

<b>4 UML (UNIFIED MODELING LANGUAGE).....</b>	<b>3</b>
4.1 Was ist UML?.....	3
4.2 Versionen:.....	3
4.3 Diagrammtypen:.....	3
4.4 Use-Case-Diagramme.....	3
4.4.1 Zweck der Darstellung.....	3
4.4.2 Beispiel: .....	4
4.5 Allgemeine UML Elemente.....	6
4.6 Klassendiagramm (Class Diagram).....	7
4.6.1 Zweck der Darstellung.....	7
4.6.2 Darstellung von Klassen.....	7
4.6.3 Beziehungsarten.....	7
4.6.3.1 Beziehung (Association).....	7
4.6.3.2 Aggregation / Komposition.....	8
4.6.3.3 Generalisierung (Generalization).....	9
4.6.4 Beispiel.....	10
4.6.5 Darstellung Interface bzw. implementierende Klasse.....	11
4.6.6 Detaillierte Darstellung von Attributen und Methoden.....	11
4.6.7 Wie kommt man zu Klassen?.....	12
4.6.8 Beispiel: KFZ-Versicherung.....	12
4.6.9 Wie kommt man zu Eigenschaften von Klassen?.....	12
4.7 Paket-Diagramm.....	13
4.7.1 Zweck der Darstellung.....	13
4.8 Aktivitätsdiagramm (Activity Diagram).....	13
4.8.1 Zweck der Darstellung.....	13
4.9 Zustandsdiagramm (Statechart Diagram) .....	19
4.9.1 Grundelemente.....	19
4.9.2 Weitere Möglichkeiten.....	20
4.10 Sequenzdiagramm –(Sequence Diagram).....	22
4.10.1 Zweck der Darstellung.....	22
4.10.2 Nachrichten zwischen Objekten.....	22
4.10.3 Einführendes Beispiel.....	22
4.10.4 Grundelemente.....	24
4.10.5 Pfeilarten.....	24
4.10.6 Nachrichtenformat.....	24



## 4 UML (Unified Modeling Language)

### 4.1 Was ist UML?

Die UML ist ein Satz von Notationen zur Beschreibung objektorientierter Softwaresysteme.  
Ihre Autoren sind:

- α **Grady Booch**
- α **James Rumbaugh**
- α **Ivar Jacobson**

(z.Zt. sind alle drei bei *Rational, Inc.*, <http://www.rational.com> beschäftigt)

Wichtigstes Tool: Rational Rose

### 4.2 Versionen:

UML 1.1	November 1997
UML 1.3	November 1997
UML 2.0	Geplant für 2002

### 4.3 Diagrammtypen:

α <b>Anwendungsfalldiagramm</b> <i>Use Case Diagram</i> (Akteure, Szenarios)	• Anforderungen (Requirements)
α <b>Klassendiagramm</b> <i>Class Diagram</i> (Klassen, Beziehungen)	Statische Sicht: log. Aufbau des Systems
α <b>Paket-Diagramm</b> (Strukturierung der Darstellung)	
α <b>Kollaborationsdiagramm</b> <i>Collaboration Diagram</i> (Zusammenwirken der Komponenten)	
α <b>Aktivitätsdiagramm</b> <i>Activity Diagram</i> (Ablaufmöglichkeiten)	Dynamische Sicht: Interaktionen, Abläufe
α <b>Sequenzdiagramm</b> <i>Sequence Diagram</i> (Objekte, Interaktionen)	
α <b>Zustandsdiagramm</b> <i>Statechart Diagram</i> (Internes Verhalten von Objekten)	
α <b>Komponentendiagramm</b> <i>Component Diagram</i> (Innere Struktur der Objekte)	Implementierung
α <b>Verteilungsdiagramm</b> <i>Deployment Diagram</i> (Einbettung der Objekte in eine Umgebung)	

### 4.4 Use-Case-Diagramme

#### 4.4.1 Zweck der Darstellung

- Zeigen die benötigten Interaktionen zwischen dem System und den Akteuren
- Werden durch Szenarios beschrieben (Normalfall + Problemfälle)
- Grundlage für die Erstellung des Systems (müssen daher vollständig sein!)

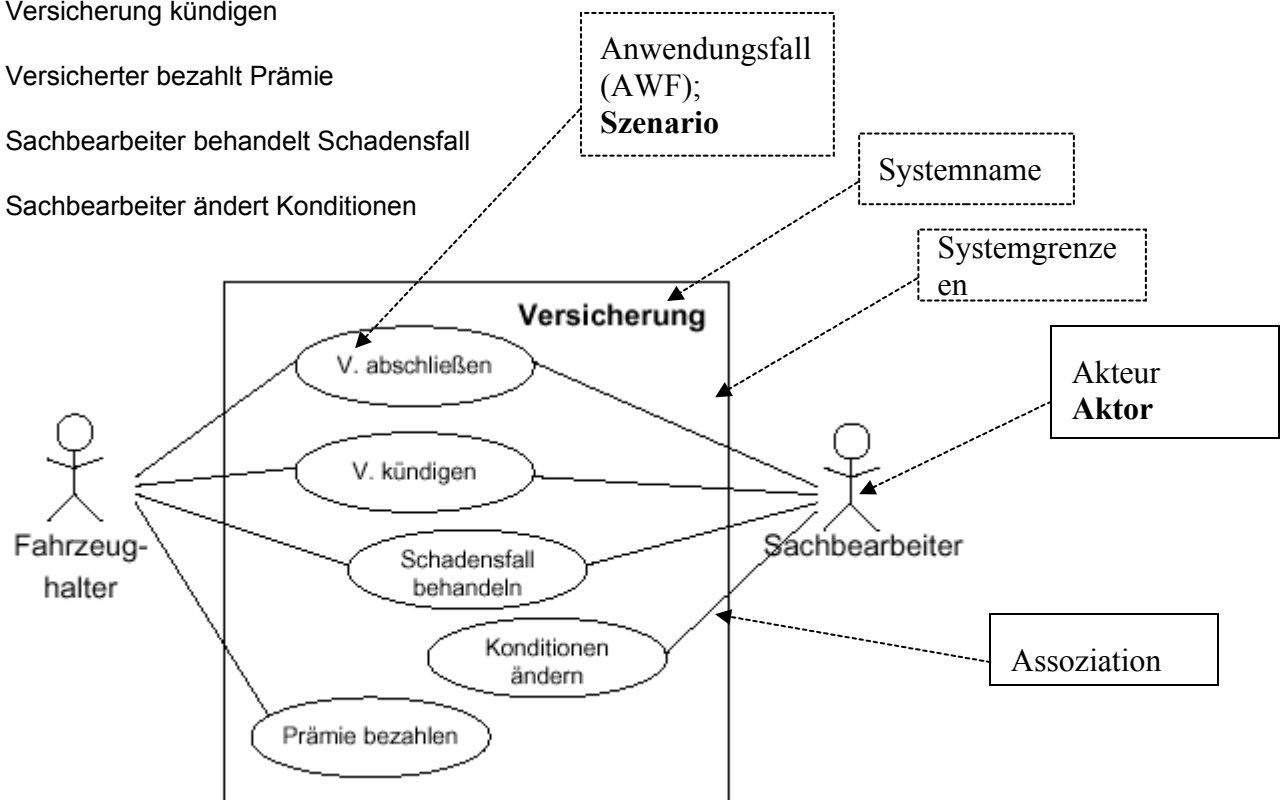
- Grundlage für das Testen des Systems nach der Erstellung

**Wichtig:** Use Cases beschreiben gewünschte Eigenschaften und nicht das System wie es ist!

#### 4.4.2 Beispiel:

### Use-Cases für eine Versicherung (KFZ):

- α Versicherung abschließen
- α Versicherung kündigen
- α Versicherter bezahlt Prämie
- α Sachbearbeiter behandelt Schadensfall
- α Sachbearbeiter ändert Konditionen



