

2 UML - Vertiefung

- 2.1 Einführung
 - 2.1.1 Entwicklungsgeschichte
 - 2.1.2 Ziele bei der Entwicklung von UML
 - 2.1.3 UML Diagramme und Zuordnung zu OOSE-Prozess
- 2.2 Fortgeschrittene Techniken für Klassendiagramme
 - 2.2.1 Stereotyp
 - 2.2.2 Aggregation und Komposition
 - 2.2.3 Abgeleitete Assoziationen und Attribute
 - 2.2.4 Schnittstellen und Abstrakte Klassen
 - 2.2.5 Assoziationsklassen
- 2.3 Modellierung des begleitenden Beispiels mit UML
- 2.4 Einige Diagramme für dynamisches Verhalten
 - 2.4.1 Sequenzdiagramm
 - 2.4.2 Kommunikationsdiagramm
 - 2.4.3 Zustandsdiagramm
 - 2.4.4 Aktivitätsdiagramm
- 2.5 Literatur und Web-Ressourcen

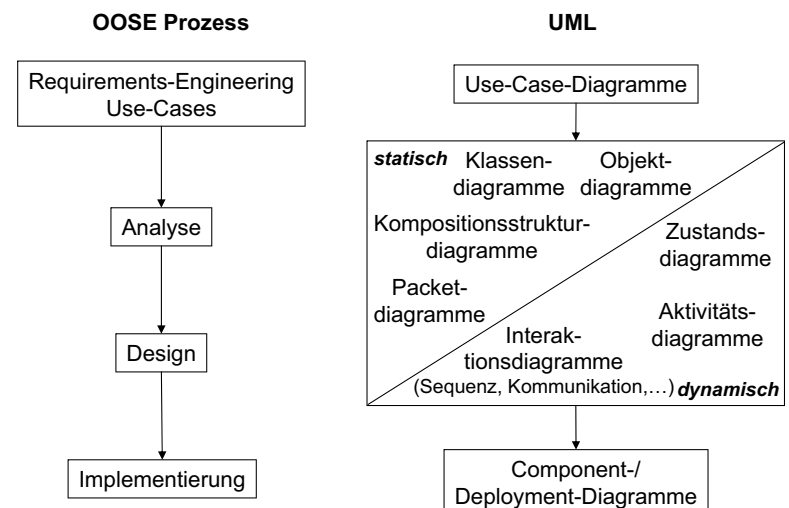
2.1.2 Ziele bei der Entwicklung von UML

- Modellierung von Systemen (nicht nur Software) unter Verwendung objektorientierter Konzepte
- Herstellen einer Verbindung zwischen konzeptuellen und ausführbaren Elementen
- Berücksichtigung der Größenprobleme von komplexen, geschäftskritischen Systemen
- Eine Modellierungssprache, die sowohl für Menschen als auch für Maschinen verwendbar ist

2.1.1 Entwicklungsgeschichte

- 1995 erste Vorversionen von UML
- 1996 erkennen viele Firmen im IT-Bereich die Bedeutung einer einheitlichen objektorientierten Modellierungssprache und schließen sich dem Vorhaben der Firma Rational an.
- Einreichung der Sprachdefinition bei der Object Management Group (OMG) zur Standardisierung
- 1997 von der OMG als Standard akzeptiert
- Oktober 2004 UML 2.0





2.1.3 UML Diagramme und Zuordnung zum OOSE-Prozess



2.2 Fortgeschrittene Techniken für Klassendiagramme

- 2.2.1 Stereotyp
- 2.2.2 Aggregation und Komposition
- 2.2.3 Abgeleitete Assoziationen und Attribute
- 2.2.4 Schnittstellen und Abstrakte Klassen
- 2.2.5 Assoziationsklassen

2.2.1 Einige vordefinierte Stereotypen

Element	textuelle Darstellung	Symbolische Darstellung
Entity Klasse	«entity»	
Interface Klasse	«interface»	
Control Klasse	«control»	
Actor Klasse	«actor»	

2.2.1 Stereotyp

- Ein Stereotyp ist ein Modellierungselement, welches zur Klassifikation bestehender Elemente (z.B. von Klassen, Assoziationen etc.) dient:
 - Gibt Hinweis auf
 - die Art der Verwendung
 - den Bezug zur Anwendungsarchitektur
- Durch die Benutzung von Stereotypen sind Elemente in einem Modell besser zu verstehen.

