Vorlesung "Modellierung"
Wintersemester 2014/15

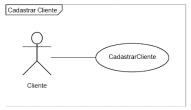
UML
(Folien teilw. von Prof. B. König)

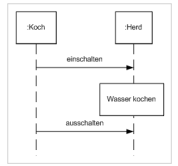
Prof. Norbert Fuhr

UML: Einführung

UML = Unified Modeling Language

- Standard-Modellierungssprache für Software Engineering.
- Basiert auf objekt-orientierten Konzepten.
- Sehr umfangreich, enthält viele verschiedene Typen von Modellen.
- Entwickelt von Grady Booch, James Rumbaugh, Ivar Jacobson (1997).





UML und Objekt-Orientierung

Was bedeutet Objekt-Orientierung?

Grundidee

Die reale Welt besteht aus Objekten, die untereinander in Beziehungen stehen. Diese Sichtweise wird auch auf Modellierung und Softwareentwicklung übertragen.

Etwas genauer ...

Daten (= Attribute) werden zusammen mit der Funktionalität (= Methoden) in Objekten organisiert bzw. gekapselt. Jedes Objekt ist in der Lage Nachrichten (= Methodenaufrufe) zu empfangen, Daten zu verarbeiten und Nachrichten zu senden.

Diese Objekte können – einmal erstellt – in verschiedenen Kontexten wiederverwendet werden.

UML und Objekt-Orientierung

Geschichte der Objekt-Orientierung

- Entwicklung von objekt-orientierten Programmiersprachen:
  - 60er Jahre: Simula (zur Beschreibung und Simulation von komplexen Mensch-Maschine-Interaktionen)
  - 80er Jahre: C++
  - 90er Jahre: Java
- Verbreitung von objekt-orientierten Entwurfsmethoden:
  - 70er Jahre: Entity-Relationship-Modell
  - 90er Jahre: Vorläufer von UML:
    OOSE (Object-Oriented Software Engineering),
    OMT (Object Modeling Technique)
  - Seit 1997: UML
  - Seit 2005: UML 2.0

Vorteile der objekt-orientierten Programmierung und Modellierung

- Leichte Wiederverwendbarkeit dadurch, dass Daten und Funktionalität zusammen verwaltet werden und es Konzepte zur Modifikation von Code (Stichwort: Vererbung) gibt.
- Nähe zur realen Welt: viele Dinge der realen Welt können als Objekte modelliert werden.
- Verträglichkeit mit Nebenläufigkeit und Parallelität: Kontrollfluss kann nebenläufig in den Objekten ablaufen und die Objekte können durch Nachrichtenaustausch bzw. Methodenaufrufe untereinander kommunizieren.

Ein Beispiel für die Modellierung von Objekten der realen Welt:

Fahrkartenautomat

- Daten: Fahrtziele, Zoneneinteilung, Fahrtkosten
- Funktionalität: Tasten drücken, Preise anzeigen, Münzen einwerfen, Fahrkarte auswerfen

Konzepte

- Klassen: definiert einen Typ von Objekten mit bestimmten Daten und bestimmter Funktionalität.

  Beispiel: die Klasse der VRR-Fahrkartenautomaten
- Objekte: Instanzen der Klasse.

  Beispiel: der Fahrkartenautomat am Duisburger Hauptbahnhof, Osteingang

Einsatzgebiete von UML

- Visualisierung
- Spezifikation
- Konstruktion (z.B. zur Codegenerierung)
- Dokumentation

Vokabular der UML (nach Booch/Rumbaugh/Jacobson)

- Dinge (things)
- Beziehungen (relationships)
- Diagramme (diagrams)

Dinge

- Strukturen (structural things)
- Verhalten (behavioral things)
- Gruppen (grouping things)
- Annotationen (annotational things)

Beziehungen

- Abhängigkeiten (dependencies)
- Assoziationen (associations)
- Generalisierungen (generalizations)
- Realisierungen (realizations)

UML-Diagramme

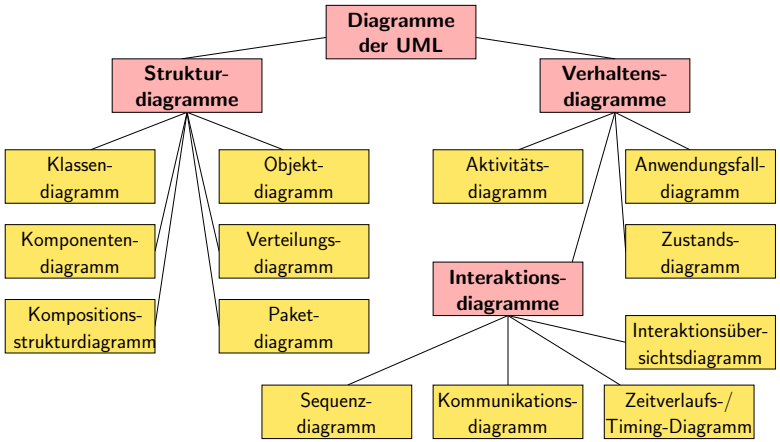


Diagramme (I)

- Strukturdiagramme
  - Klassendiagramme (class diagrams)
  - Objektdiagramme (object diagrams)
  - Kompositionsstrukturdiagramme (composite structure diagram)
  - Paketdiagramme (package diagram)
  - Verteilungsdiagramme (deployment diagram)
  - Komponentendiagramme (component diagrams)

UML: Einführung

Diagramme (II)

- Verhaltensdiagramme
  - Aktivitätsdiagramme (activity diagrams)
  - Zustandsdiagramme (state diagrams)
  - Anwendungsfalldiagramm (use case diagrams)
  - Interaktionsdiagramme
    - Sequenzdiagramme (sequence diagrams)
    - Kommunikationsdiagramm (communication diagram)
    - Zeitverlaufsdiagramm (timing diagram)
    - Interaktionsübersichtsdiagramm (interaction overview diagram)

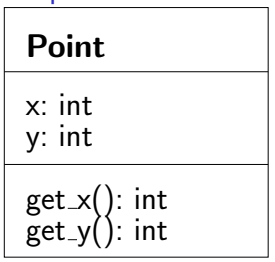
Wir werden im Folgenden einige dieser Begriffe mit Leben füllen.

Klassen- und Objektdiagramme

Wir beginnen mit Klassen- und Objektdiagrammen ...

Beispiel: Klasse der zweidimensionalen Punkte mit x-, y-Koordinaten und Operationen zum Auslesen der Koordinaten

Graphische Darstellung einer Klasse



Klassenname

Attribute (evtl. mit Typ)

Operationen/Methoden

Klassen- und Objektdiagramme

Bemerkungen:

- Bei den Attributen handelt es sich um sogenannte Instanzattribute, d.h., sie gehören zu den Instanzen einer Klasse (nicht zur Klasse selbst).
- Man kann die Sichtbarkeit eines Attributes bzw. einer Methode spezifieren, indem man + (öffentlich/sichtbar) oder − (privat/nicht sichtbar) vor den Attribut-/Methodennamen schreibt.
  Auch die Angabe # (geschützt/protected) ist möglich. In diesem Fall ist das Attribut nur für Klassen sichtbar, die von der entsprechenden Klasse erben.