### Szenario "Schadensfall behandeln"

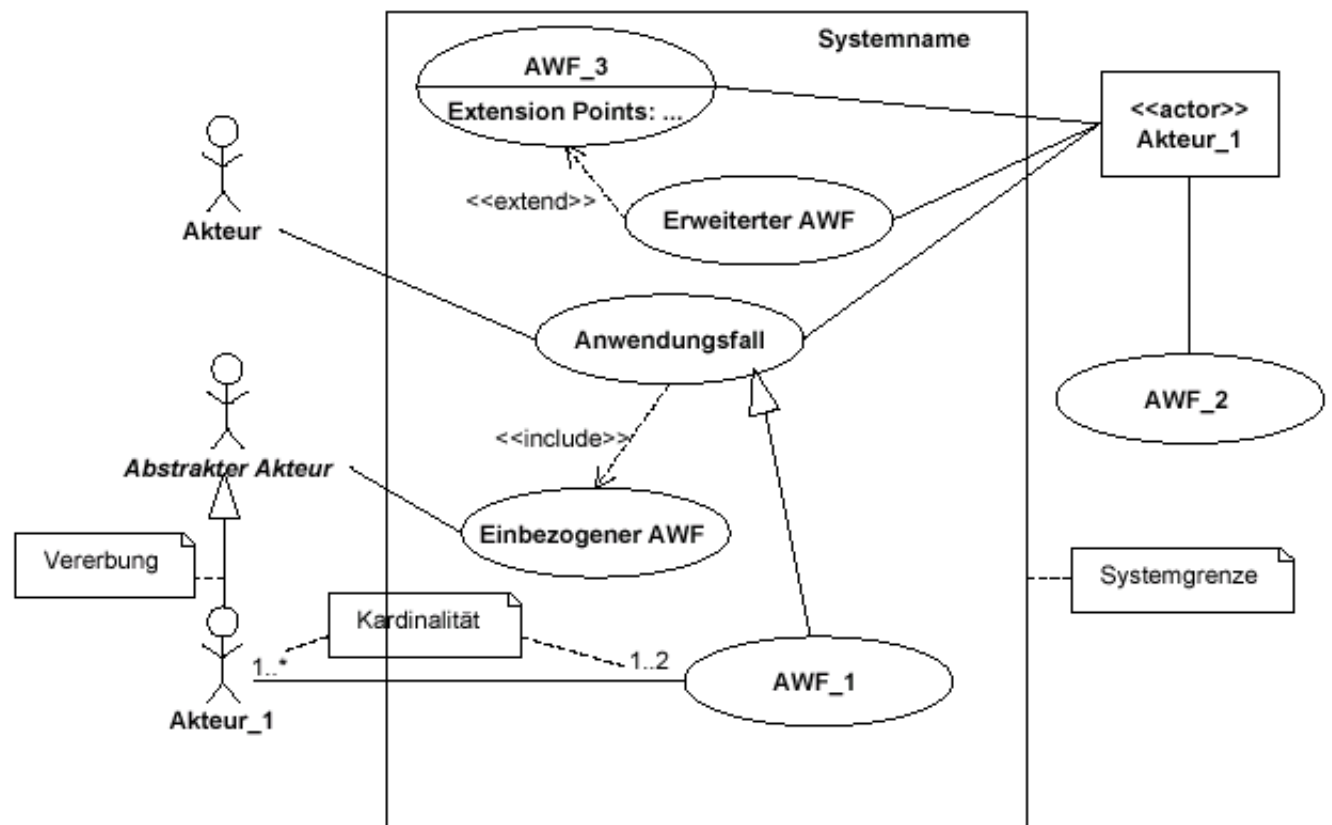
- α • Schaden wird von der Versicherung bezahlt
  - Eigenverschulden
  - Fremdverschulden
- α • Fahrzeughalter bezahlt selbst
- α • Gerichtsverhandlung
- α • Andere Versicherung bezahlt
- α • ...




### Akteur:

Ein Anwendungsfall muss von einer außenstehenden Person oder Sache instanziiert werden. Diese Instanz wird als Akteur bezeichnet.

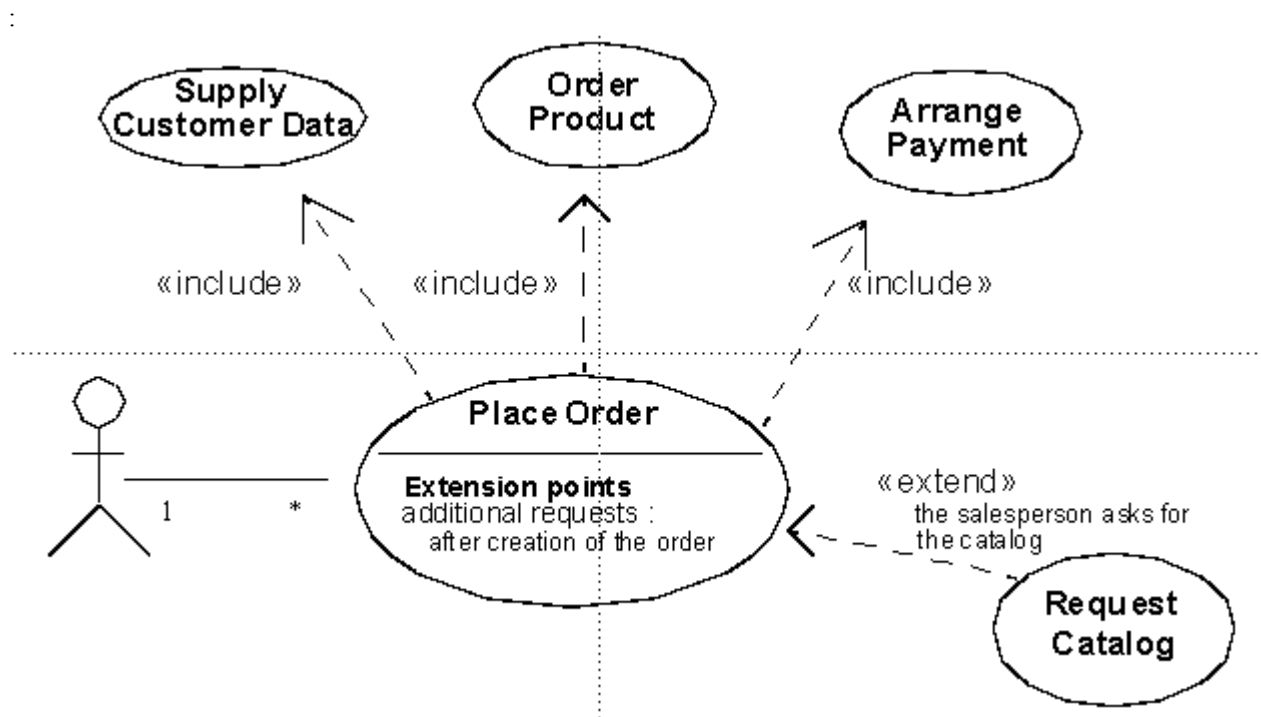
### Akteur/Anwendungsfall-Assoziation:

Eine Beziehung zwischen einem Akteur und einem Anwendungsfall bedeutet, dass entweder der Akteur den Anwendungsfall initiiert oder der Anwendungsfall dem Akteur Ergebnisse liefert oder beides.



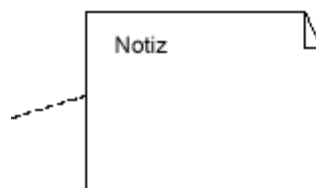
<b>Begriff</b>	<b>Beschreibung</b>	<b>Syntax</b>
generalization	Beziehung eines allgemeinen Anwendungsfall zu einem mehr Spezialisierten.	 <<extend>>
extend	Beziehung von einem erweiterten Anwendungsfall zum Basis Anwendungsfall.	
include	Beziehung von einem Basis Anwendungsfall zu einem inkludierten Anwendungsfall	<<include>> 
Kardinalität	Anzahl der Akteure bzw. Anwendungsfälle (nächstes Beispiel: 1 Person kann mehrere Aufträge geben)	1..2, oder 1..*

Weiteres Beispiel:

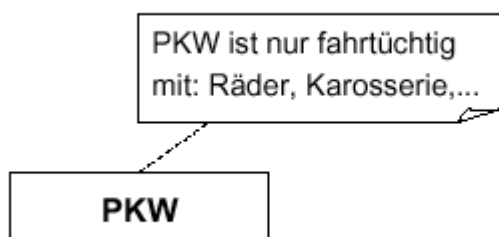


#### 4.5 Allgemeine UML Elemente

Notiz:



Beispiel:



#### Aufgabenstellungen:

Reservierungssystem für Equipment (Beamer, Präsentationsräume)

Akteure:     Equipmentverwalter  
               Person die Reservierung durchführen will

Bücherei

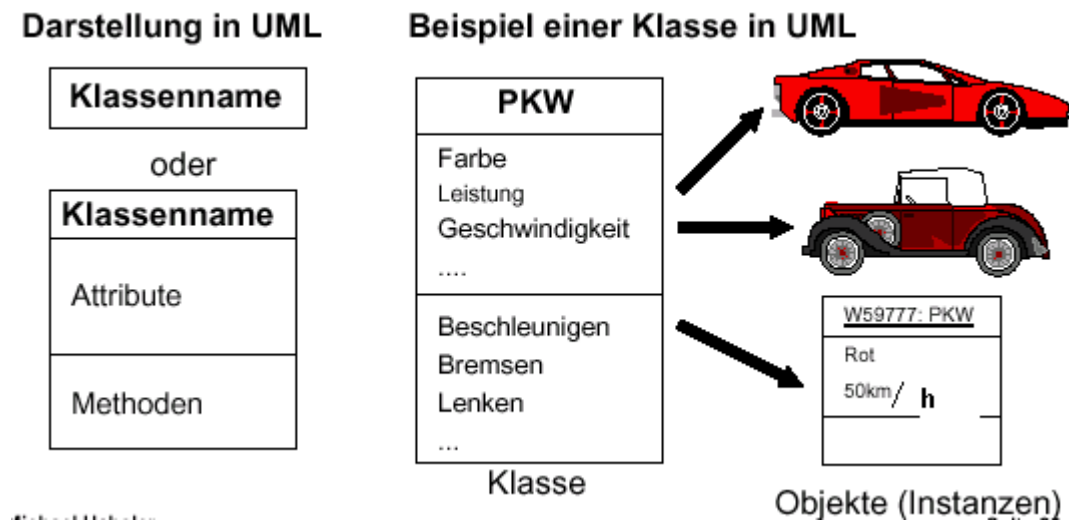
Andere von den Lastenheften

## 4.6 Klassendiagramm (Class Diagram)

### 4.6.1 Zweck der Darstellung

- α Logischen Aufbau des Systems
- α Statischen Aspekte (z. B. aus welchen Klassen besteht das System)
- α Zusammenhänge und Beziehungen zwischen den Komponenten

### 4.6.2 Darstellung von Klassen



### 4.6.3 Beziehungsarten

Eine **Beziehung** verbindet wechselseitig zwei oder mehrere Klassen.