2.2.1 Stereotyp - Darstellung

- Entweder entsprechender **Text** in Guillemets (französische Anführungszeichen, « »)



- oder entsprechendes **Symbol** wird über dem Namen des zu klassifizierenden Elementes positioniert



2.2.2 Aggregation

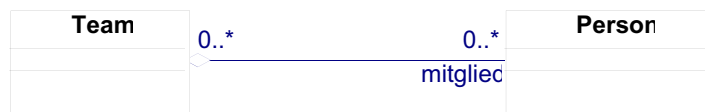
- Eine Aggregation ist
 - eine spezielle Assoziation
 - sie drückt eine Beziehung zwischen einem Ganzen und seinen Teilen aus. Z.B.
 - A besteht aus B
 - A enthält B
 - B ist Teil von A...

2.2.2 Komposition

- Komposition ist stärkere Aggregation bei der
 - Die Lebensdauer von Objekten vom Typ B ist durch die Lebensdauer der A-Objekte, in denen sie enthalten sind, beschränkt.
 - Ein Objekt vom Typ B kann nicht gleichzeitig in anderen Objekten als seinem „Vaterobjekt“ vom Typ A enthalten sein.
 - D.h. die Kardinalität der Klasse A ist in dieser Beziehung 0..1 oder 1..1.

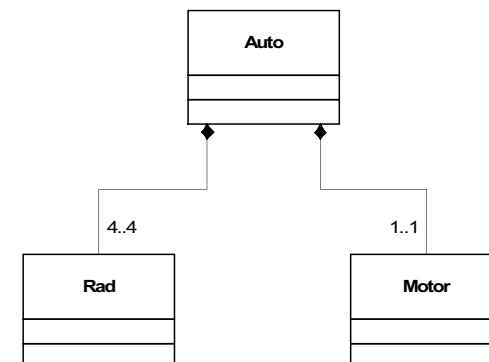
2.2.2 Aggregation - Darstellung

- Durch einen **Pfeil vom Teil zum Ganzen** mit **ungefüllter Raute** als Pfeilspitze in Richtung des Aggregats
 - Beschriftung des Pfeils und Kardinalitätsangabe ist wie bei Assoziationen



2.2.2 Komposition - Darstellung

- Durch einen **Pfeil vom Teil zum Ganzen** mit **gefüllter Raute** als Pfeilspitze in Richtung des Aggregats
 - Beschriftung des Pfeils und Kardinalitätsangabe ist wie bei Assoziationen.



2.2.3 Abgeleitete Assoziationen und Attribute

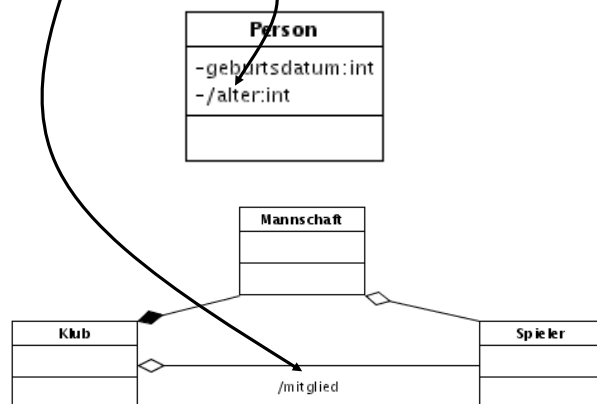
- Assoziationen und Attribute, die aus anderen Assoziationen und Attributen berechnet werden können
- Beispiel:** **Alter** von einer Person kann **berechnet**, wenn das **Geburtsdatum** der Person bekannt ist.
 - Speichern von sowohl Alter als auch Geburtsdatum als unabhängige Attribute in der Klasse Person wäre **redundant** und mit **zusätzlichem Synchronisierungsaufwand** verbunden.