Klassen- und Objektdiagramme

Bemerkungen:

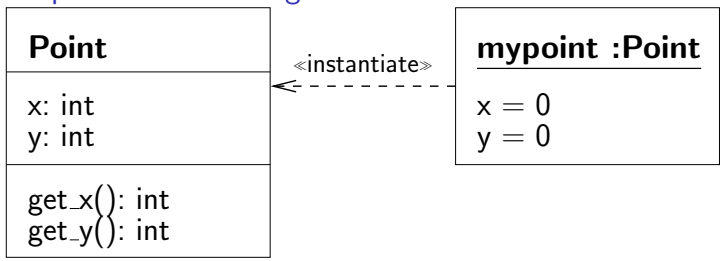
- Attribute haben im allgemeinen Typen, manchmal auch Vorgabewerte (= initiale Werte). Dies wird dann folgendermaßen notiert:

  x: int = 0

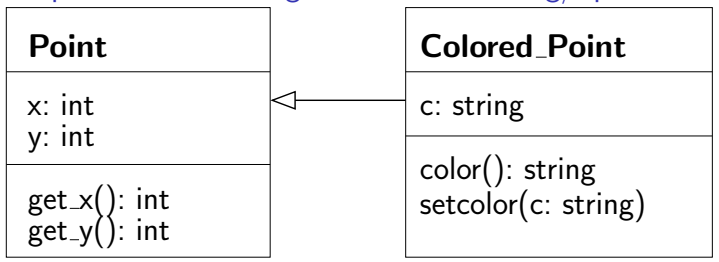
- Mehrstellige Operationen mit Rückgabewert, werden mit ihren Typen folgendermaßen notiert:

  add(m: int, n: int): int

Graphische Darstellung einer Instanz einer Klasse



Graphische Darstellung von Generalisierung/Spezialisierung



Die Klasse **Colored_Point** spezialisiert die Klasse **Point**.
Umgekehrt generalisiert **Point** die Klasse **Colored_Point**.

Bemerkungen:

- In einer solchen Situation nennt man **Point** Superklasse und **Colored_Point** Subklasse.
- Da die Subklasse die Attribute, Methoden und Assoziationen der Superklasse erbt (und diesen noch weitere hinzufügen kann), spricht man auch von Vererbung. Außerdem kann man in der Subklasse Methoden der Superklasse überschreiben und durch neue ersetzen.

Wie kann man Beziehungen zwischen Klassen in UML darstellen?

Es gibt folgende mögliche Beziehungen:

- Assoziation
- Aggregation
- Komposition

Wir beginnen mit der schwächsten Relation zwischen zwei Klassen: der Assoziation.

Assoziation

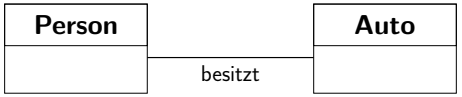
Es gibt eine Assoziation zwischen den Klassen **A** und **B**, wenn

es eine semantische Beziehung zwischen den Klassen gibt.

Üblicherweise werden Assoziationen durch Referenzen realisiert (d.h., eine der Klassen hat ein Attribut vom Typ der anderen Klasse).

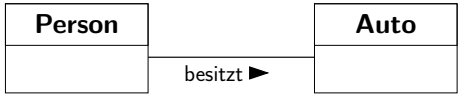
Beispiel für eine Assoziation

Eine Person besitzt ein Auto.

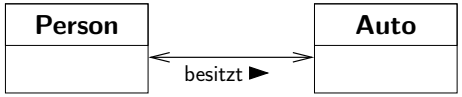


Bemerkungen:

- Es kann eine Leserichtung der Assoziation eingeführt werden:

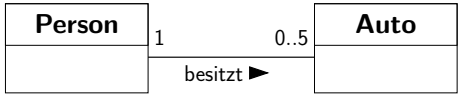


- Die Navigationsrichtung (beschrieben durch eine Pfeilspitze) kann von der Leserichtung abweichen. Sie beschreibt, welche Klasse ihren Assoziationspartner kennt (und daher seine Methoden aufrufen kann).



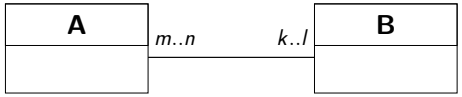
Hier kennen sich beide Klassen gegenseitig.

- Multiplizitäten: an beide Enden der Assoziationen können Multiplizitäten in Form von Intervallen $m..n$ (oder einfach nur m für $m..m$) angegeben werden. Hier besitzt eine Person bis zu fünf Autos. Ein Auto ist im Besitz genau einer Person.



Falls die Multiplizität größer als eins ist, muss dies in der Implementierung durch eine Liste (oder Menge oder Array) von Referenzen realisiert werden.

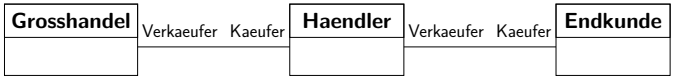
- Multiplizitäten allgemein: in folgendem Diagramm können i Instanzen von **A** (mit $m \leq i \leq n$) mit einer Instanz von **B** assoziert sein. Umgekehrt können j Instanzen von **B** (mit $k \leq j \leq l$) mit einer Instanz von **A** assoziiert sein.



Falls es keine obere Schranke gibt, wird ein Stern (= unendlich) verwendet. Beispielsweise steht $2..*$ für "mindestens zwei".

Die Multiplizität $0..*$ wird als Standardwert angenommen, wenn keine Angabe vorhanden ist.

- Rollen: Klassen können in verschiedenen Assoziationen verschiedene Rollen spielen. Rollen werden auch an den Assoziationen notiert (und können alle Bestandteile eines Attributs enthalten).



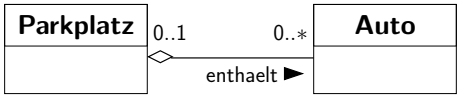
Die nächste Relation – Aggregation – ist etwas stärker.

Aggregation

Es gibt eine Aggregation zwischen den Klassen **A** und **B**, wenn Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten. (Ein "Ganzes" enthält mehrere "Teile".)

Beispiel für eine Aggregation

Ein Parkplatz "enthält" mehrere Autos.



Die stärkste Relation ist die sogenannte Komposition.

Komposition

Es gibt eine Komposition zwischen den Klassen **A** und **B**, wenn

> Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten *und* die Lebenszeit der Teile wird vom "Ganzen" kontrolliert. Das heißt, die Teile können (müssen) gelöscht werden, sobald die Instanz der Klasse **A** gelöscht wird.