Eine **Verknüpfung** ist eine physikalische oder konzeptuelle Verbindung zwischen **Objekten**.

z.B. *Hr. Huber arbeitet bei Fa. K&A*  
*Hr. Schuster besitzt ein KFZ mit Kennzeichen HA 12345*

#### 4.6.3.1 Beziehung (Association)

Eine Association beschreibt Verknüpfungen zwischen Objekten der beteiligten Klassen.

Klasse A „ist zugeordnet zu“, „spricht mit“ oder „hat Beziehung zu“ Klasse B.

Die Begriffe sind im Gegensatz zu „besteht aus“ zu sehen.

#### • Beziehungen (Paths/Associations)



#### • Richtung der Beziehungen (Navigability)



## $\alpha$ Kardinalität (Multiplicity)

Die Kardinalität spezifiziert, wie viele Objekte einer Klasse mit Objekten einer assoziierten Klasse verknüpft sein können.

- Kardinalitäten (Multiplicity)

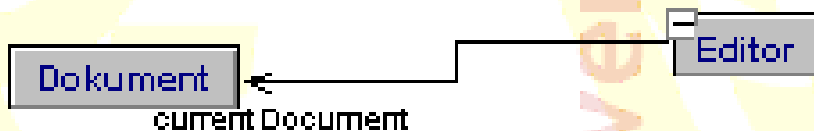


## $\alpha$ Rollennamen

Ein Rollenname beschreibt die Funktion von Objekten in einer Beziehung.



*Beispiel: Beziehung der Editorklasse zu einer Dokumentklasse:*



Sourcecode:

```
class Dokument {
}
class Editor {
    Dokument currentDocument;
}
```

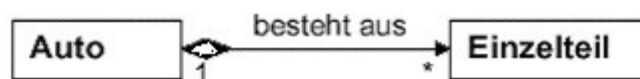
## 4.6.3.2 Aggregation / Komposition

Die **Aggregation** ist eine spezielle Form der Assoziation, bei der Teile eines Ganzen mit dem Ganzen in Beziehung gebracht werden.

Klasse A „besteht aus“ Klasse B bzw. Klasse B „ist Teil von“ Klasse A

Z.B.

- Aggregation



Das Auto „besteht aus“ Einzelteil bzw. ein Einzelteil „ist Teil“ eines Autos.

Diese Art der Beziehung hat die Auswirkung, dass die Objekte der Klasse „**Einzelteil**“ nur existieren, wenn ein Objekt der Klasse „**Auto**“ existiert.



Beim Auflösen des Autos (Löschung des Objektes) können aber die Einzelteile einer anderen Verwendung (in einem anderen Fahrzeug integriert) zugeführt werden.

```
public class Auto {
    Einzelteil[] einzelteile = new Einzelteil[1000];

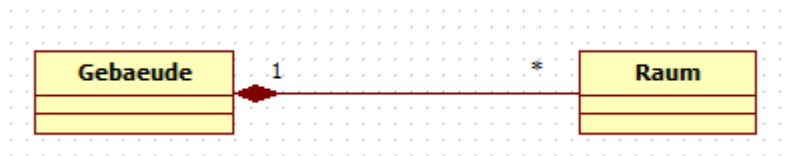
    public Auto() {
        einzelteile[0] = new Einzelteil("Lenkrad");
        ...
    }
}
```

Die **Komposition** ist ein Sonderfall der Aggregation und beschreibt die Beziehung zwischen einem *Ganzen* und seinen *Teilen*. Der Unterschied zur Aggregation ist das ein Objekt nur zu **einem** (nicht mehreren) übergeordneten Objekte zugeordnet ist.

Z. B. ein Raum kann immer nur zu genau einem Gebäude gehören, nie zu keinem oder mehreren.

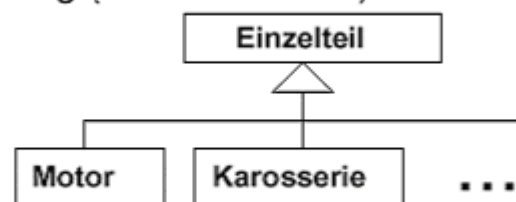
Dies hat auch Auswirkungen auf den Lebenszyklus der *Teile (Objekte)*, d. h. die [Instanz](#), welche das *Ganze* repräsentiert, übernimmt auch die Verantwortung für die Lebensdauer der Instanzen. Hört das übergeordnete Objekt zu existieren auf, dann existieren auch die untergeordneten Objekte nicht mehr (in JAVA automatisch gelöscht).

**Begriffe für C++:** Aufruf des Destruktors von Gebäude, dieser Destruktor muss die Destruktoren der Einzelteile aufrufen.



#### 4.6.3.3 Generalisierung (Generalization)

- Generalisierung (Generalization)



Durch Vererbung (Generalisierung bzw. Spezialisierung) werden die Klassen aufgrund ihrer strukturellen Gemeinsamkeit und Unterschiede hierarchisch gegliedert.

Hier: Ein Motor „ist ein“ Einzelteil.

Eine Unterklasse erbt von der Oberklasse:

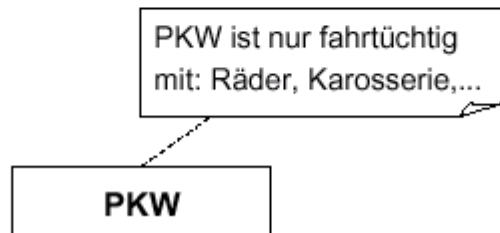
α Attribute

- $\alpha$  Methoden
- $\alpha$  Assoziationen

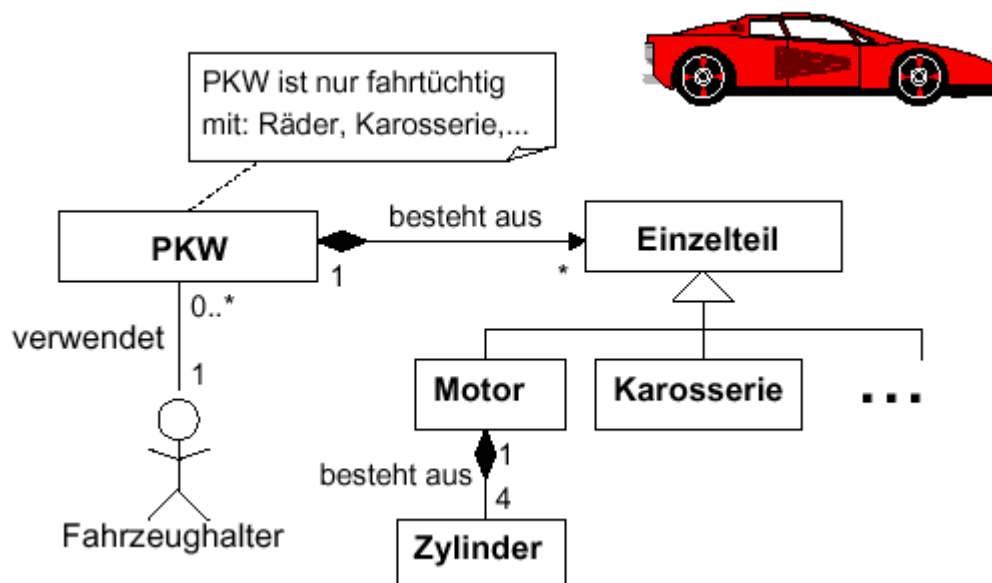
## • Abhängigkeiten (Dependencies)