2.2.4 Schnittstellen und Abstrakte Klassen

- Eine **Schnittstelle** ist eine Klasse ohne Implementierung, d.h., eine Schnittstelle hat
 - keine Attribute und
 - keinen Methodenrumpf.
- Schnittstellen bieten dem Benutzer eine von Implementierung unabhängige Sicht.
- Eine **abstrakte Klasse** ist eine Klasse, die mindestens eine abstrakte (nicht implementierte) Methode hat.
 - Eine abstrakte Klasse kann nicht instanziiert werden.

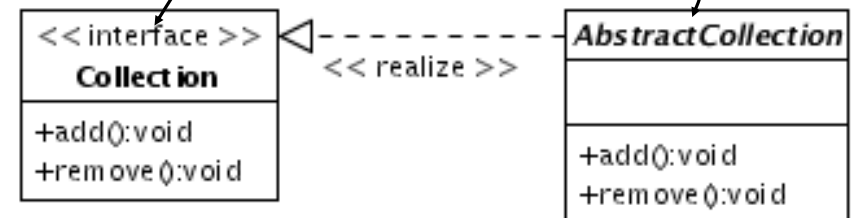
2.2.3 Abgeleitete Assoziationen und Attribute - Darstellung

- Ein **abgeleitetes Attribut** bzw. eine **abgeleitete Assoziation** wird durch ein „/“ vor der Beschriftung gekennzeichnet.



2.2.4 Schnittstellen und Abstrakte Klassen - Darstellung

- Schnittstellen und abstrakte Klassen werden wie Klassen dargestellt, außer
 - Verwendung vom Stereotyp „interface“ für Schnittstellen und kursiver Schrift bei abstrakten Klassen.

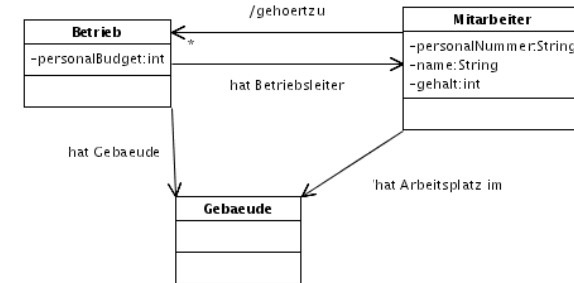


2.2.5 Assoziationsklassen

- Wenn eine Assoziation selbst Attribute oder Operationen haben soll, dann kann sie in Form einer Klasse dargestellt werden.
- Die Instanzen der Assoziationsklasse sind die konkreten Objektverbindungen zwischen den an der Assoziation beteiligten Klassen.
- Zwei beteiligte Objekte dürfen maximal nur eine Beziehung zueinander haben.

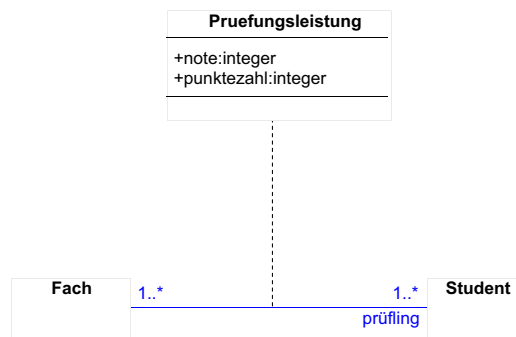
2.3 Modellierung des begleitenden Beispiels mit UML

- Das Unternehmen ist hierarchisch gegliedert. Es besteht aus verschiedenen Unternehmensbereichen (Elektro, KFZ, ...), die auf Betriebe verteilt sind, die sich an verschiedenen Standorten befinden. An einem Standort befindet sich nur jeweils ein Betrieb. Zu jedem Betrieb soll der Betriebsleiter und das Personalbudget, sowie die zum Betrieb gehörenden Gebäude festgehalten werden.



