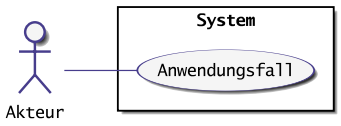
UML Anwendungsfall- (UseCase-) Diagramm mit plantUML



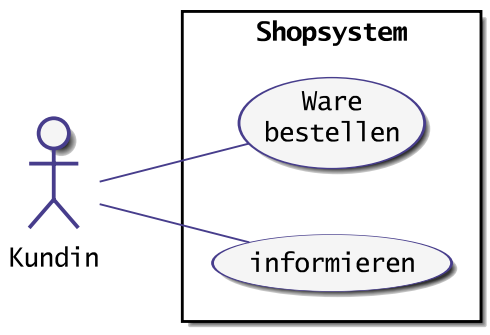
1 Akteurinnen und Anwendungsfälle

Ein Anwendungsfalldiagramm (*UseCase*) beschreibt *wie* ein (Software)-System mit Anwenderinnen[1] interagiert. Es beschreibt, welche Anwendungsfälle ein System anbietet. Die Reihenfolgen oder Abläufe der Anwendungsfälle müssen jedoch auf andere Art modelliert werden.
Anwendungsfalldiagramme helfen v.a. dabei, die Vollständigkeit und Korrektheit des Systemverständnisses der Projektbeteiligten (Kunden, Fachdomäne, Entwicklerinnen) abzugleichen.


```
@startuml 'Muss immer am Anfang stehen
 'Generell zum Lesen von Use-Case-Diagrammen
     einfacher:
 left to right direction

 'Akteurin festlegen
 actor :Kundin: as customer

 ' UseCase definieren (mit Zeilenumbruch \n=Newline)
 usecase (Ware \n bestellen) as bestellen

 customer -- bestellen
@enduml
```

Akteurinnen sind Menschen oder andere Systeme, die Anwendungsfälle des Systems nutzen. Sie können auch Rollen (Kundin) oder Typen zusammenfassen. Sie werden in der Regel als Strichmensch (*stick man*) notiert.
Mit einem **Anwendungsfall (Use Case)** erzeugt das modellierte System erkennbaren Nutzen für die zugeordneten Akteurinnen. Ein Anwendungsfall fasst Aktionen zusammen, die (funktionale) Anforderungen erfüllen. Anwendungsfälle werden in der Regel als Ellipse notiert.
Assoziationen verbinden Anwendungsfälle mit den auslösenden oder benötigten Akteurinnen. Sie werden mit durchgezogenen Linien notiert.

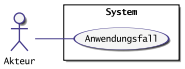
2 Systemgrenzen und Systemname




```
actor :Kundin: as customer

'Definition der Systemgrenze über rectangle{}
rectangle Shopsystem {

   usecase (Ware \n bestellen) as bestellen

   'Kurzform ohne Deklaration des UseCases:
   customer -- (informieren)

   customer -- bestellen
}
```

Die **Systemgrenze** legt fest, welche Anwendungsfälle im modellierten System enthalten sind. Das abgegrenzte System trägt einen Namen und spannt einen Namensraum auf.
Alle **Akteurinnen** stehen außerhalb der Systemgrenzen - andernfalls wären sie als Teil des Systems nicht gesondert

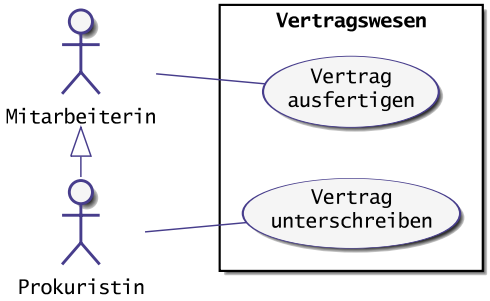
[1] Grundsätzlich: es wird die feminine gramatikalische Form gewählt, maskuline Akteure sind immer #mitgemeint



zu modellieren.
Systemgrenzen müssen nicht zwingend angegeben werden, dienen aber dem Verständnis und der Abgrenzung von Akteurinnen, Anwendungsfällen und externen Systemen.