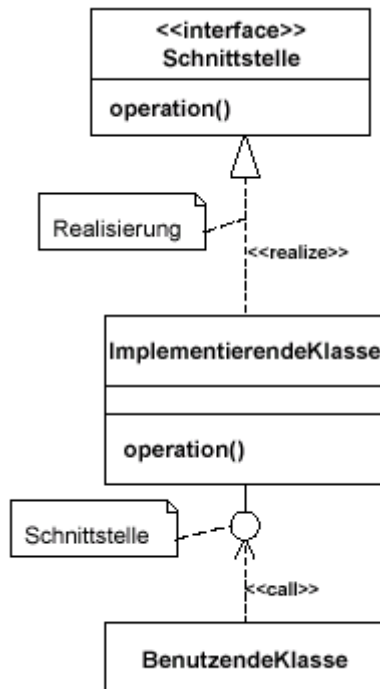
## • Anmerkungen oder Einschränkung



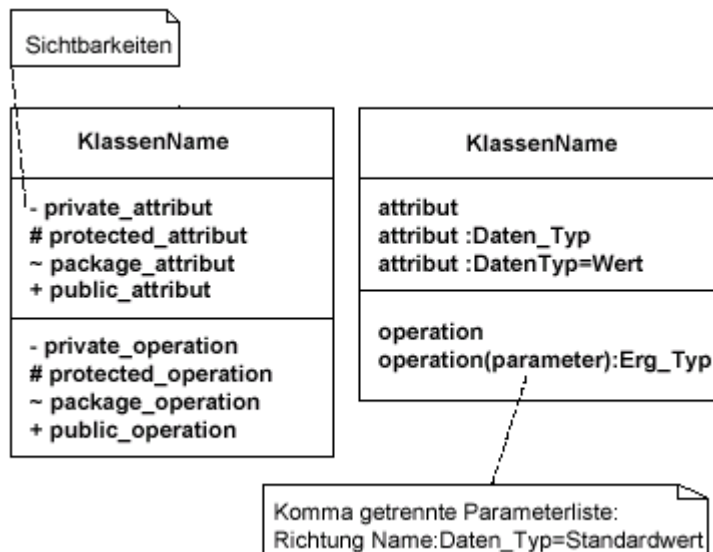
### 4.6.4 Beispiel



#### 4.6.5 Darstellung Interface bzw. implementierende Klasse

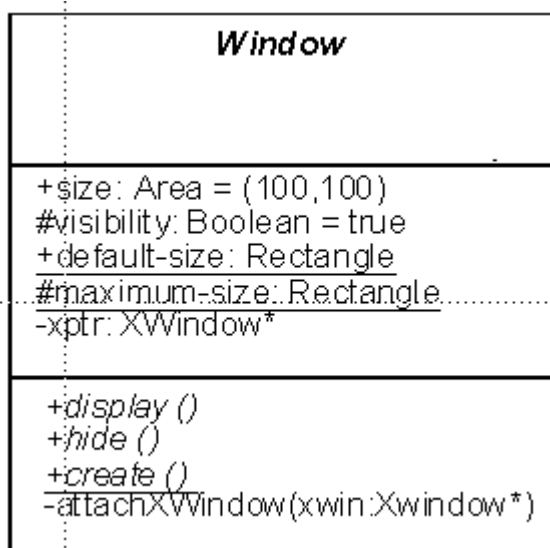


#### 4.6.6 Detaillierte Darstellung von Attributen und Methoden



z.B.

Beispiel: für Richtung



#### 4.6.7 Wie kommt man zu Klassen?

α Analyse der Use-Cases:

- Substantive sind Kandidaten für Klassen
- Verben sind Kandidaten für Methoden

α Auswahl der Klassen

- Synonyme finden
- Unwichtige Klassen entfernen
- Wichtige Klassen hinzufügen (Brainstorming)

α Organisation der Klassen

#### 4.6.8 Beispiel: KFZ-Versicherung

Szenario "KFZ-Versicherung"

Ein **Fahrzeughalter** muss für sein **KFZ** eine **Versicherung** abschließen. Er wird bei der **Versicherung** von einem **Sachbearbeiter** betreut. Nach dem Abschluss bekommt der **Versicherte** einen **Vertrag** mit den genauen **Versicherungsdaten** und der Sachbearbeiter führt **Unterlagen** zu etwaigen **Schadensfällen**.

#### 4.6.9 Wie kommt man zu Eigenschaften von Klassen?

##### • Attribute

Was muss sich das Objekt merken können?

##### • Methoden

Welches Verhalten (Funktionen) soll das Objekt haben?

z.B. PKW

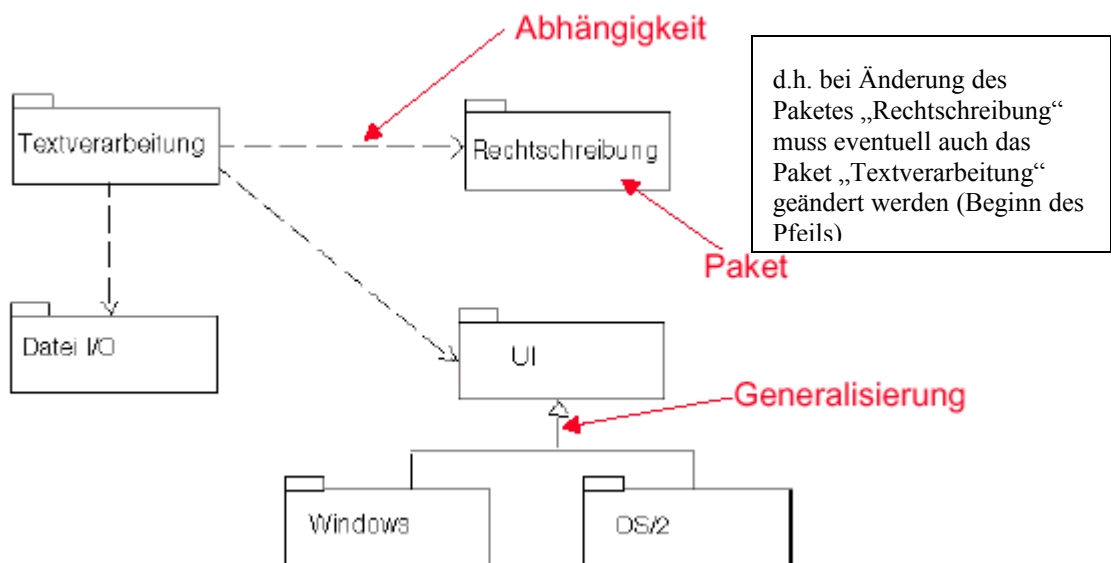
PKW
Farbe Leistung Geschwindigkeit ....
Beschleunigen Bremsen Lenken ...

**Aufgaben:** *Klassendiagramme für erstellte USE CASES:*

## 4.7 Paket-Diagramm

### 4.7.1 Zweck der Darstellung

- α Es werden als **Übersicht** Gruppen von Diagrammen oder Elementen zusammengefasst. Pakete enthalten mehrere Klassen.
- α Es werden **Abhängigkeiten** angegeben. Eine Abhängigkeit bedeutet, dass das abhängige Paket durch Änderungen im anderen Paket betroffen ist (Änderung des Codes oder neuerliche Kompilierung kann notwendig sein)



## 4.8 Aktivitätsdiagramm (Activity Diagram)

### 4.8.1 Zweck der Darstellung

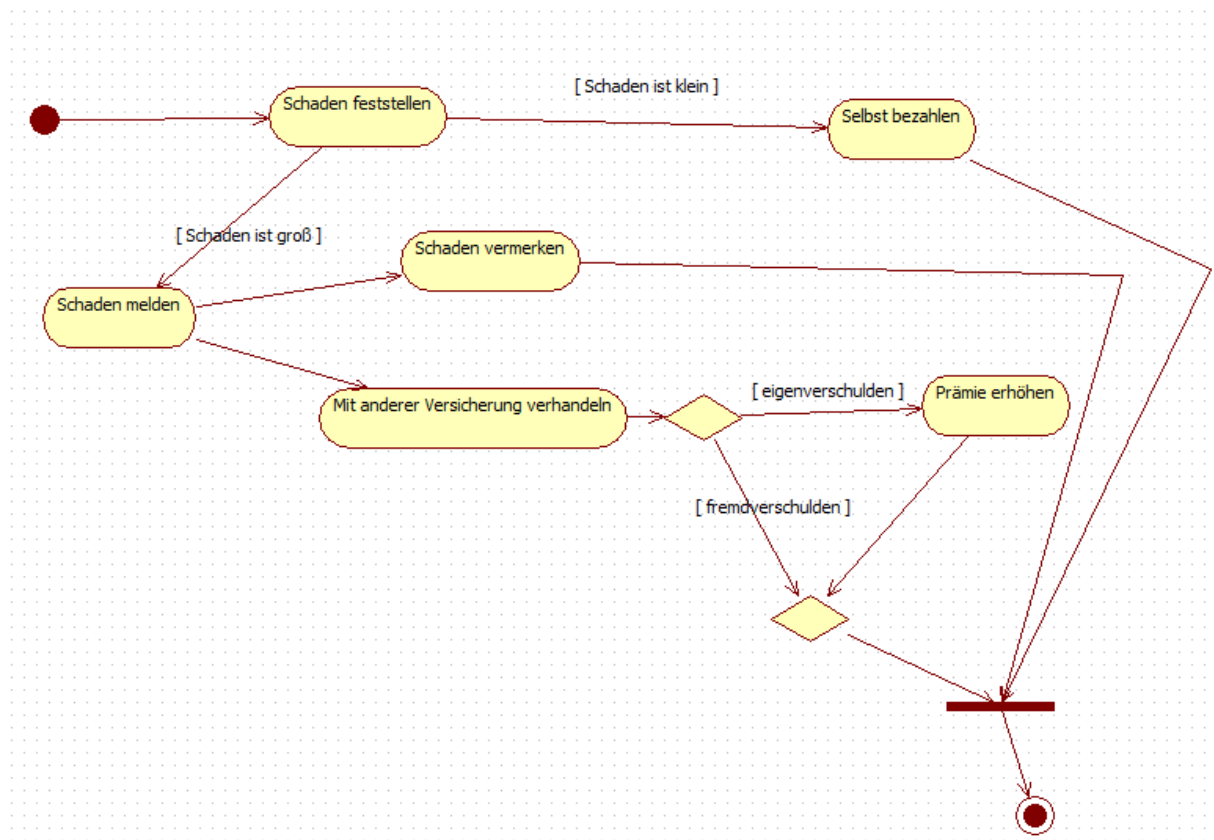
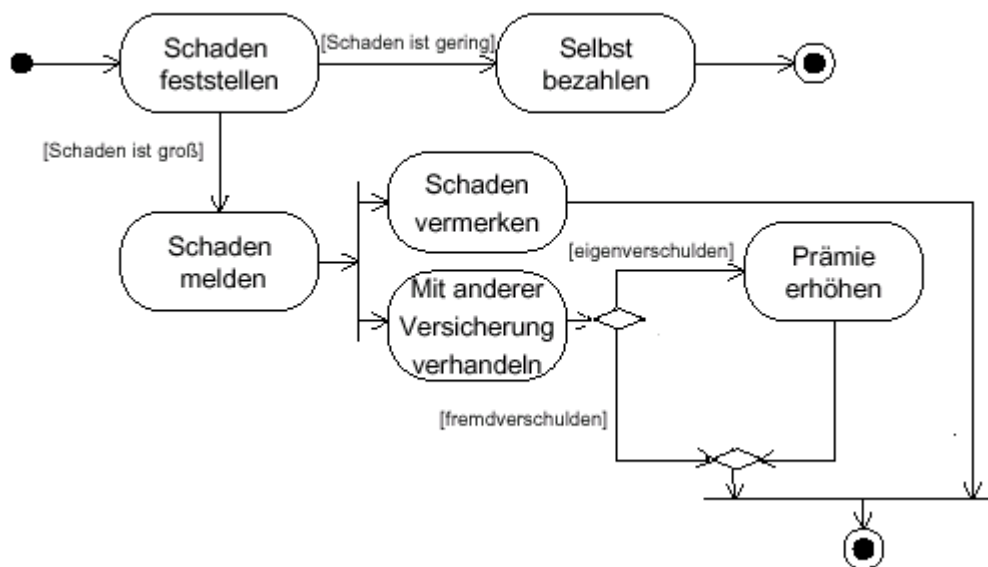
- α Ablaufmöglichkeiten eines Systems
- α Aktivitäten sind zugeordnet zu
  - Klassen
  - Operationen