Beispiel für eine Komposition

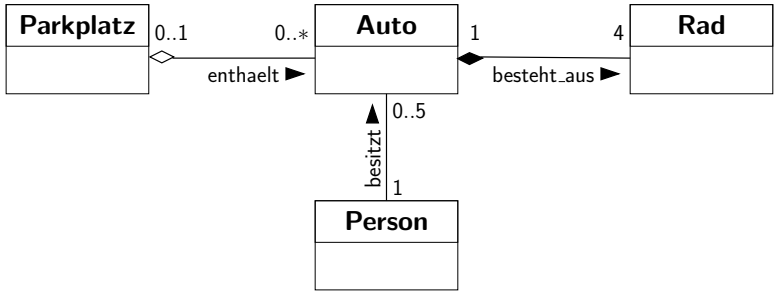
Ein Auto besteht aus vier Rädern.

Die Räder werden zerstört, sobald das Auto zerstört wird.

Bemerkung: In diesem Fall muss die Multiplizität, die an der schwarzen Raute steht, immer 0 oder 1 sein. Jedes Teil kann höchstens zu einem Ganzen gehören.

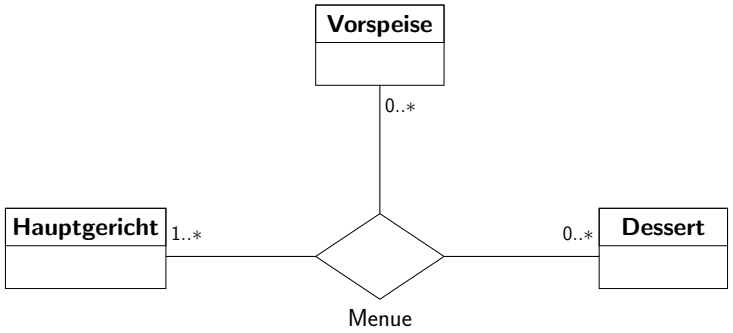
In Klassendiagrammen befinden sich normalerweise nicht nur zwei Klassen mit einer Assoziationen, sondern verschiedene Klassen eines Programms oder Moduls, mit ihren Beziehungen untereinander.

Beispiel:



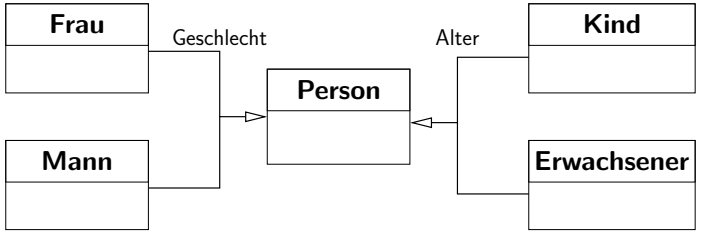
n-äre Assoziation

Neben binären (zweistelligen) Assoziationen gibt es auch *n*-äre Assoziationen, die eine Beziehung zwischen *n* Klassen beschreiben.



Generalisierungsgruppen

Klassen können in unterschiedlicher Weise spezialisiert bzw. unterteilt werden. Daher können Generalisierungen zu Gruppen zusammengefasst werden.



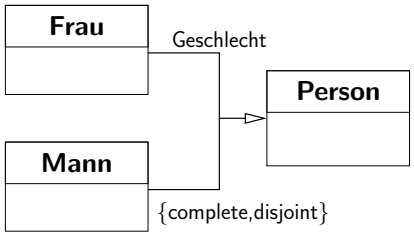
Dabei wird die jeweilige Generalisierungsgruppe (hier: Alter bzw. Geschlecht) im Diagramm angegeben.

Den Generalisierungsgruppen können Eigenschaften (in geschweiften Klammern) zugeordnet werden.

- complete/incomplete:
  - complete: die Generalisierungsgruppe ist vollständig, d.h., sie überdeckt alle Instanzen der Klasse.
  - incomplete: die Generalisierungsgruppe ist unvollständig.
- disjoint/overlapping:
  - disjoint: die spezialisierenden Klassen besitzen keine gemeinsamen Instanzen (keine Überlappung).
  - overlapping: die spezialisierenden Klassen können gemeinsame Instanzen besitzen.

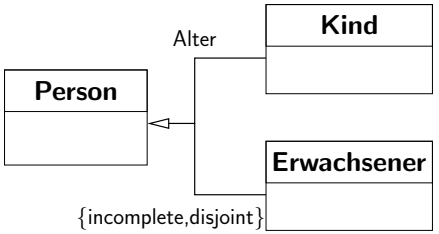
Beispiele für die Eigenschaften complete/incomplete und disjoint/overlapping:

{complete,disjoint}



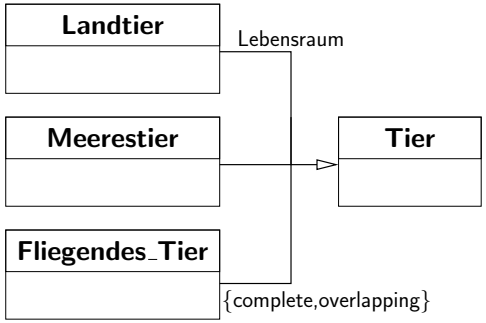
Hier handelt es sich um eine Partitionierung der Instanzen der Klasse.

{incomplete,disjoint}



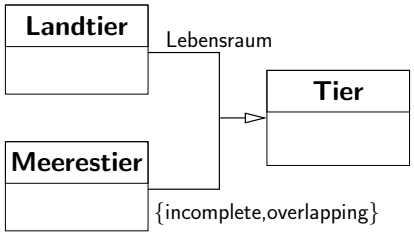
Warum unvollständig? $\rightsquigarrow$ es fehlt eine Klasse Jugendlicher.

{complete,overlapping}



Schildkröten sind sowohl Land- als auch Meerestiere.

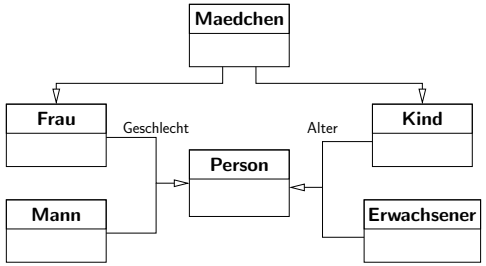
{incomplete,overlapping}



Fliegende Tiere fehlen.

Mehrfachvererbung

Es ist auch möglich, dass eine Klasse von mehreren Klassen erbt, d.h., eine Spezialisierung verschiedener Klassen ist. Dies bezeichnet man als Mehrfachvererbung.



Ein Mädchen ist sowohl ein Kind als auch ein weiblicher Mensch (= Frau).