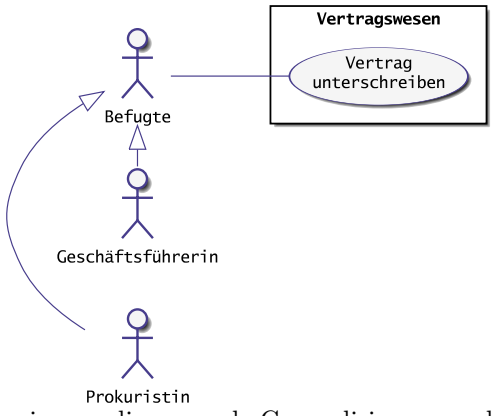
3 Vererbung von Akteuren und "oder"-Beziehungen

Mit Hilfe von Vererbungsbeziehungen können Akteurinnen spezialisiert werden: Im Beispiel unten ist die Prokuristin eine spezielle Mitarbeiterin, der zusätzlich zu allen Anwendungsfällen einer Mitarbeiterin auch noch über eigene Anwendungsfälle verfügt. Vererbung wird - wie in der UML üblich - mit einer geschlossenen Pfeilspitze symbolisiert:



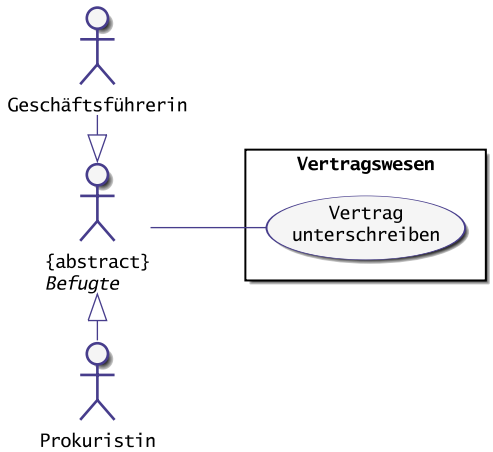

```
actor :Mitarbeiterin: as mitarbeiter
actor :Prokuristin: as prokurist

rectangle Vertragswesen {
   usecase (Vertrag \n ausfertigen) as ausfertigen
   usecase (Vertrag \n unterschreiben) as
       unterschreiben

   'Kurzform ohne Deklaration des UseCases:
   mitarbeiter -- ausfertigen
   prokurist -- unterschreiben
   mitarbeiter <|- prokurist
}
```

Vererbungsbeziehungen werden auch genutzt, um ODER-Beziehungen zu modellieren: Eine Geschäftsführerin oder eine Prokuristin darf Verträge unterschreiben. Solche Zusammenhänge lassen sich nur über einen generalisierten Akteur (hier: Befugte) realisieren.



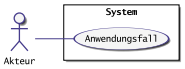

```
actor :Befugte: as befugte
actor :Prokuristin: as prokuristin
actor :Geschäftsführerin: as gf

rectangle Vertragswesen {
   usecase (Vertrag \n unterschreiben) as
       unterschreiben

   befugte -- unterschreiben
   befugte <|- prokuristin
   befugte <|- gf
}
```

Akteurinnen, die zwar als Generalisierung anderer Rollen modelliert werden, die es aber konkret (als Instanz im OOP-Sinne) nie gibt, können als abstrakte Akteurinnen modelliert werden. Ihr Name wird kursiv geschrieben und/oder mit dem *Constraint* {abstract} gekennzeichnet:



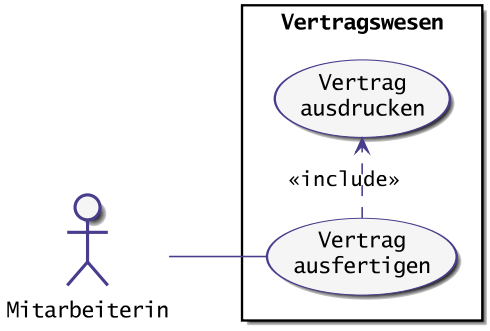

```
actor :{abstract}\n<i>Befugte</i>: as befugte
actor :Prokuristin: as prokuristin
actor :Geschäftsführerin: as gf

rectangle Vertragswesen {
   usecase (Vertrag \n unterschreiben) as
       unterschreiben

   befugte -- unterschreiben
   befugte <|- prokuristin
   gf -|> befugte
}
```

Hier wurde ein Akteur oberhalb und einer unterhalb positioniert, in dem die Pfeilrichtung umgekehrt wurde.