- einem Use Case

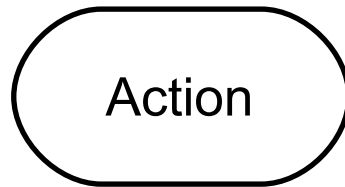
*Beispiel:*

### **Szenario "Schadensfall"**

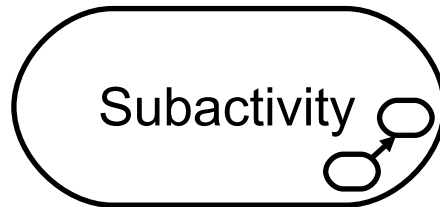
Ein Schadensfall tritt ein. Der Sachbearbeiter wird informiert und bearbeitet den Fall. Der Vorfall wird in den Versicherungsunterlagen vermerkt. Bei Verschulden durch den Versicherungsnehmer wird die monatliche Prämie erhöht.



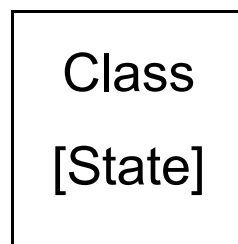
### Aktion (action)



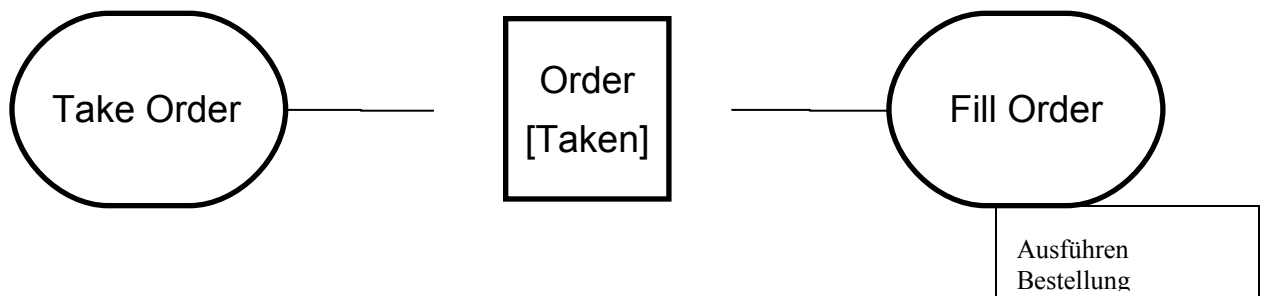
### Unteraktivitäten (Subactivity)



### Objektfluss (Objectflow)



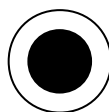
Ein Objekt der Klasse steht in diesem Zustand zur Verfügung. Z.B. eine Bestellung nach einem Bestellvorgang.



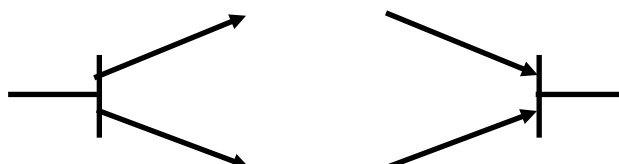
### Startzustand



### Endzustand



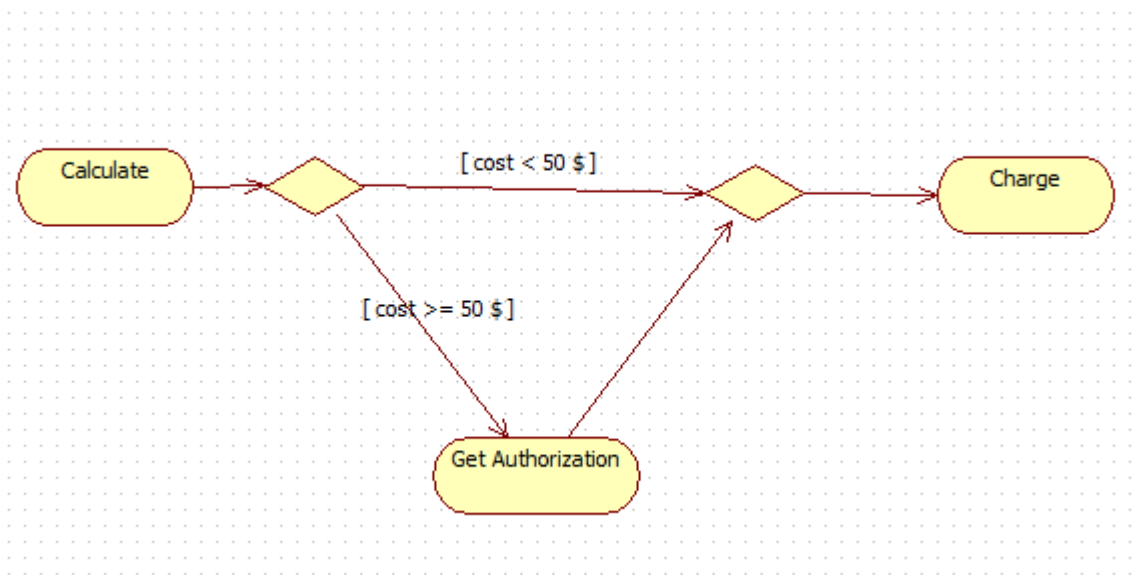
### Parallele Aktivitäten



Fork

Join

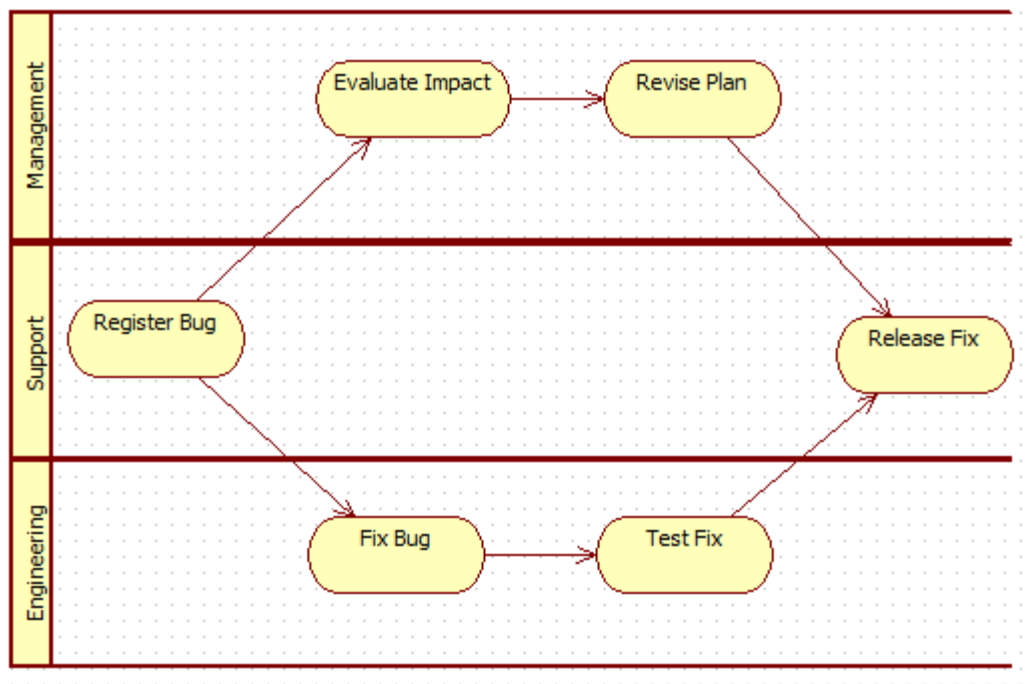
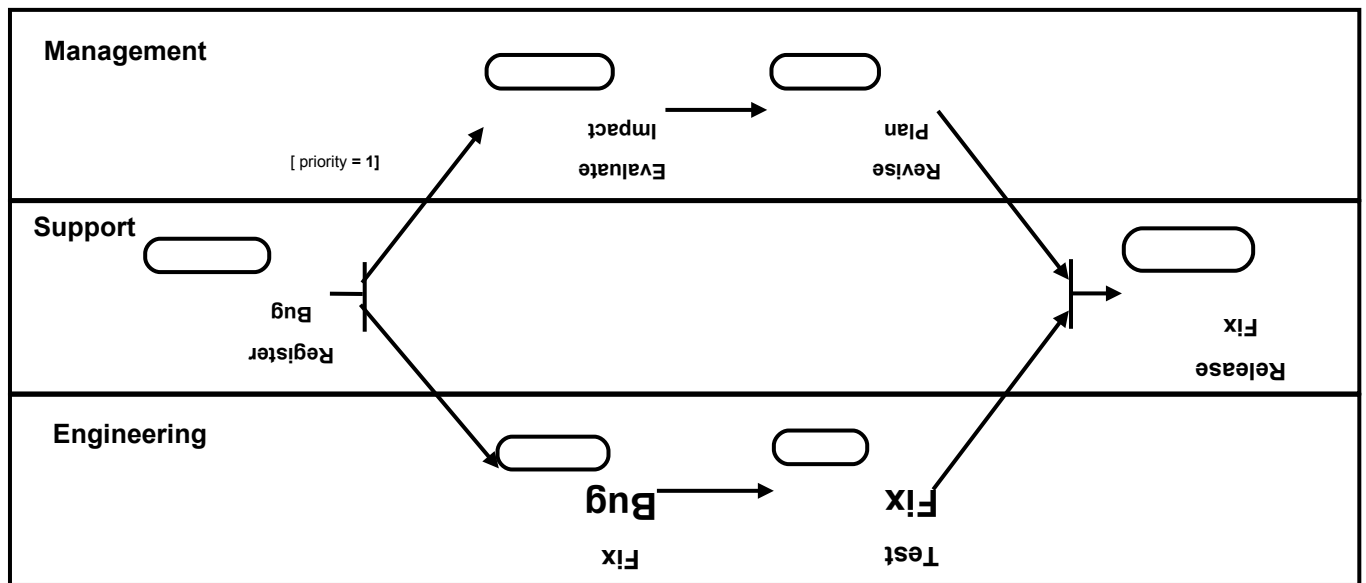
Entscheidung  
z.B.