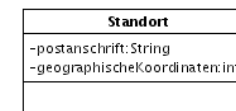
2.2.5 Assoziationsklassen - Darstellung

- Klassensymbol mit **gestrichelter Verbindung zur Assoziationslinie**.
- **Name der Assoziation** steht im **Klassensymbol**



2.3 Modellierung des begleitenden Beispiels mit UML

- Zu jedem Standort sollten sowohl die Postanschrift, als auch die geografischen Koordinaten gespeichert werden. Letztere dienen der Tourenplanung mittels eines GPS (Global Positioning System).

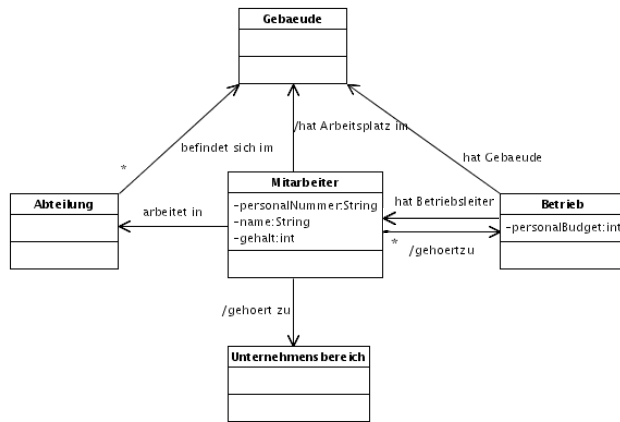


- Die Abteilungen des Unternehmens sind jeweils geschlossen in einem Gebäude untergebracht, um kurze Wege zu gewährleisten.

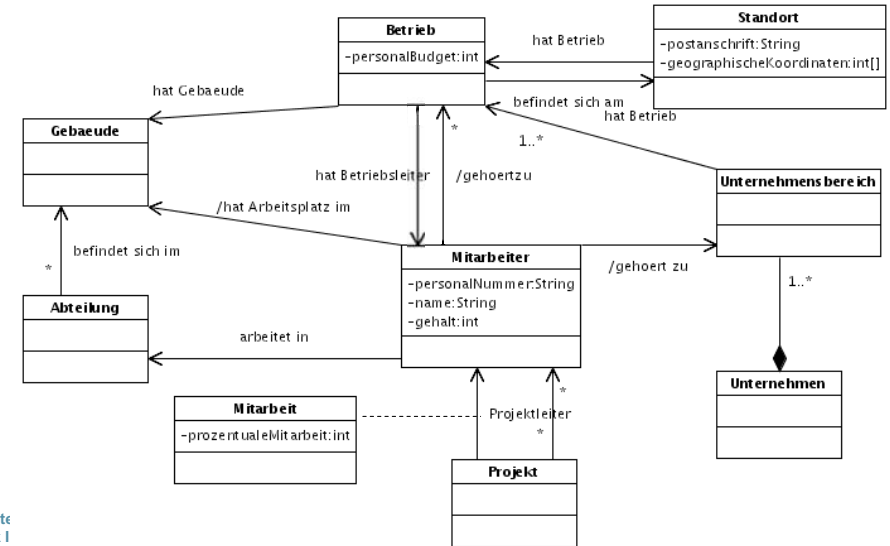


2.3 Modellierung des begleitenden Beispiels mit UML

- Zu den zur Verwaltung der Mitarbeiter notwendigen Informationen gehören neben Personal-Nummer, Name und Gehalt auch die Zugehörigkeit zu Unternehmensbereich, Betrieb und Abteilung, sowie das Gebäude in dem sich der Arbeitsplatz befindet.