Basierend auf diesen graphischen Notation sollte man ein objekt-orientiertes System modellieren, bevor es implementiert wird. Dabei stellen sich insbesondere folgende Fragen:

- Welche Objekte und Klassen werden benötigt?
- Welche Merkmale haben diese Klassen und welche Beziehungen bestehen zwischen Ihnen?
- Wie sollen die Klassen eingesetzt werden?
- Welche Methoden stellen diese Klassen zur Verfügung? Wie wirken diese Methoden zusammen?
- In welchen Zuständen können sich Objekte befinden und welche Nachrichten werden wann an andere Objekte geschickt?

Beispiel: Modellierung einer Bank

Ein größeres Beispiel für objekt-orientierte Modellierung und Programmierung: Wir modellieren eine Bank.

Folgende Anforderungen werden gestellt:

- Eine Bank
  - hat mehrere Kunden
  - und mehrere Angestellte
  - und führt eine Menge von Konten.
- Konten können
  - Giro- oder Sparkonten sein. (Ein Sparkonto wirft höhere Zinsen ab, darf aber nicht ins Minus absinken.)
  - in Euro oder in Dollar geführt werden.

Beispiel: Modellierung einer Bank

- Auf den Konten sollen folgende Operationen ausgeführt werden können:
  - Einzahlen
  - Abheben
  - Verzinsen
  - Umbuchen
- Außerdem sollen alle Objekte (inklusive der Bank) ihre Darstellung ausdrucken können. Dazu hat jedes Objekt eine eigene print-Methode.

  Das bezeichnet man auch als Overloading: eine Methode gleichen Namens kann bei Objekten verschiedener Klassen aufgerufen werden und erzielt dort unterschiedliche Effekte.

Beispiel: Modellierung einer Bank

Klasse Bank:

Bank
name: string
neuen_kunden_anlegen() angestellten_einstellen() konten_verzinsen() print()

Methoden: neuen Kunden anlegen; neuen Angestellten einstellen; alle Konten verzinsen

Außerdem: Eine Bank besteht (in Kompositionsrelation) aus einer Menge von Konten, die verschwinden, wenn die Bank verschwindet. Außerdem gibt es Aggregationen mit einer Liste von Kunden und von Angestellten.

Beispiel: Modellierung einer Bank

Klasse Konto:

Konto
nummer: string zins: float kontostand: Betrag
einzahlen(wert: Betrag) abheben(wert: Betrag) umbuchen_auf(kto: konto, wert: Betrag) print() verzinsen()

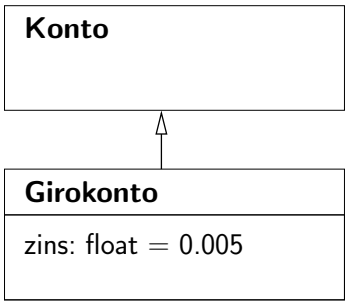
Attribute: Kontonummer, Zins

Methoden: Kontostand abfragen, einzahlen, abheben, umbuchen eines Betrags auf ein anderes Konto, Konto verzinsen

Außerdem: Konto steht in einer Kompositionsrelation mit Betrag, d.h., jedes Konto enthält einen Betrag (siehe Klassendiagramm).

Beispiel: Modellierung einer Bank

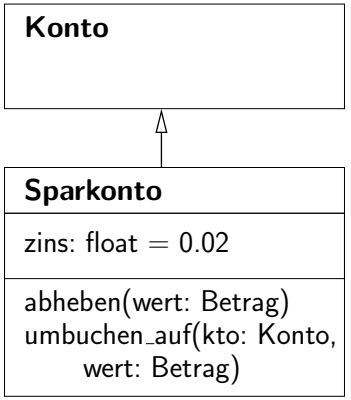
Klasse Girokonto:



Die Klasse Girokonto wird von Konto abgeleitet. Typischerweise ist der Zins bei Girokonten niedriger als bei Sparkonten. Von daher wird dieser auf einen niedrigeren Anfangswert gesetzt.

Beispiel: Modellierung einer Bank

Klasse Sparkonto:

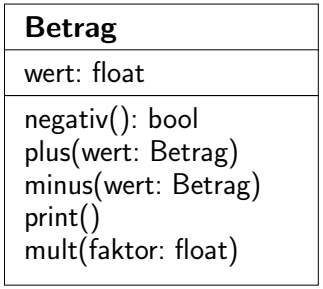


Bei der Klasse Sparkonto muss – wie beim Girokonto – ein neuer Anfangswert für den Zins gesetzt werden.

Außerdem: es muss darauf geachtet werden, dass das Konto nicht ins Minus abgleitet. Dazu werden die entsprechenden Methoden überschrieben (in der Implementierung muss die Bedingung entsprechend getestet werden).

Beispiel: Modellierung einer Bank

Klasse Betrag:

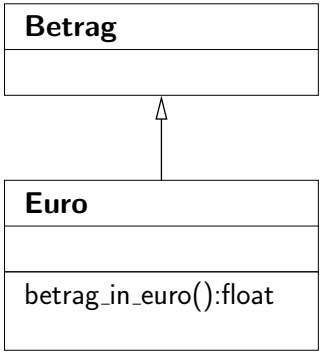


Methoden: Test, ob Konto im Minus; Betrag addieren, subtrahieren; Multiplikation mit einer Gleitpunktzahl (zum Verzinsen!)

Beträge müssen im allgemeinen in einer Währung angegeben werden. Daher werden von der Klasse Betrag die Unterklassen Euro und Dollar abgeleitet.

Beispiel: Modellierung einer Bank

Klasse Euro:

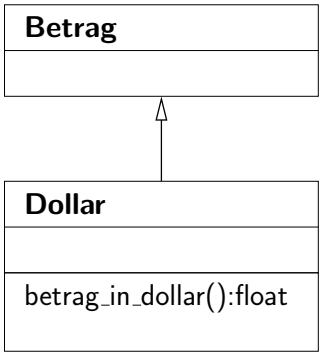


Von Betrag wird zunächst die Klasse Euro abgeleitet.

Methoden: Betrag in Euro ausgeben

Beispiel: Modellierung einer Bank

Klasse Dollar:

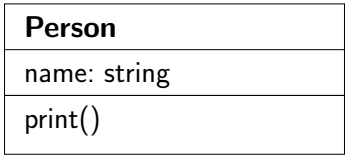


Von Betrag wird außerdem die Klasse Dollar abgeleitet.

Methoden: Betrag in Dollar ausgeben

Beispiel: Modellierung einer Bank

Klasse Person:

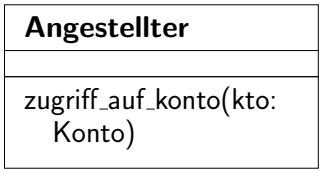


Methoden: nur die print-Methode, weitere Methoden werden in den Unterklassen definiert

Außerdem: Person steht in einer Assoziationsrelation mit einer Liste von Konten, die entweder dieser Person gehören (Kunde) oder auf die diese Person Zugriff hat (Angestellte/r).

