	58	64:28
2	Thun	29	57	64:34
3	Winterthur	29	53	65:43

15.3. Store (Komponentendiagramm)

Übung:

Modellieren Sie folgenden Sachverhalt mittels eines Komponentendiagramms. Das Diagramm beinhaltet alle Elemente die aus dem Text hervorgehen. Modellieren Sie die einzelnen Komponenten sowie deren Verbindungen.

Aufgabe:

1) Wir haben eine Komponente Store mit zwei Ports.
2) Der erste Port stellt eine Schnittstelle OrderEntry zur Verfügung, für die Aufnahme von Bestellungen.
3) Der zweite Port benötigt zwei Schnittstellen. Die erste ist Account für die Abfrage von Kundendaten, die zweite ist Warehouse für die Abbuchung von Artikeln.
4) Innerhalb der Komponente Store hat die folgenden drei Komponenten: Order, Product und Customer.
5) Die Order Komponente stellt die Schnittstelle OrderEntry zur Verfügung und benötigt die beiden Schnittstellen OrdableItem und Person.
6) Die Product Komponente stellt die Schnittstelle OrdableItem zur Verfügung und benötigt die Schnittstelle Warehouse.
7) Die Customer Komponente stellt die Schnittstelle Person zur Verfügung und benötigt die Schnittstelle Account.

15.4. WebApp (Komponentendiagramm)

Übung:

Modellieren Sie folgenden Sachverhalt mittels eines Komponentendiagramms. Das Diagramm beinhaltet alle Elemente die aus dem Text hervorgehen. Modellieren Sie die einzelnen Komponenten sowie deren Verbindungen.

Aufgabe:

1) Wir möchten ein Komponentendiagramm mit einer Firewall, einem WebServer und einer Datenbank erstellen.
2) Die Firewall stellt den Aufrufenden Clients die Schnittstelle AcceptRequest zur Verfügung. Für die Kommunikation mit dem WebServer wird die Schnittstelle DoRequest benötigt und die Schnittstelle Forward Request angeboten.
3) Der WebServer stellt die Schnittstelle DoRequest zur Verfügung und benötigt die beiden Schnittstellen ForwardRequest und ProcessSQL. Diese sollen via Ports nach aussen visualisiert werden.
4) Die Datenbank stellt die Schnittstelle ProcessSQL zur Verfügung:
5) Innerhalb des Web Servers gibt es weitere Komponenten.
    a. Eine Komponente PHP Skripts. Sie ist zuständig für die Verarbeitung der Requests und das Senden der Antworten.
    b. Einen XML Konverter Komponente für die Transformation von XML Daten in HTML Seiten mit Hilfe von einem XSL Stylesheet.
6) Neben der Verarbeitung der Requests ist die PHP Komponente auch zuständig für die Datenbank aufrufe. Zudem benötigt Sie die XML Konverter Komponente.
7) Der XML Konverter stellt die Schnittstelle KonvertXML zur Verfügung und hat eine Abhängigkeit auf die Stylesheet.xsl Datei, welche im Diagramm ebenfalls gezeigt werden soll.

15.5. Reporting (Einsatz- / Verteildiagramm)

Übung:

Modellieren Sie folgenden Sachverhalt mittels eines Einsatz-/Verteildiagramm. Das Diagramm beinhaltet alle Elemente die aus dem Text hervorgehen. Modellieren Sie die einzelnen Komponenten sowie deren Verbindungen.

Für das Reporting von Verkaufszahlen wird eine mehrschichtige Web Applikation erstellt bestehend aus einem Server für die Datenbank, einem Server für die Business Logik, einem Statistik Server sowie beliebigen Client Rechnern.

Auf dem Datenbank Server ist eine DB 2 Laufzeitumgebung installiert. Der Name des Schemas für die Applikation lautet „reporting“.

Auf dem Statistik Server hat es eine Komponente „Sales Figures Service“, welche die Kennzahlen für unsere Reporting Lösung bereitstellt.

Auf dem Business Logik Server ist eine Laufzeitumgebung für einen Application Server installiert. Auf diesem laufen die beiden Komponenten „Reporting“ und „Persistence“.

Als Beispiel für einen Client Rechner soll die Instanz „x“ im Diagramm eingezeichnet werden. Die Bedienung der Reporting Lösung erfolgt über eine Browser.

Die Konten verwenden die folgenden Protokolle um miteinander zu kommunizieren

- HTTP zwischen Client und Business Logik Server
- Native DB2 zwischen Business Logik Server und DB Server
- SOAP over HTTP zwischen Business Logik Server und Statistik Server

Aufgabe:

1) Erstellen Sie ein Einsatz- / Verteildiagramm mit den beschriebenen Konten.
2) Zeichnen Sie die Verbindungen zwischen den Konten ein und geben Sie die verwendeten Protokolle an.
3) Geben Sie die logischen Abhängigkeiten der Komponenten untereinander an.