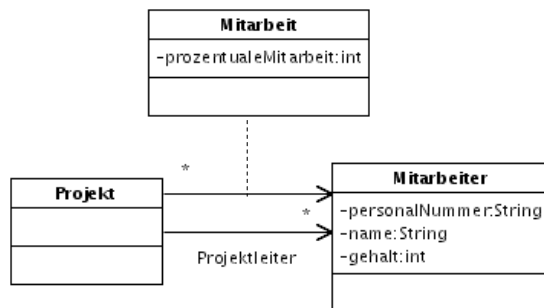


2.3 Modellierung des begleitenden Beispiels mit UML



2.3 Modellierung des begleitenden Beispiels mit UML

- Die Aufgaben der Firma sind stark projektbezogen. Deshalb sind für eine Projektverwaltung alle Projekte, Projektleiter und die zugehörigen Mitarbeiter zu speichern. Da manche Mitarbeiter parallel an verschiedenen Projekten arbeiten, ist auch der prozentuale Anteil der Arbeitszeit, mit dem an einem Projekt gearbeitet wird, von Bedeutung.



2.4 Einige Diagramme für Dynamisches Verhalten

- 2.4.1 Sequenzdiagramme
- 2.4.2 Kommunikationsdiagramme
- 2.4.3 Zustandsdiagramme
- 2.4.4 Aktivitätsdiagramme

2.4.1 Sequenzdiagramme

- Sequenzdiagramme zeigen die Interaktion zwischen Objekten (d.h. die zeitliche Abfolge von Methodenaufrufen).
- In der Designphase eines OOSE Prozesses wird für jeden Use-Case ein Sequenzdiagramm erstellt.

2.4.1 Sequenzdiagramme - Darstellung

- Ein Pfeil mit **durchgezogener Linie** stellt einen **Methodenaufruf** dar.
 - Pfeilrichtung zeigt **vom aufrufenden Objekt** zum Objekt, dessen Methode aufgerufen wird.
 - Bei einem **synchronen** Aufruf ist die Pfeilspitze ein **gefülltes Dreieck** \longrightarrow
 - Bei einem **asynchronen** Aufruf ist die Pfeilspitze **nicht geschlossen** \longrightarrow
- Ein Pfeil mit **gestrichelter Linie** stellt die **Terminierung** der aufgerufenen Methode dar.
 - Pfeilrichtung zeigt vom Objekt, dessen Methode aufgerufen wurde **zum aufrufenden Objekt**.

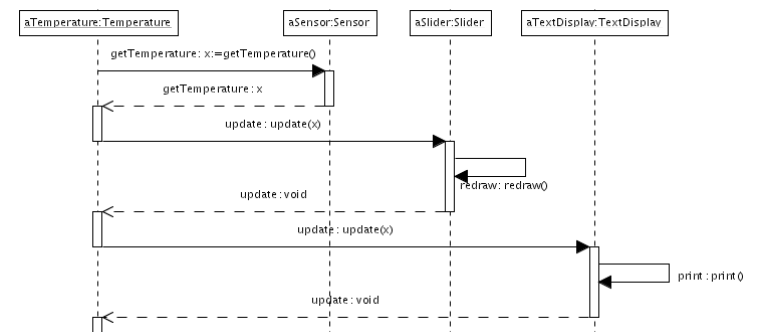
\longleftarrow - - - -

2.4.1 Sequenzdiagramme - Darstellung

- Auf der **horizontalen Achse** werden die **Objekte** aufgetragen
- Die **vertikale Achse** unter einem Objekt stellt die **Lebenszeit** des Objekts dar
- Ein **vertikaler Balken** unter einem Objekt stellt den **Steuerungsfokus** des Objekts dar
 - Steuerungsfokus ist der Zeitbereich, in dem das Objekt **aktiv** ist, also etwas tut.