Beispiel: Modellierung einer Bank

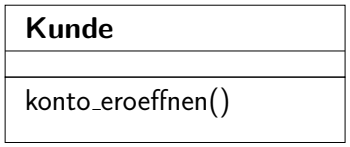
Klasse Angestellter (erbt von Person):



Methoden: Zugriff auf ein Konto erlangen

Beispiel: Modellierung einer Bank

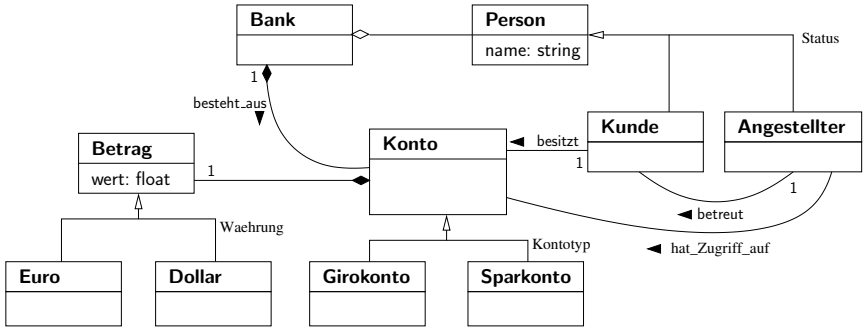
Klasse Kunde (erbt von Person):



Methoden: Konto eröffnen (hier könnte noch ein Parameter übergeben werden, der beschreibt, ob das Konto ein Giro- oder Sparkonto sein soll und ob es in Euro oder Dollar geführt werden soll)

Außerdem: Ein Kunde steht in einer Assoziationsrelation mit seinem Betreuer.

Beispiel: Modellierung einer Bank

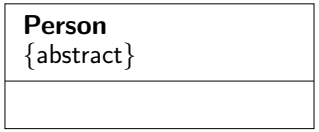


Klassen- und Objektdiagramme

Abstrakte Klasse

Eine Klasse heißt abstrakt, wenn sie selbst keine Instanzen haben kann. Dazu wird die Eigenschaft {abstract} unter dem Klassennamen angegeben. Manchmal wird stattdessen auch der Klassenname *kursiv* geschrieben.

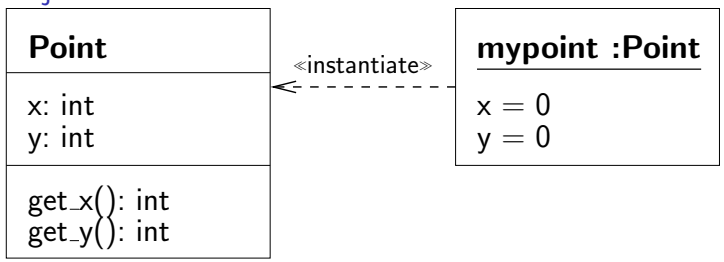
Beispiel: in dem Bank-Beispiel möchte man beispielsweise nie eine Instanz der Klasse Person bilden, sondern nur von Kunde oder Angestellter.



Klassen- und Objektdiagramme

Wir betrachten nun Objektdiagramme. Ein Objekt ist eine Instanz oder Ausprägung einer Klasse.

Objekte und Klassen



Ein Objektdiagramm beschreibt eine Art Momentaufnahme des Systems: eine Menge von Objekten, wie sie zu einem bestimmten Zeitpunkt vorhanden sind.

Klassen- und Objektdiagramme

Bemerkungen:

- Ein Objekt kann, muss aber keinen Namen haben. Es muss aber die Klasse angegeben werden, von der dieses Objekt eine Instanz ist. (In der Form: :Point.)
- Die Klassen müssen in dem Diagramm nicht graphisch dargestellt werden. Es kann aber manchmal angemessen sein, "Bauplan" und "Produkt" gemeinsam darzustellen.
- Nicht alle Attributbelegungen des Objekts müssen angegeben werden.

Links

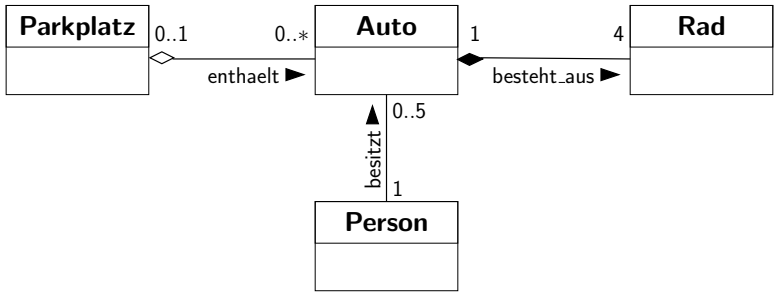
Ein Link beschreibt eine Beziehung zwischen zwei Objekten. Er ist eine Instanz einer Assoziation auf Klassenebene.

Bemerkungen:

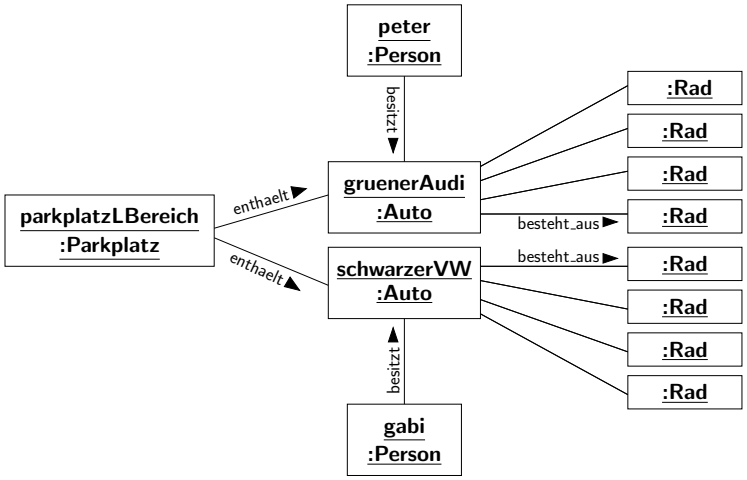
- Links sind *nicht* mit Multiplizitäten beschriftet, ein Link repräsentiert genau eine Beziehung.
- Es ist jedoch darauf zu achten, dass die Multiplizitätsbedingungen des Klassendiagramms eingehalten werden. D.h., die Anzahl der Objekte, die miteinander in Beziehung stehen, müssen innerhalb der jeweiligen Schranken sein.