## Gruppierung in Partitions

swimlane notation:



## Signale senden empfangen

Wird die Darstellung einer Kommunikation mit einem anderen Szenario bzw anderen Klassen verwendet.

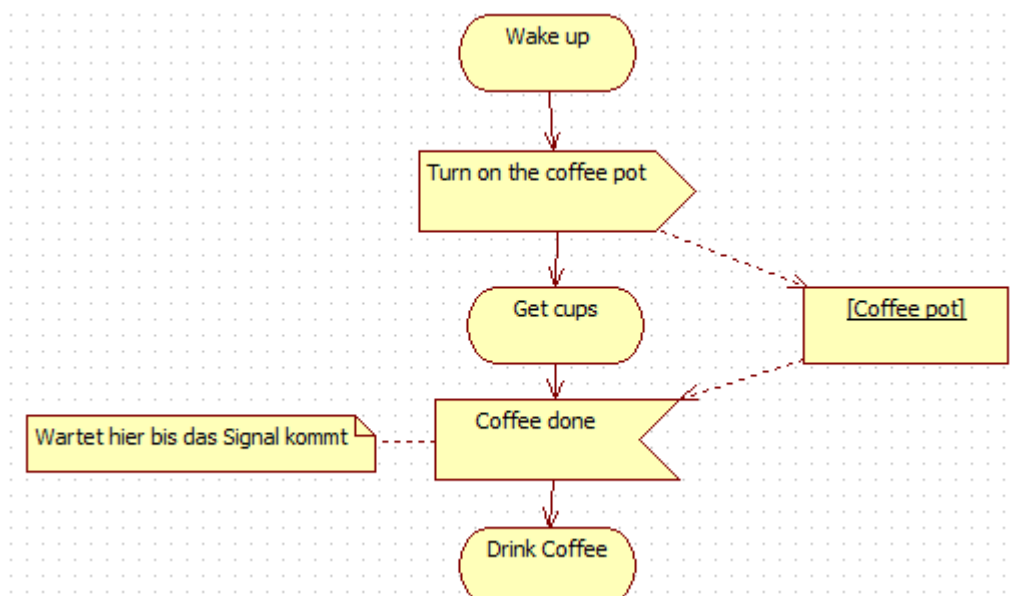
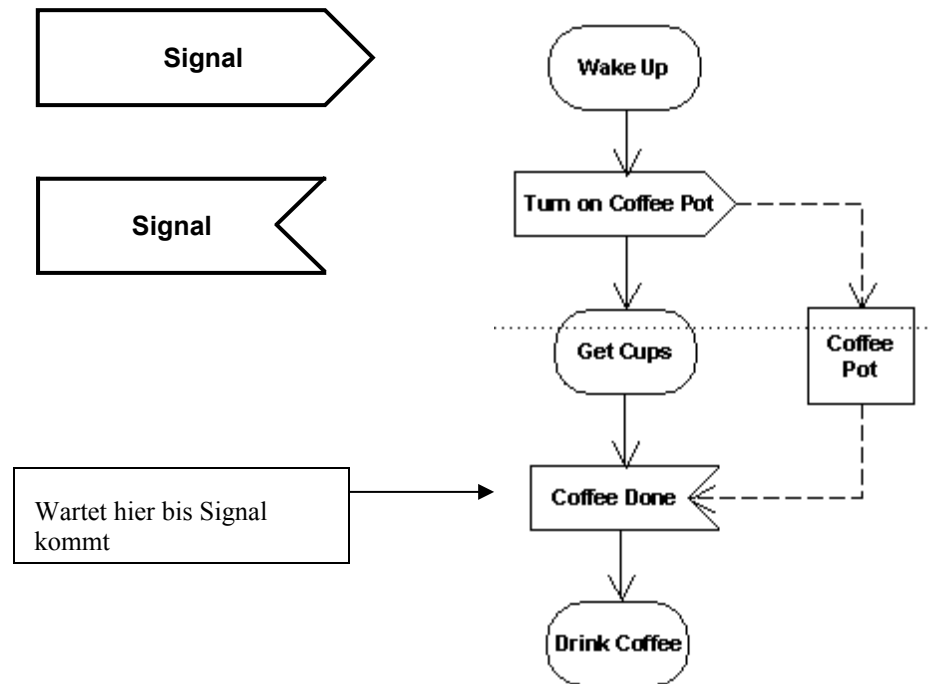
Signal senden:



Signale empfangen:



z.B.



## 4.9 Zustandsdiagramm ( Statechart Diagram)

### 4.8.1 Zweck der Darstellung

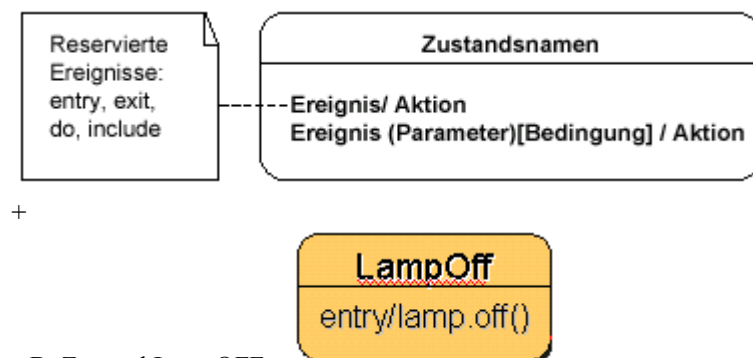
Dient der Darstellung des internen Verhaltes von Objekten

Das Zustandsdiagramm enthält

- Zustände (States)
- Ereignisse (Events)
- Übergänge zwischen Zuständen (Transitions)

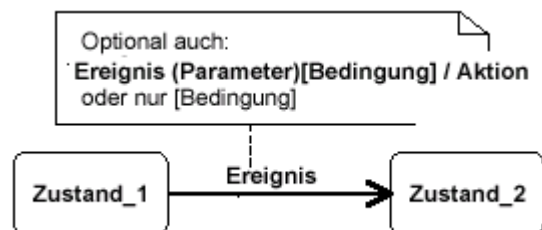
### 4.9.1 Grundelemente

**Zustand:**

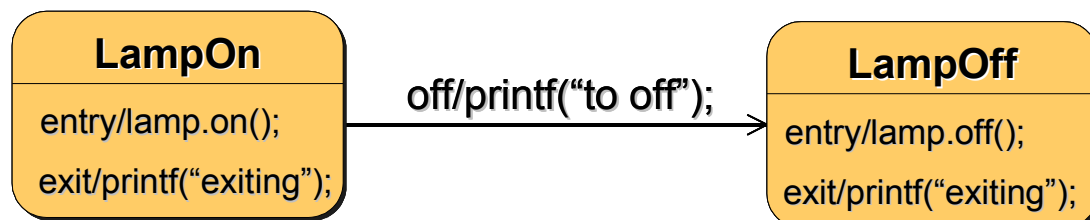


z.B. Zustand LampOFF

**Zustandsübergang:**

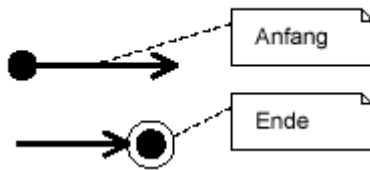


Z.B. Statuswechsel bei von LampOn auf LampOff

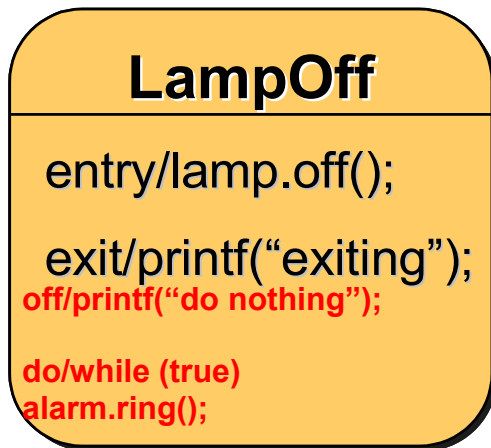


#### 4.9.2 Weitere Möglichkeiten

Anfangs-Endzustand:

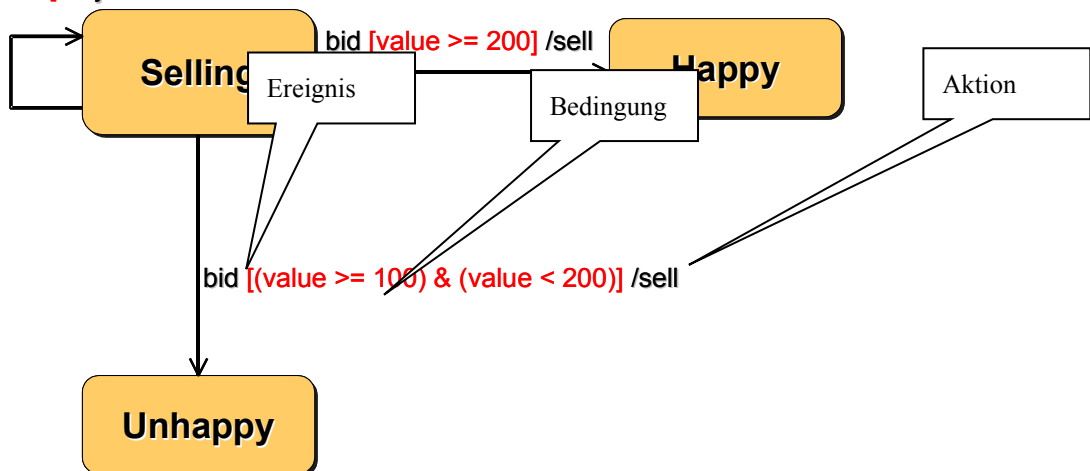


Interne Aktionen und „do“-Aktivität (paralleler Thread):

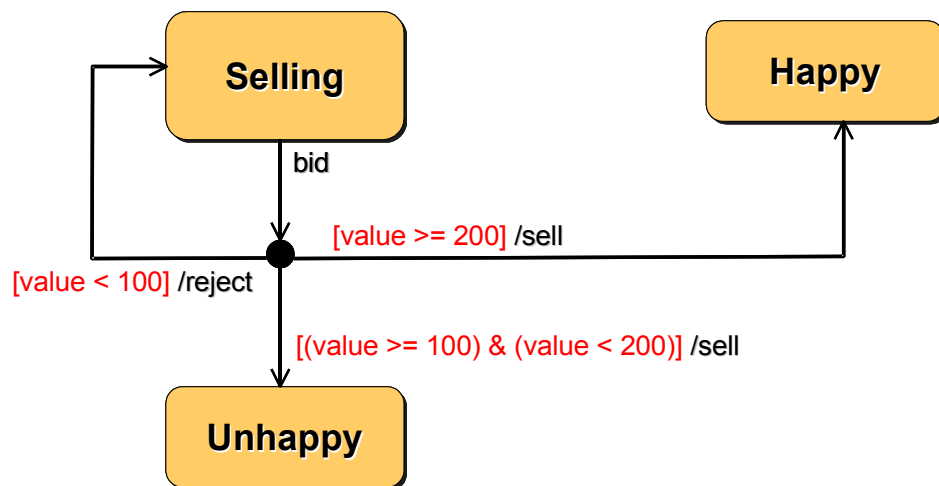


Bedingte Ausführung:

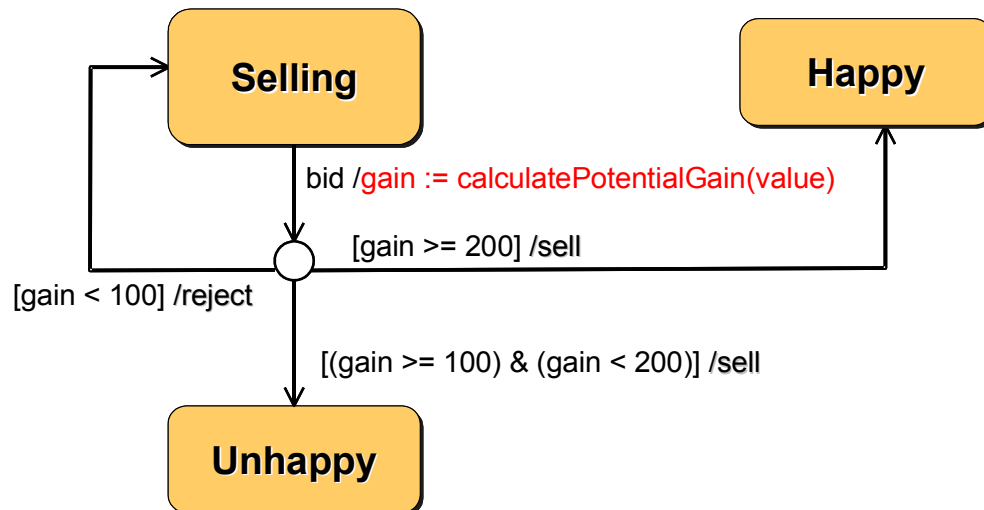
bid [value < 100] /reject



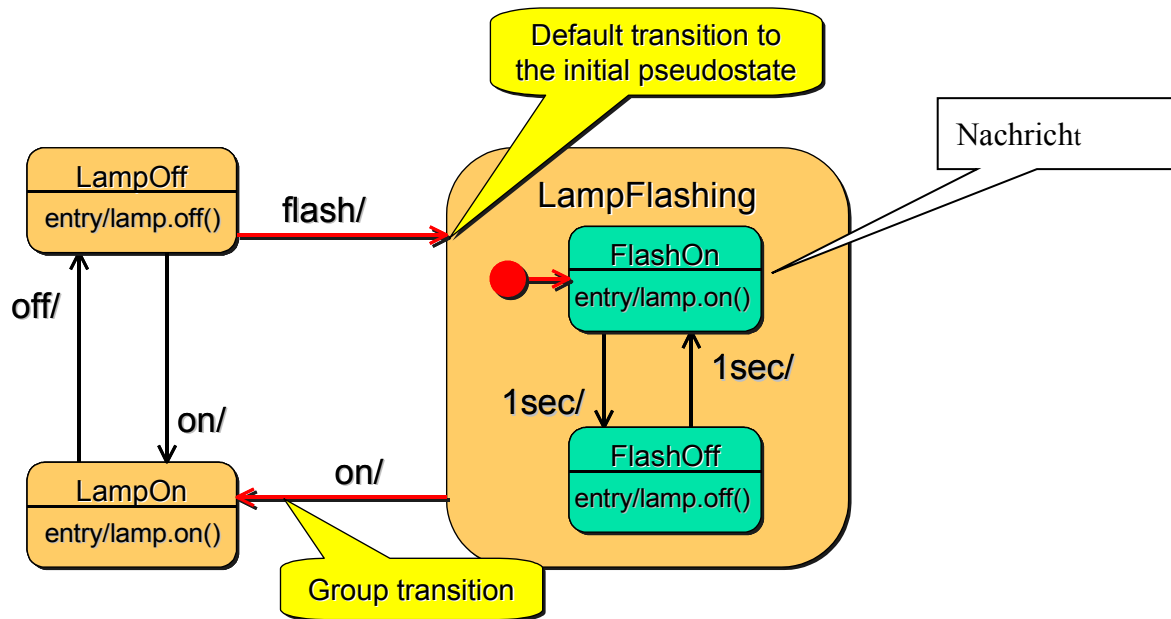
**Bedingte Ausführung mit Knotenpunkt:**



**Dynamischer Knotenpunkt ( dynamic choicepoint):**



Unterzustände:



Sehr detailliert S87

## 4.10 Sequenzdiagramm –(Sequence Diagram)

### 4.10.1 Zweck der Darstellung

- **Interaktion** zwischen Objekten aus Szenarios
- Identifikation von zusätzlichen Klassen und Methoden

Das Sequenzdiagramm stellt

- ⤴ Objekte
- ⤴ Lebenszeit von Objekten

dar.