2.4.1 Sequenzdiagramme - Darstellung

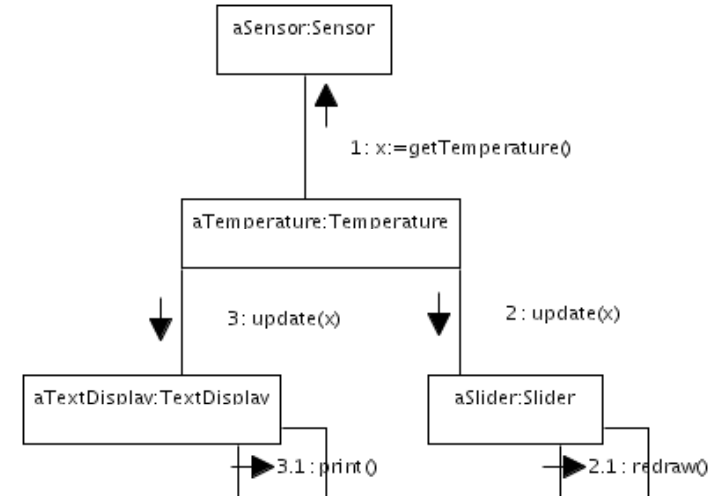
- Beispiel: es soll am Bildschirm die aktuelle Temperatur sowohl graphisch als Schieberegler als auch als Text angezeigt werden.
- Erst wird der Sensor aufgefordert, die aktuelle Temperatur zu liefern. Dann sollen der Schieberegler und die Textanzeige sich aktualisieren.



2.4.2 Kommunikationsdiagramm

- Ein Kommunikationsdiagramm stellt – ähnlich wie das Sequenz-Diagramm – die Interaktion zwischen Objekten dar.
- Der Schwerpunkt bei dieser Darstellung liegt weniger auf der zeitlichen Reihenfolge der Interaktion, sondern auf der Zusammenarbeit der Objekte untereinander.
- Kompaktere Darstellung als die eines Sequenzdiagramms.

2.4.2 Kommunikationsdiagramme - Darstellung



2.4.2 Kommunikationsdiagramme - Darstellung

- Ein Objekt wird ähnlich dargestellt wie bei Sequenzdiagrammen. Aber, die **Platzierung** der Objekte ist **frei wählbar**.
- Für jede **Interaktion** gibt es **einen Pfeil** vom aufrufenden Objekt zum Objekt, das die aufgerufene Methode besitzt. \longrightarrow
 - Die **Pfeilspitze** ist immer **gefülltes Dreieck**. Bei dieser Diagrammart ist es wichtig welche Objekte miteinander interagieren und nicht ob die Interaktion synchron oder asynchron ist.
- Durch Aufnahme der **Nummerierung** in die Beschriftung kann die **Reihenfolge** der Interaktion festgehalten werden. $\xrightarrow{1 \text{ getTemperature()}}$

2.4.3 Zustandsdiagramme

- Ein Zustandsdiagramm beschreibt einen Kontrollfluss, durch den Übergang von einem Zustand in den nächsten.
- Es dient zur Modellierung von Objekten,
 - deren Verhalten durch ihre Reaktion auf äußere Ereignisse gekennzeichnet ist bzw.
 - deren aktuelles Verhalten durch ihre Vergangenheit bestimmt ist.
- Ein Zustandsdiagramm besteht aus Zuständen und Transitionen (Zustandsübergängen).

2.4.3 Zustandsdiagramme - Darstellung

- Die **Zustände** werden als **Rechtecke mit gerundeten Ecken** dargestellt; das Rechteck enthält den Namen des Zustandes