15.6. Coiffeur Salon (Anwendungsfall)

Übung:

Erstellen Sie ein Anwendungsfalldiagramm für die folgenden Geschäftsprozesse eines Coiffeur Salons.

Geschäftsprozesse:

- Anmelden eines Kunden für einen Salonbesuch.
- Kassieren von Verkäufen bei Laufkundschaft.
- Nachbestellung von Artikeln.
- Kontrollieren der Arbeitszeiten für Mitarbeiter
- Berechnung der Löhne für Mitarbeiter.
- Bedienung von Kunden und Kassieren der Dienstleistung.

Aufgabe:

Erstellen Sie ein Anwendungsfalldiagramm für die beschriebenen Geschäftsprozesse.
mit folgenden Bedingungen:

1) Die Kunden werden vom Stylisten bedient und bezahlen die erbrachten Dienstleistungen auch gleich beim Stylisten.
2) Der Rezeptionist nimmt Kundenanmeldungen entgegen und erledigt die Verkäufe an die Laufkundschaft.
3) Der Salon Verwalter erledigt die Bestellungen und die Administration der Mitarbeiter.
4) Sowohl bei der Bedienung der Kunden als auch beim Verkauf von Artikeln an die Laufkundschaft wird der interne Anwendungsfall „Einkassieren“ verwendet.

15.7. Patentverwaltung (Systemkontext)

Übung:

Erstellung eines Systemkontext Diagramm für eine Patentverwaltung mit folgenden Sachverhalt:

- Die Patenverwaltung hat eine Datenbank, eine Schicht mit Geschäftsdiensten, einen Web Client für den Zugriff der Kunden (Erfinder) sowie einen Rich Client für die internen Prüfer und Administratoren.
- Beide Clients haben nur Zugriff auf die Geschäftsservice.
- Im Weiteren gibt es eine Registrierungsservice der WIPO (Word Intellectual Property Organisation) welcher zwecks Datenaustausch mit dem Patenverwaltungssystem kommuniziert (und zwar mit der Geschäftsservice Komponente).
- Für den Versand der Rechnungen benutzt die Patentverwaltung zudem eine SAP System. Dieses wird ebenfalls von den Geschäftsservice Komponente angesteuert.

Aufgabe:

1) Erstellen Sie ein Systemkontext Diagramm für die beschriebene Patentverwaltung.
2) Zeichen sie auch die Assoziationen der internen Komponenten (und zwar als Abhängigkeiten) ein.

15.8. Reisebüro (Aktivitätsdiagramm)

Übung:

Mit einem Aktivitätsdiagramm soll der folgende Ablauf in einem Reisebüro dargestellt werden:

- Ein Kunde äussert einen Reisewunsch.
- Der Reiseberater klärt die Rahmenbedingungen und erstellt anschliessend eine Offerte.
- Der Kunde prüft die Offerte. Falls er nicht einverstanden ist, äussert er nochmals seinen Reisewunsch.
- Falls der Kunde einverstanden ist, erfasst der Reiseberater die Kundendaten und überprüft ob das Angebot ausgebucht ist.
- Falls das Angebot bereits ausgebucht ist, schreibt er eine neue Offerte für den Kunden.
- Sonst erstellt der Reiseberater die Reiseunterlagen. Parallel dazu reserviert er das Angebot.
- Sobald die Unterlagen fertig sind und die Reise gebucht werden konnte, bezahlt der Kunde die Reise.
- Nach der Bezahlung durch den Kunden wird die Reise vom Reiseberater bestätigt.

Aufgabe:

1) Erstellen Sie ein Aktivitätsdiagramm mit dem beschriebenen Sachverhalt. Zeichnen Sie dabei auch die Verantwortlichkeitsbereiche vom Kunden und Reiseberater ein.

15.9. Bestellvorgang (Aktivitätsdiagramm)

Übung:

Mit einem Aktivitätsdiagramm soll der folgende Bestellvorgang in einem WebShop dargestellt werden:

- Der Kunde bestellt ein Produkt.
- Anschliessend wartet der Kunde auf die Bestätigung des WebShop Systems. Parallel dazu führt der WebShop folgenden Schritte (sequentiell) aus: Verfügbarkeit prüfen, Kreditkarte belasten und Bestätigung senden.
- Nach dem Senden der Bestätigung wartet der WebShop auf die Liefermeldung des Lagers. In dieser Zeit wird im Lager das Produkt bereitgestellt, das Produkt ausgeliefert und eine Lieferbestätigung zurückgemeldet.
- Nach erfolgter Rückmeldung wird der Bestellvorgang (resp. die Bestellung) vom WebShop abgeschlossen und der Ablauf beendet.