4 Anwendungsfälle, die weitere Anwendungsfälle immer beinhalten

Sofern zur Erfüllung eines Anwendungsfalls in jedem Fall auf die Funktionalität eines zweiten Anwendungsfalls zurückgegriffen werden muss, kann dieser über eine *include*-Beziehung verknüpft werden. Wichtig ist, dass der eingebundene Anwendungsfall auch isoliert einen abgeschlossenen Nutzen generiert (also nicht fester Bestandteil des anderen Anwendungsfalls ist). Diese ”beinhaltet”-Beziehung wird durch eine gestrichelte Linie mit Pfeilspitze dargestellt, die in Richtung des einbezogenen Anwendungsfalls zeigt und die mit dem Stereotyp «include»versehen wird. Der Pfeil kann als ”beinhaltet” in Pfeilrichtung gelesen werden.

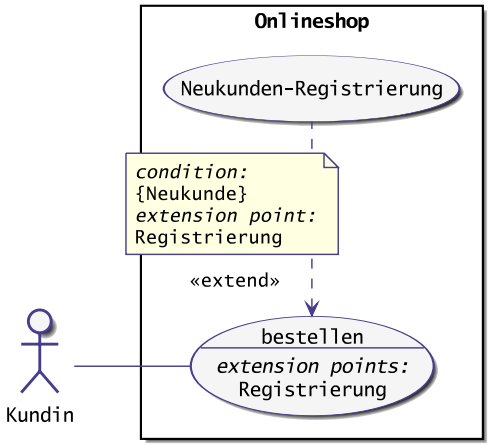



```
actor :Mitarbeiterin: as mitarbeiter

rectangle Vertragswesen {
   usecase (Vertrag \n ausfertigen) as ausfertigen
   usecase (Vertrag \n ausdrucken) as drucken

   mitarbeiter -- ausfertigen

   'Die gestrichelte Linie wird per .> angegeben
   'und das Stereotyp nach dem Doppelpunkt:
   ausfertigen .> drucken : <<include>>
}
```

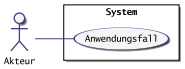
Die Gefahr ist groß über *include*-Beziehungen Programmabläufe und Unterfunktionsaufrufe zu modellieren. Daher bitte immer prüfen: Stellt der inkludierte Anwendungsfall wirklich einen eigenständig auslösbaren Anwendungsfall dar?

5 Anwendungsfälle, unter Umständen durch weitere Anwendungsfälle erweitert werden

Sofern ein Anwendungsfall nur unter bestimmten Umständen um die Funktionalitäten eines zweiten Anwendungsfalls erweitert wird, werden beide über eine *extend*-Beziehung verknüpft. Zu jeder *extend*-Beziehung *sollte* angegeben werden, unter welcher Bedingung (*condition*) welcher Anwendungsfall erweitert wird. Ein gestrichelter Pfeil zeigt vom erweiternden auf den zu erweiternden Anwendungsfall und ist mit dem Stereotyp «include»versehen. An dieser Linie sollte eine Notiz mit *condition* und *extension point* notiert werden. Der *extension point* wird auch am Ursprungs-Anwendungsfall notiert. Der Pfeil kann als ”erweitert” in Pfeilrichtung gelesen werden.

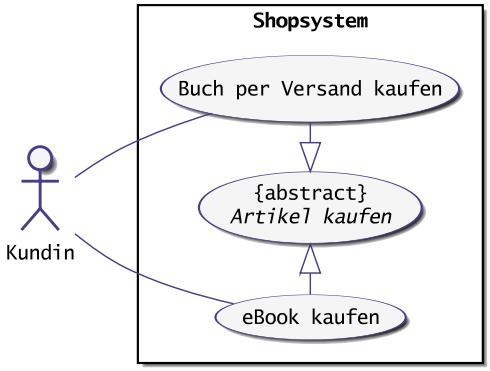



```
actor :Kundin: as customer

rectangle Onlineshop {
   'Angabe des extention point
   usecase bestellen as "bestellen
      --
      <i>extension points:</i>
      Registrierung"

   customer -- bestellen

   'der Stereotyp
   bestellen <.(Neukunden-Registrierung) :
       <<extend>>

   'condition und extension point
   note top on link
      <i>condition:</i>
      {Neukunde}
      <i>extension point:</i>
      Registrierung
   end note
}
```