Zurück zum Beispiel: Autos, Parkplatz, Räder

Wir betrachten zunächst noch einmal das Klassendiagramm:



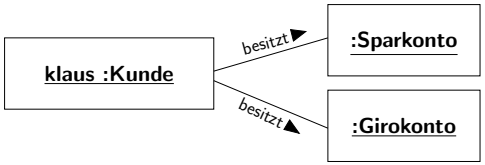
Dieses Objektdiagramm passt zum vorherigen Klassendiagramm:



Bemerkungen:

- Auch Aggregations- und Kompositionssymbole dürfen in Objektdiagrammen auftauchen.
- Achtung: Beziehungen (Assoziationen, Aggregationen, Kompositionen) zwischen Klassen werden vererbt und müssen dann auch entsprechend im Objektdiagramm bei den Instanzen der Unterklassen auftauchen.

Beispiel:



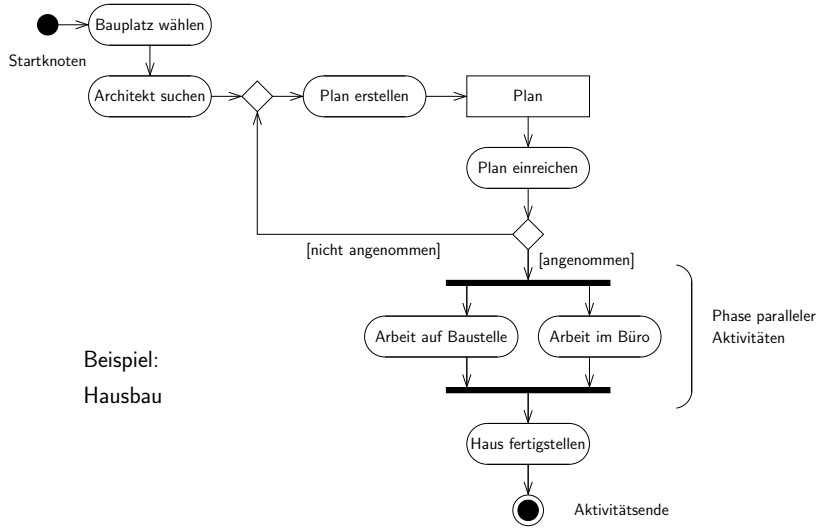
Aktivitätsdiagramme

Wir betrachten nun sogenannte Aktivitätsdiagramme (activity diagrams), das sind UML-Diagramme mit denen man Ablaufpläne, Reihenfolgen von Aktivitäten, parallele Aktivitäten, etc. modellieren kann.

Sie werden beispielsweise verwendet, um Geschäftsprozesse (auch Workflow-Prozesse) des Auftraggebers zu modellieren. Sie können ebenso eingesetzt werden, um interne Systemprozesse zu beschreiben.

Aktivitätsdiagramme sind in vielen Aspekten ähnlich zu Petri-Netzen. Im Unterschied zu Petri-Netzen bieten sie zusätzliche Modellierungsmöglichkeiten, haben jedoch keine formale Semantik.

Aktivitätsdiagramme



Aktivitätsdiagramme

Wir vergleichen nun die Bestandteile von Aktivitätsdiagrammen mit Petrinetzen (angelehnt an die Semantik von Störrle):

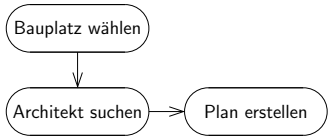
Aktion

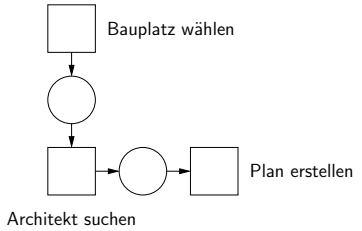
Eine Aktion wird durch ein Rechteck mit abgerundeten Ecken dargestellt. Es entspricht einer Transition eines Petrinetzes.

Aktivitätsdiagramme

Kontrollfluss

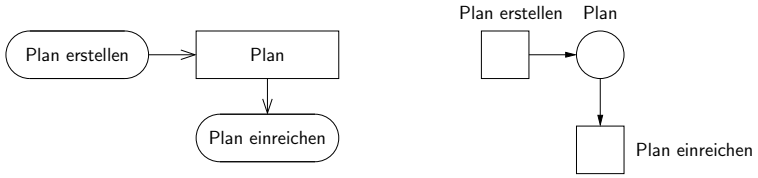
Der Kontrollfluss, d.h. die Kanten, zwischen den Aktionen wird in dem entsprechenden Petrinetz durch Hilfsstellen dargestellt.





Objektknoten

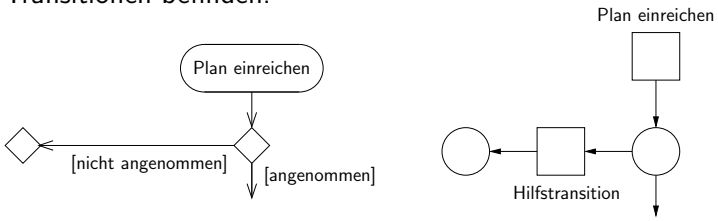
Objektknoten beschreiben Speicher für die Übergabe von Objekten bzw. Ressourcen. Sie entsprechen den Stellen des Petrinetzes.



Bemerkung: Objektknoten sind mit Klassennamen beschriftet. Sie können mehrere Instanzen dieser Klasse enthalten.

Entscheidungsknoten (auch: Verzweigungsknoten)

Entscheidungsknoten (decision nodes) beschreiben eine Verzweigung des Kontrollflusses, wobei aus den möglichen Kontrollflüssen genau einer ausgewählt wird. Sie werden durch Hilfsstellen repräsentiert, die sich in der Vorbedingung mehrerer Transitionen befinden.



Bei Bedarf (v.a. bei nachfolgenden Entscheidungs- oder Objektknoten) muss noch eine Hilfstransition eingeführt werden.

Bemerkung zu Entscheidungsknoten:

Überwachungsbedingungen (Guards), die den Kontrollfluss steuern, werden in eckigen Klammern an den ausgehenden Kontrollflüssen notiert. Ähnliche Guards existieren auch in attributierten Netzen.

Verbindungsknoten

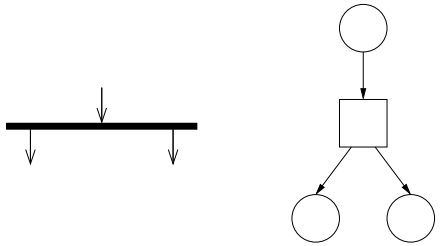
Es gibt auch sogenannte Verbindungsknoten (merge nodes), die mehrere alternative Kontrollflüsse zusammenfassen. Sie können durch Hilfsstellen dargestellt werden, die sich in der Nachbedingung mehrerer Transitionen befinden.