Moving to first floor

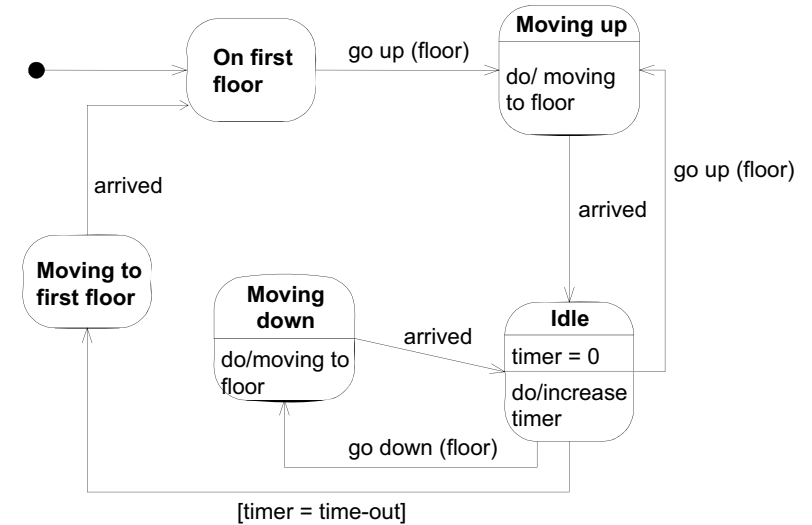
- optional kann das Rechteck unterteilt und mit Operationen beschriftet werden, die im betreffenden Zustand ausgeführt werden

Moving down
do/moving to floor

- Anfangs- bzw. Endzustand

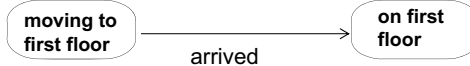


2.4.3 Zustandsdiagramme – Darstellung „Aufzugssteuerung“



2.4.3 Zustandsdiagramme - Darstellung

- Die **Transitionen** zwischen den Zuständen werden als **Pfeile** eingezeichnet




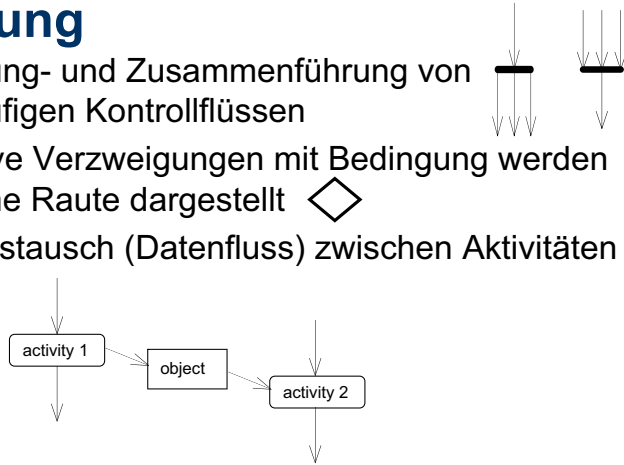
- Eine Transition kann mit dem Ereignis beschriftet werden, das sie auslöst. Ein Ereignis kann z.B. sein
 - eine Bedingung wird wahr $\xrightarrow{\text{arrived}}$
 - Empfang eines Signals $\xrightarrow{\text{go down (floor)}}$
 - Aufruf einer Operation $\xrightarrow{\text{timer = time-out}}$
 - Verstreichen einer bestimmten Zeitdauer nach Eintreten eines Ereignisses
- Wenn eine Transition unbeschriftet ist, dann schaltet sie sobald der davor liegende Zustand erreicht ist oder, falls der Zustand Operationen enthält, die Operationen ausgeführt wurden.
- zusätzlich kann eine Transition mit einer Operation beschriftet werden, die ausgeführt wird, wenn die Transition schaltet

2.4.4 Aktivitätsdiagramme

- Ein Aktivitätsdiagramm beschreibt **Geschäftslogik**.
- Ein Aktivitätsdiagramm besteht aus
 - Aktivitäten
 - Übergängen zwischen Aktivitäten (Kontrollfluss)
 - bei alternativen Übergängen Angabe von Entscheidungskriterien möglich
 - Objekttausch zwischen Aktivitäten (Datenfluss)

2.4.4 Aktivitätsdiagramme - Darstellung

- Aufspaltung- und Zusammenführung von nebenläufigen Kontrollflüssen
- Alternative Verzweigungen mit Bedingung werden durch eine Raute dargestellt 
- Objektaustausch (Datenfluss) zwischen Aktivitäten