### 4.10.2 Nachrichten zwischen Objekten

### 4.10.3 Einführendes Beispiel

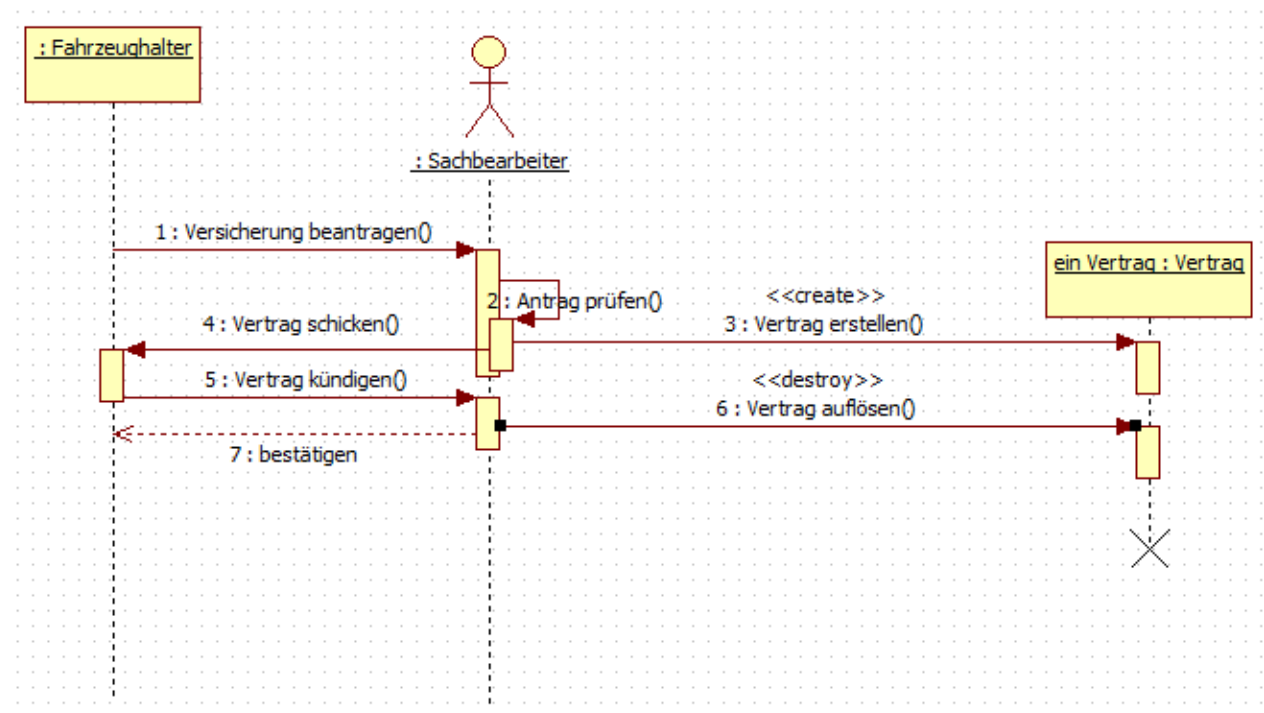
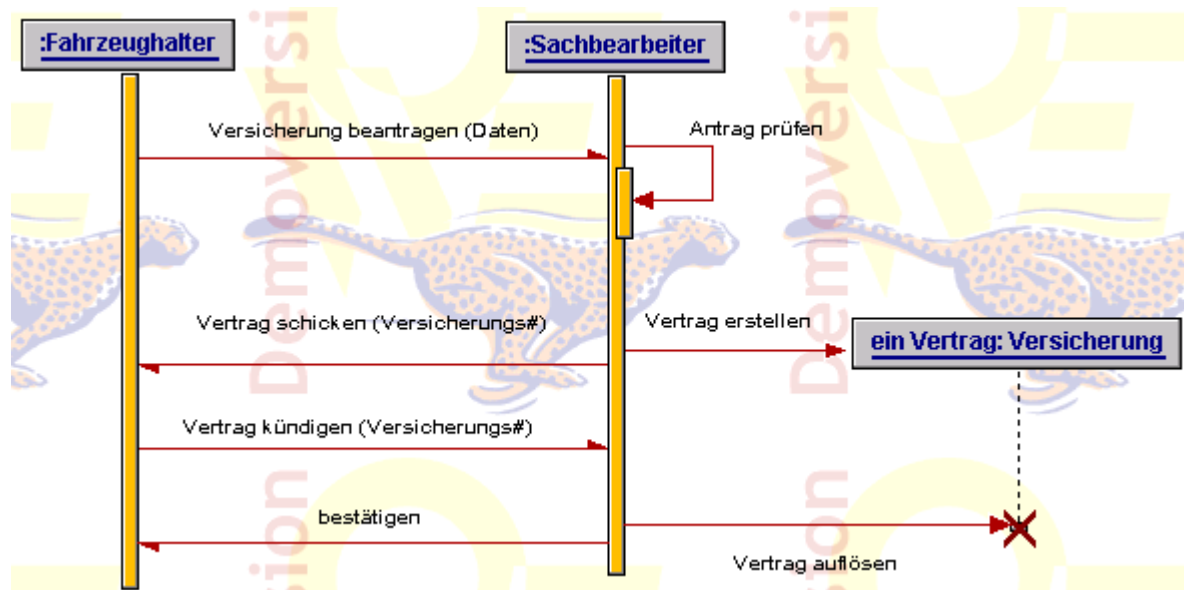
## KFZ-Versicherung

Szenario "Versicherung abschließen"

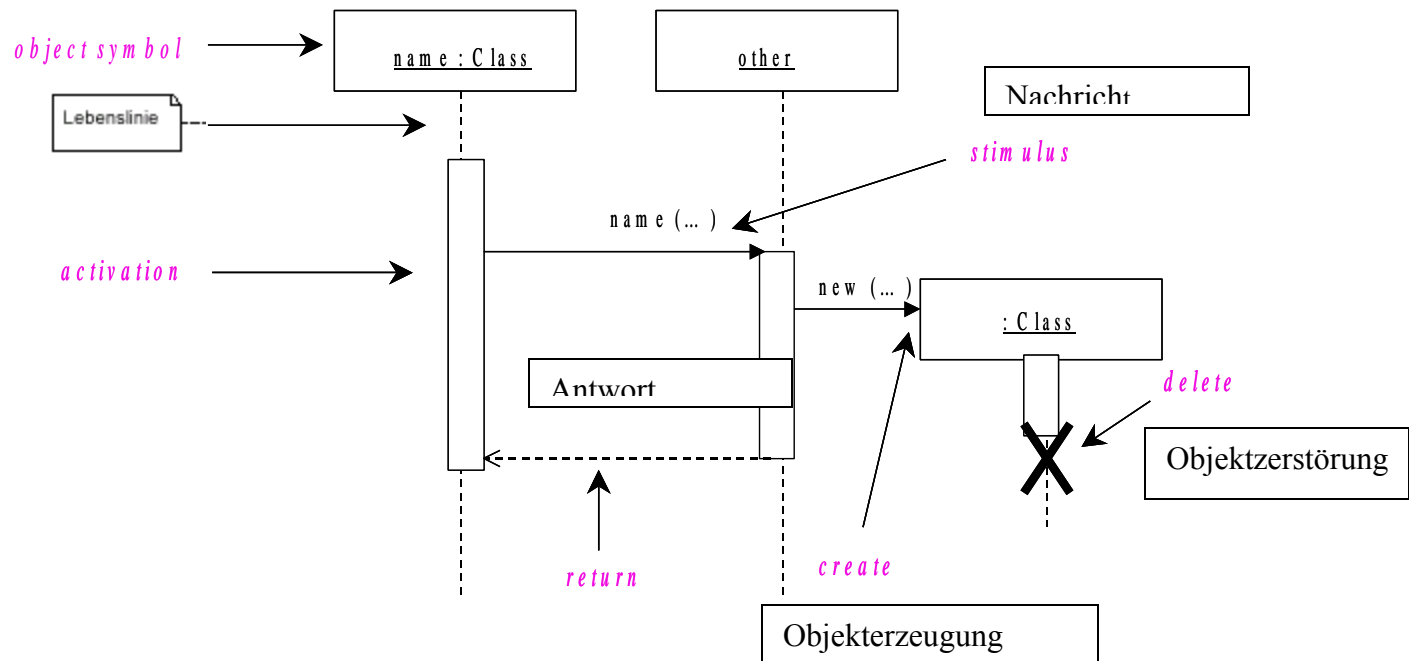
Ein **Fahrzeughalter** *beantragt* eine **KFZ-Versicherung**. Der **Antrag** wird vom zuständigen **Sachbearbeiter** *geprüft*. Nach erfolgreicher Prüfung wird ein **Vertrag abgeschlossen**.

## Szenario "Versicherung kündigen"

Der **Versicherte** *kündigt* die Versicherung. Der Sachbearbeiter *löst den Vertrag auf*.



#### 4.10.4 Grundelemente



#### 4.10.5 Pfeilarten



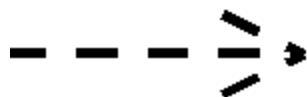
**Synchrone Nachricht**



**Asynchrone Nachricht**



**Asynchrone Nachricht**



**Antwort (Return)**

#### 4.10.6 Nachrichtenformat

(Bedingung) : Nachricht (Parameter)

z.B. [ x > 0 ] : getValue ()



UML2 Seite 24

UML.ppt

Weiter mit UMLTUWien.pdf und UMLREF\_deu.pdf

Notationsverfahren.

UML <http://ivs.cs.uni-magdeburg.de/~dumke/UML/index.htm>

<http://www.oio.de/m/uml-referenz/>

[http://www.sigs-datacom.de/sd/publications//os/1998/02/OBJEKTSpektrum\\_UM\\_kompakt.htm](http://www.sigs-datacom.de/sd/publications//os/1998/02/OBJEKTSpektrum_UM_kompakt.htm)