Es gibt auch Knoten, die sowohl Entscheidungs- als auch Verbindungsknoten sind, d.h., mehrere eingehende und mehrere ausgehende Kontrollflüsse haben.

Aktivitätsdiagramme

Gabelung (auch: Parallelisierungsknoten)

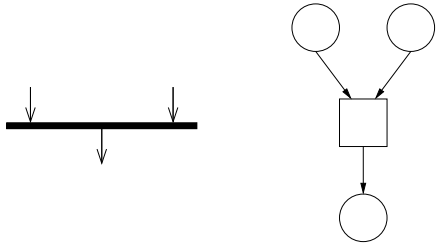
Eine Gabelung (fork node) teilt einen Kontrollfluss in mehrere parallele Kontrollflüsse auf.
Sie entspricht einer Transition mit mehreren Stellen in der Nachbedingung.



Aktivitätsdiagramme

Vereinigung (auch: Synchronisationsknoten)

Analog dazu gibt es die Vereinigung (join node), die mehrere parallele Kontrollflüsse zusammenfasst. Sie wird durch eine Transition dargestellt, die mehrere Stellen in der Vorbedingung hat.



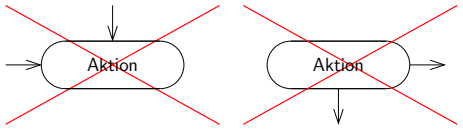
Wie Entscheidungs- und Verbindungsknoten kann man Gabelungen und Vereinigungen zu einem Element zusammenfassen.

Aktivitätsdiagramme

Bitte beachten:

Kontrollflüsse dürfen nur an Objektknoten, Entscheidungsknoten und Gabelungen aufgespalten und an Objektknoten, Verbindungsknoten und Vereinigungen wieder zusammengeführt werden.

Folgendes ist nicht erlaubt:



Aktivitätsdiagramme

Startknoten

Ein Startknoten entspricht *einer* initial markierten Stelle. Ein Aktivitätsdiagramm darf nur einen Startknoten haben, in den kein Kontrollfluss hineinführt

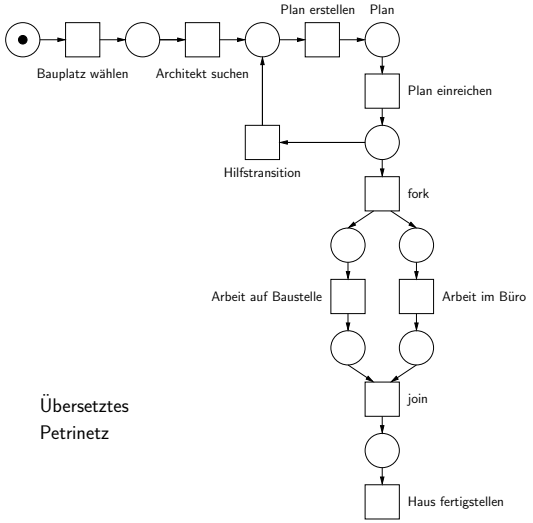
Aktivitätsdiagramme

Aktivitätsende

Das Aktivitätsende signalisiert, dass *alle* Kontrollflüsse beendet werden. Es gibt keine Entsprechung in Petrinetzen.

Es gibt auch ein Symbol für das Flussende, das nur den in es hineinlaufenden Kontrollfluss beendet.

Aktivitätsdiagramme



Übersetztes Petrinetz

Aktivitätsdiagramme

Aktivitätsdiagramme enthalten noch mehr Anleihen aus Petrinetzen. Beispielsweise können auch die Kapazität eines Objektknotens und das Gewicht eines Kontrollflusses spezifiziert werden.

Dabei beschreibt upperBound die Kapazität einer Stelle (maximal 6 Gerichte dürfen gleichzeitig fertig sein) und weight das Gewicht (immer 2 Gerichte werden gleichzeitig serviert).

In Aktivitätsdiagrammen darf noch zusätzlich spezifiziert werden, in welcher Reihenfolge die Objekte aus dem Objektknoten genommen werden (unordered, ordered, LIFO, FIFO).

Aktivitätsdiagramme

Vorsicht:

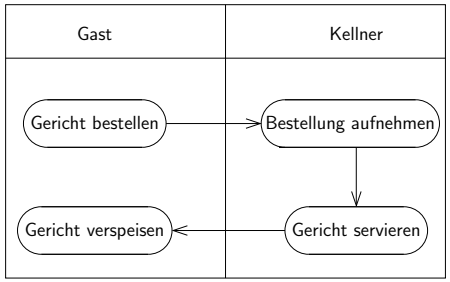
- Die Entsprechung zwischen Aktivitätsdiagrammen und Petrinetzen ist nicht immer ganz exakt. Nicht alle Konzepte entsprechen einander.
- Das liegt teilweise auch daran, dass Aktivitätsdiagramme nur ein semi-formales Modell sind und nicht alle Aspekte vollständig spezifiziert sind.

Aktivitätsdiagramme

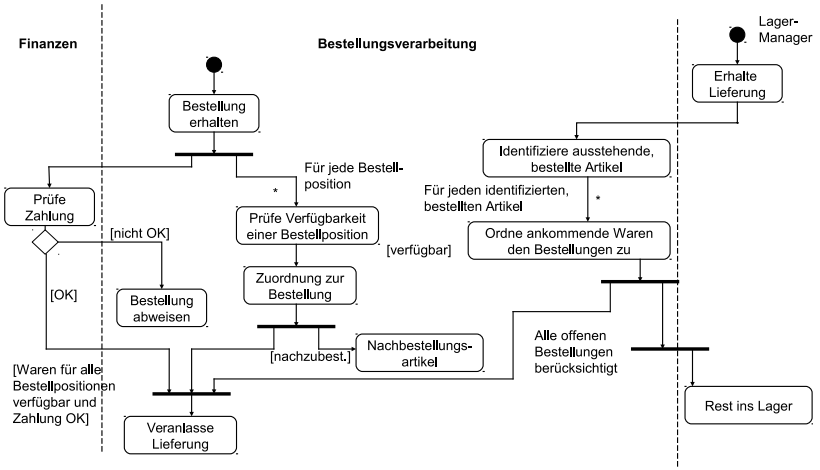
Es gibt auch die Möglichkeit zur weiteren Strukturierung von Aktivitätsdiagrammen:

Aktivitätsbereiche (activity partitions/swimlanes)

Zusammenfassung mehrerer Knoten (Aktionen, Objektknoten, etc.) zu einer Einheit. Dies dient im allgemeinen dazu, um die Verantwortung für bestimmte Aktionen festzulegen.