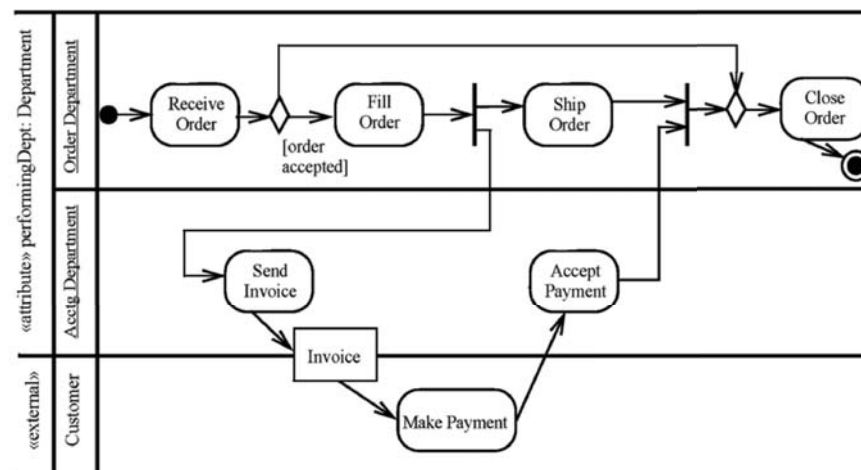


- Symbole können in sogen. Swimlanes (vertikalen oder horizontalen Bahnen) angeordnet werden, so dass Zuständigkeiten der Akteure deutlich werden

2.5 Literatur und Web-Ressourcen

- <http://www.smartdraw.com/resources/centers/uml/>
- H. Balzert: Lehrbuch der Software-Technik Software-Entwicklung, Spektrum Akad. Vlg. 2000, Band 1
- Boehm B. W.: A spiral model of software development and enhancement, Software Engineering Notes, 11(4), 1986
- Booch G.: Objektorientierte Analyse und Design: Mit praktischen Anwendungsbeispielen, Addison-Wesley, Bonn, Paris, 1. korr. Nachdruck, 1995
- Booch G., The Unified Modeling Language User Guide, Addison-Wesley, Reading/Mass., 1998
- Oestereich B., Die UML 2.0 Kurzreferenz für die Praxis, Oldenburg, 2004
- Eriksson H-E., Penker M.: UML toolkit, John Wiley & Sons, New York, 1998

2.4.4 Aktivitätsdiagramme - Darstellung



2.5 Literatur und Web-Ressourcen

- Fowler, Martin: UML Distilled Third Edition - A Brief Guide to the Standard Object Modeling Language. 3.Auflage, Addison-Wesley, 2004.
- Jacobson I., Christerson M., Jonsson P., Övergård G.: Object-oriented software engineering – A use case driven approach, Addison-Wesley, Reading/Mass., 1992
- Kahlbrandt B.: Software-Engineering – Objektorientierte Software-Entwicklung mit der Unified Modeling Language, Springer-Verlag, Berlin, 1998
- Mayer B.: Object-oriented software construction, 2. Auflage, Prentice-Hall, London, 1997
- www.uml.org
- Rational Software Corp. (editor): Object Constraint Language Specification, Vers. 1.1, Sept. 1997

2.5 Literatur und Web-Ressourcen

- Rumbaugh J.: The Unified Modeling Language Reference Manual, Addison-Wesley, Reading/Mass., 1998
- Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorenzen W.: Object-oriented modeling and design, Prentice-Hall, Englewood Cliffs/NJ, 1991
- Ross D. T.: Applications and extensions of SADT, IEEE Computer, April 1985
- Royce W. W.: Managing the development of large software systems: concepts and techniques, Proc. IEEE WESTCON, Los Angeles, 1-9[9]
- Yourdon E., Constantine L. L.: Structured design: Fundamentals of a discipline of computer program and systems design, Prentice-Hall, Englewood Cliffs/NJ, 1979