Aufgabe:

1) Erstellen Sie ein Aktivitätsdiagramm mit dem beschriebenen Sachverhalt.

   Zeichnen Sie dabei auch die Verantwortlichkeitsbereiche von Kunden, WebShop und Lager ein.

   Starten Sie parallele Aktivitäten mit einer Verzweigung und führen Sie diese auch wieder korrekt mit einer Synchronisation zusammen.

15.10. Thread (Zustandsdiagramm)

Übung:

Es sollen die verschiedenen Zustände eines Thread aus Sicht des Betriebssystems aufzeigt werden.

Nach dem Start befindet sich der Thread im Zustand Runnable. Sobald er vom Scheduler aktiviert wird wechselt der Zustand auf Running.

Ein Thread im Zustand Running kann ein blocking Event erhalten, was einen Wechsel in den Zustand Blocked bewirkt. Von dort kommt der Thread mit einem unblocked wieder in den Zustand Runnable.

Wird ein laufender Prozess suspendiert, kommt er wieder in den Zustand Runnable, wird er beendet, ist der Ablauf fertig.

Aufgabe:

1) Zeichen sie ein Zustandsdiagramm mit dem beschriebenen Sachverhalt.

15.11. Kreditvergabe (Zustandsdiagramm)

Übung:

Erhält eine Bank von einem Kunden einen Antrag für einen Kredit, wird eine Prüfung durchgeführt und entschieden, ob der Kredit gewährt werden soll oder nicht. Die möglichen Stati der Prüfung sollen nun mit einem Zustandsdiagramm visualisiert werden.

Folgende Zustände sind möglich / werden dabei durchlaufen:

Als erstes erhält der Kreditantrag den Status Gesuch. Während diesem Status wird ein Auszug aus dem Betreibungsregister angefordert und geprüft.

Sind Betreibungen vorhanden, so wechselt der Status auf Zurückgewiesen. Sind keine Betreibungen vorhanden, wechselt der Status auf Vorgeprüft.

Im Status Zurückgewiesen wird dem Kunden eine Absage gesendet. Anschliessend ist der Ablauf zu Ende.

Im Staus Vorgeprüft wird das Einkommen überprüft. Ist diese (im Verhältnis zum gewünschten Kredit) in Ordnung wechselt der Status auf Geprüft. Ist das Verhältnis von Einkommen und Kreditbetrag ungenügend, wechselt der Status auf Zurückgewiesen.

Im Status Geprüft, werden nacheinander die folgenden Aktionen ausgeführt: Vertrag erstellen, Warten auf Kundenunterschrift, Vergabe des Kredites. Wurden alle Aktionen ausgeführt, wechselt der Status auf Vergeben und der Ablauf ist zu Ende.

Aufgabe:

1) Erstellen Sie ein Zustandsdiagramm mit dem beschriebenen Verhalten.

15.12. CD Sales Report (Sequenzdiagramm)

Übung:

Es soll ein Sequenzdiagramm erstellt werden, das aus folgenden Komponenten besteht:

- aServlet von Typ Servlet
- Eine Instanz gen vom Typ ReportGenerator
- Ein Element vom Typ CDSalesReport
- Eine Instanz aCDReport vom Typ CDSalesReport

Als erstes sendet das Servlet die Nachricht generateReport (mit dem Parameter cdId) an den Generator.

Dieser erstellt mit einer Factory Methode createReport eine Instanz aCDReport. Die Factory Methode wird von Element CDSalesReport zur Verfügung gestellt.

Anschliessend sendet der Generator nacheinander folgende Nachrichten an die Instanz aCDReport:

- setCreatedOn mit den Parameter date
- setWeekNumber mit dem Parameter week
- setSales4Week mit dem Parameter sales

Am Schluss liefert der Generator den Report an das aufrufende Servlet zurück.

Aufgabe:

1) Modellieren Sie ein Sequenzdiagramm mit dem beschriebenen Ablauf.

15.13. Mikrowelle (Sequenzdiagramm mit Nebenläufigkeit)

Übung:

Wir wollen ein Sequenzdiagramm mit einem parallelen Ablauf modellieren.

Dazu haben die beiden Elemente Person und Mikrowelle.

- Die Person (Max) sendet dabei die Nachricht kocheEssen zur Mikrowelle und erhält als Antwort das fertige Essen.
- Die Zubereitung es Essen in der Mikrowelle besteht aus den beiden parallelen Tasks heizeEssen und rotiereEssen welche als Selbstaufrufe modelliert werden.