6 Vererbung von Anwendungsfällen

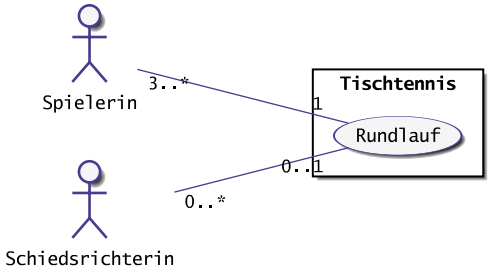
Analog zu Akteurinnen können auch Anwendungsfälle spezialisiert werden. Beispielsweise kann ein generalisierter Anwendungsfall "Artikel kaufen" bestehen aus dem Szenario:
1. Artikel in Warenkorb legen / 2. Warenkorb bestellen / 3. Kauf abwickeln
Die spezialisierten Anwendungsfälle ändern Details in den Szenarien, z.B. bei "Buch per Versand kaufen":
1. Artikel in Warenkorb legen / 2. Warenkorb bestellen / 3a Versandadresse abfragen / 3b Bezahldetails abfragen
oder bei "eBook kaufen":
1. Artikel in Warenkorb legen / 2. Warenkorb bestellen / 3. Paypal-Kaufabwicklung starten / 4. Downloadlink bereitstellen
Auch Anwendungsfälle kennen das konzept der Abstraktion: "Artikel kaufen" selbst kann nicht ausgeführt werden, sondern modelliert nur ein Gerüst, das in konkreten Anwendungsfällen noch ausformulieren müssen.
Die Notation entspricht der für Vererbung (geschlossene Pfeilspitze) und Abstraktion (kursive Schrift, *constraint {abstract}*) bekannten Darstellung.




```
93  actor :backend: as backend
94
95  rectangle Shopsystem {
96     usecase (Bestellung bearbeiten) as bestellung
97     usecase (Online-Bestellung bearbeiten) as
           onlinebestellung
98
99     bestellung <|-onlinebestellung
100 }
```

7 Wie viele Akteurinnen stehen mit wie vielen UseCases in Beziehung?

Um festlegen zu können, wie viele Akteurinnen für Anwendungsfälle nötig sind und an wie vielen Anwendungsfällen Akteurinnen beteiligt sind werden - analog zum UML-Klassendiagramm - Multiplizitäten angegeben, wie am Beispiel zu sehen:
An einem Tischtennis-Rundlauf sind mindestens 3 Spielerinnen beteiligt, jede Spielerin jedoch an exakt einem Rundlauf-Spiel. An einem Rundlaufspiel kann eine Schiedsrichterin beteiligt sein. Jede Schiedsrichterin kann an keinem oder beliebig vielen Rundlaufspielen beteiligt sein:

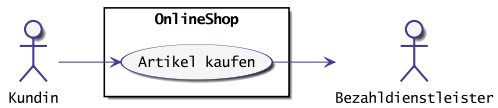



```
101 actor :Spielerin: as player
102 actor :Schiedsrichterin: as referee
103
104 rectangle Tischtennis {
105    usecase (Rundlauf) as rundlauf
106
107    'Multiplizitäten werden an den Assoziationen
           angegeben
108    player "3..*" --- "1" rundlauf
109    referee "0..1" --- "0..*" rundlauf
110 }
```

Da diese Information jedoch häufig für die Adressaten des Use-Case-Diagramms keine Rolle spielt werden Multiplizitäten eher selten notiert.

8 Gerichtete Assoziationen (initiiernde und sekundäre Akteurinnen)

In seltenen Fällen wir die Kommunikationsrichtung in Anwendungsfalldiagrammen dargestellt. So kann verdeutlicht werden, welche Akteurinnen den Anwendungsfall aktiv triggern (primäre Akteurinnen) und wer nur passiv vom Anwendungsfall benötigt wird (sekundäre Akteurinnen).

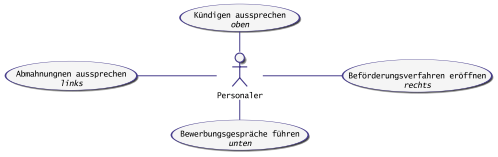



```
actor :Bezahldienstleister: as bezahl
actor :Kundin: as kundin
rectangle OnlineShop {
   usecase (Artikel kaufen) as kauf
}
kundin --> kauf
kauf --> bezahl
```

(Wenn left to right direction gewählt wurde muss der Akteur mit up positioniert werden.)

9 plantUML-Formatierung: Ausrichtung der Linien durch einfache oder zweifache Zeichen (- / – / . / ..)

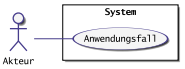
PlantUML bietet die Möglichkeit Assoziationsrichtungen vorzugeben über die Operantoren -up-, -down-, -left-, -down-. Wenn man jedoch die Programme mit der Option ”left to right direction” nutzt sind durch die Drehung sämliche Richtungsanweisungen verkehrt...
Als Alternative kann man auch die Notation mit einfachen und doppelten Bindestrichen wählen:

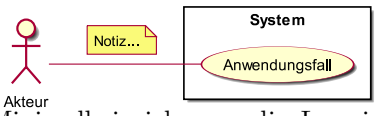



```
'wie immer wurde die Richtung gedreht:
left to right direction

actor :Personaler: as personaler

'nach oben dann mit nur einem Bindestrich:
(Kündigen aussprechen \n <i> oben </i>)- personaler

'nach unten durch vertauschen von Akteurin und
    UseCase
personaler - (Bewerbungsgespräche führen  \n <i>
    unten </i>)

'seitlich nach rechts mit zwei Strichen
personaler -- (Beförderungsverfahren eröffnen  \n
    <i> rechts </i>)

'seitlich nach links mit Vertauschten Positionen
(Abmahnungnen aussprechen \n <i> links </i>) --
    personaler
```

10 plantUML-Formatierung: Aufhübschen von Anwendungsfall-Diagrammen

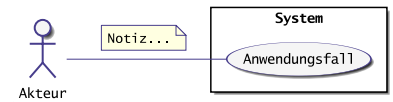
Wenn die Diagramme erstmal stehen will man sie aufhübschen. Dafür stehen allerlei möglichkeiten zur Verfügung, die v.a. auf der plantUML-Seite dargestellt werden. Einige Beispiele sind hier abgebildet:





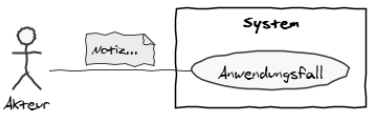
Minimalbeispiel: nur die Leserichtung wurde angepasst:

```
@startuml

left to right direction

actor :Akteur:
rectangle System {
usecase Anwendungsfall
Akteur -- Anwendungsfall
note top on link
Notiz...
end note
}
```



Anpassung von Schriftart und Farben

```
@startuml

skinparam DefaultFontName
    "Lucida Sans Typewriter"
skinparam UseCase{
 BorderColor DarkSlateBlue
 BackgroundColor whitesmoke
}
skinparam Note{
 BorderColor DarkSlateBlue
 BackgroundColor LightYellow
}
skinparam Actor{
 BorderColor DarkSlateBlue
 BackgroundColor whitesmoke
}
skinparam ArrowColor
    DarkSlateBlue
left to right direction

actor :Akteur:
rectangle System {
usecase Anwendungsfall
Akteur -- Anwendungsfall
note top on link
Notiz...
end note
}
@enduml
```



Sieht nach Entwurf aus: um die Vorläufigkeit und Änderbarkeit zu unterstreichen kann man es nach einer Skizze aussehen lassen.

```
@startuml

' Welchs Schriften gibt es
    auf dem System?
' listfonts als
    plantUML-Kommando gibt's
    aus.
skinparam DefaultFontName
    "FG Virgil"
skinparam handwritten true
skinparam monochrome true
skinparam packageStyle rect
skinparam shadowing false

left to right direction

actor :Akteur:
rectangle System {
usecase Anwendungsfall
Akteur -- Anwendungsfall
note top on link
Notiz...
end note
}
@enduml
```

References

[plantUML] Projektwebsite., Dokumentation
https://www.plantuml.com/

[plantText] Projektwebsite., Website, auf der direkt plantUML-QUelltexte geparst werden können:
https://www.planttext.com/