Aufgabe:

1) Schauen Sie sich im Kapitel mit den Sequenzdiagrammen die Notation für nebenläufige Teilsequenzen an.
2) Modellieren Sie ein Sequenzdiagramm mit dem beschriebenen Sachverhalt.

15.14. Zahlungsauftrag (Sequenzdiagramm mit alternativem Ablauf und Abbruch)

Übung:

Als nächstes wollen wir je ein Sequenzdiagramm mit einem alternativen Ablauf und einem Abbruch erstellen. Dazu schauen wir uns die Abwicklung eines Zahlungsauftrages an.

Folgende Elemente sind involviert: Ein Kunde, die Bank, eine Mietzinszahlung (von Typ Einzahlungsschein) und ein Konto. Der Ablauf sieht folgendermassen aus:

- Der Kunde sendet einen Zahlungsauftrag mit einem Einzahlungsschein für eine Miete an die Bank.
- Die Bank fragt als erstes beim Einzahlungsschein den Betrag und anschliessend beim Konto den aktuellen Kontostand ab.
- Falls der Betrag grösser oder gleich ist wie der aktuelle Kontostand werden folgende Schritte ausgeführt:
    - Die Bank führt beim Konto eine Kontobelastung aus, mit der Nummer des Einzahlungsscheines und dem Betrag als Parameter.
    - Die Bank speichert ein Foto des Einzahlungsscheines beim Konto
- Falls der Betrag kleiner ist als der aktuelle Kontostand, führt die Bank folgende Schritte aus:
    - Auf dem Konto wird eine nicht ausgeführte Zahlung, mit der Nummer des Einzahlungsscheines als Parameter, vermerkt.
    - Der Einzahlungsschein wird zurückgewiesen und an den Kunden zurückgesendet.

Aufgabe 1:

1) Schauen Sie sich im Kapitel mit den Sequenzdiagrammen die Notation für einen alternativen Ablauf an.
2) Zeichen Sie ein Sequenzdiagramm mit den beschriebenen Ablauf für die ausgeführte Zahlung und dem alternativen Ablauf für die Zurückweisung.

Aufgabe 2:

3) Schauen Sie sich im Kapitel mit den Sequenzdiagrammen die Notation für einen Ablauf mit einem Abbruch (break) an.
4) Modellieren Sie den gleichen Sachverhalt wie bei der ersten Aufgabe. Verwenden Sie jetzt aber anstelle eines alternativen Ablaufs ein break.

15.15. CD Sales Report (Kommunikationsdiagramm)

Übung:

Erstellung eines Kommunikationsdiagramms anhand einer Ablaufsequenz.

Aufgabe:

1) Zeichnen Sie den Ablauf des „CD Sales Report“ Sequenzdiagramms als Kommunikationsdiagramm auf.

16. Literatur und Weblinks

[1] **Analyse und Design mit UML 2.3**
Bernd Oestereich, 2009 Oldenburg, ISBN 978-3-486-58855-2

[2] **Object Management Group (OMG)**
http://www.omg.org/uml

[3] **Sparx System**
http://www.sparxsystems.com

[4] **Wikipedia**
http://en.wikipedia.org

[5] **T. Horn, Ingenieurbüro für Softwareentwicklung**
http://www.torsten-horn.de

[6] **Head First Object-Oriented Analysis and Design**
Brett McLaughlin, Gary Pollice, David West, David Wood, Bert Bates, Kathy Sierra
2006 O'Reilly, ISBN 0-596-00867-8

[7] **OOSE GmbH**
http://www.oose.de

[8] **Requirements-Engineering und -Management**
Chris Rupp, 2009 Hanser, ISBN 3-446-41841-5

[9] **J2EE Architect's Handbook**
Derek C. Ashmore, 2004 DVT Press, ISBN 0-972-95489-9

[10] **Test Driven Development**
Kent Beck, 2002 Addison Wesley, ISBN 0-321-14653-0

[11] **Design Pattern**
Gamma / Helm / Johnson / Vlissides, 1995 Prentice Hall, ISBN 0-201-63361-2

[12] **SOA Blueprint**
http://www.soablueprint.com/

[13] **OASIS**
http://www.oasis-open.org

[14] **The Open Group**
http://www.theopengroup.org

[15] **IBM Developer Works**
http://www.ibm.com/developerworks

[16] **The Prinziples of OOD**
Robert C. Martin
http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

[17] **Clean Code: A Handbook of Agile Software Craftsmanship**
Robert C. Martin, Michael C. Feathers, Timothy R. Ottinger,
2009 Prentice Hall, ISBN 0-13-235088-2