Aktivitätsdiagramm mit Aktivitätsbereichen



Aktivitätsdiagramme

Weitere Elemente von Aktivitätsdiagrammen:

- Pins: Parameter und Parametersätze für Aktionen
- Senden und Empfang von Signalen
- Kontrollstrukturen: Schleifenknoten, Bedingungsknoten
- Unterbrechungsbereiche (interruptible activity region) zur Behandlung von Ausnahmen (Exceptions)
- Expansionsbereiche (expansion region) zur wiederholten Ausführung von Aktivitäten für mehrere übergebene Objekte

Zustandsdiagramme

Zustandsdiagramme (state diagrams, auch state machine diagrams oder statecharts) genannt, sind eng verwandt mit den bereits eingeführten Zustandsübergangsdiagrammen.

Sie werden eingesetzt, wenn bei der Modellierung der Fokus auf die Zustände und Zustandsübergänge des Systems gelegt werden soll.

Im Gegensatz zu Aktivitätsdiagrammen werden auch weniger die Aktionen des Systems, sondern eher die Reaktionen des Systems auf die Umgebung beschrieben.

Zustandsdiagramme

Anwendungen sind die Modellierung von:

- Protokolle, Komponenten verteilter Systemen
- Benutzeroberflächen
- Eingebettete Systemen
- ...

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt. Harel modellierte damit vollständig seine Armbanduhr.

Zustandsdiagramme

Eigenschaften von Zustandsdiagrammen:

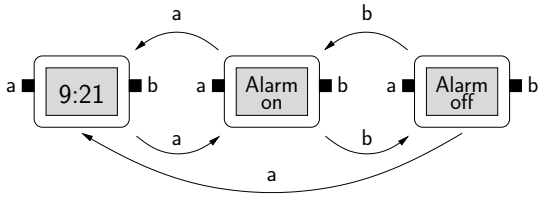
- Zustände und Zustandsübergänge (Transitionen)
- Hierarchisch aufgebaute Zustände
- "Parallelschalten" von Zustandsdiagrammen durch Regionen
- Historien, um sich früher besuchte Zustände zu merken und in diese zurückzukehren

Viele dieser Eigenschaften dienen dazu, unübersichtliche Zustandsdiagramme mit vielen Zuständen und Übergängen übersichtlicher und kompakter zu gestalten.

Zustandsdiagramme

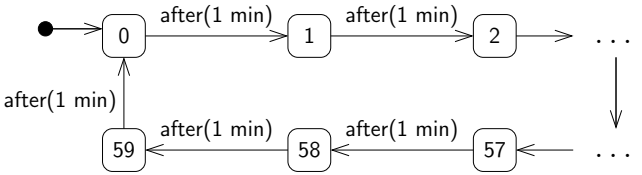
Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber Harels Armbanduhr).

Die Armbanduhr hat zwei Knöpfe (a,b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen diesen Modi wechselt man mit Hilfe von Knopf a.
Der Alarm kann aus (off) oder an (on) sein. Man kann zwischen den Alarmzuständen mit Hilfe von Knopf b wechseln.



Zustandsdiagramme

Wir beginnen zunächst mit der Modellierung der Minutenanzeige.



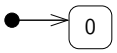
Zustandsdiagramme

Zustand

Ein Zustand eines Zustandsdiagramms wird durch ein Rechteck mit abgerundeten Ecken dargestellt.

Startzustand

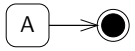
Der Startzustand wird durch einen schwarzen ausgefüllten Kreis gekennzeichnet (ähnlich wie bei Aktivitätsdiagrammen).



Zustandsdiagramme

Endzustand

Endzustände werden wie das Aktivitätsende in Aktivitätsdiagrammen gekennzeichnet.



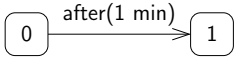
Für den Fall, dass man ein System modelliert, das nicht terminieren soll, gibt es keinen Endzustand (wie in unserem Beispiel).

Zustandsdiagramme

Transition (= Zustandsübergang)

Eine Transition ist ein Pfeil, der mit Ereignis [Bedingung]/Effekt beschriftet ist. (Bedingung und Effekt sind optional.)

- Ereignis: Signal oder Nachricht, die die entsprechende Transition auslösen.



- Bedingung: Überwachungsbedingung (auch Guard genannt).
- Effekt: Effekt, der durch die Transition ausgelöst wird .

Im obigen Beispiel (Minutenanzeige) gibt es nur Ereignisse (sogenannte time events), die die Zeitspanne spezifizieren, nach der die Transition ausgelöst wird. Es gibt aber auch andere Ereignisse, beispielsweise Methodenaufrufe.

Zustandsdiagramme

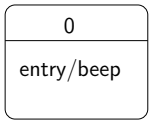
Neben den Effekten, die durch Transitionen ausgelöst werden, können in einem Zustand weitere Aktionen bei Eintritt, Verweilen oder Verlassen ausgelöst werden.

Sie haben den gleichen Aufbau wie die Beschriftung einer Transition: Ereignis [Bedingung]/Effekt. Dabei kann Ereignis unter anderem folgendes sein:

- entry: der entsprechende Effekt wird bei Eintritt in den Zustand ausgelöst.
- do: der Effekt ist eine Aktion, die nach Betreten des Zustands ausgeführt wird und die spätestens dann endet, wenn der Zustand verlassen wird
- exit: der entsprechende Effekt wird bei Verlassen des Zustands ausgelöst.

Zustandsdiagramme

Beispiel: die Uhr soll zu jeder vollen Stunden ein Signal von sich geben. Daher wird die Aktion beep bei Eintritt in den Zustand 0 ausgelöst.



Zustandsdiagramme

Wie sieht es mit der Stundenanzeige aus? Diese soll parallel zur Minutenanzeige laufen, d.h., die Uhr ist in zwei Zuständen gleichzeitig, ein Zustand für die Stunden, der andere für die Minuten. Diese Situation wird durch Regionen modelliert.

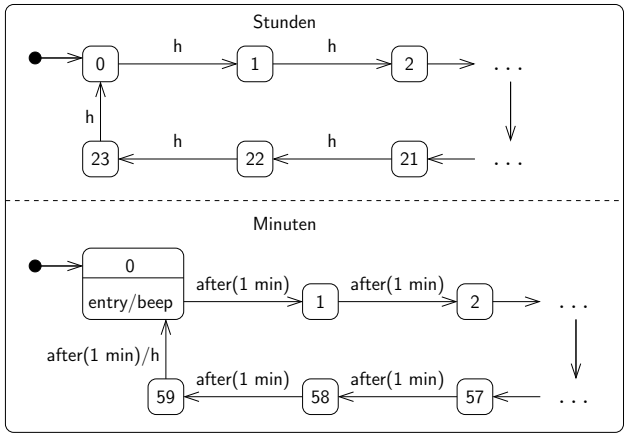
Region

Regionen unterteilen ein Zustandsdiagramm in zwei Bereiche, die parallel zueinander ausgeführt werden.

Dies erlaubt uns, insgesamt nur $24 + 60 = 84$ Zustände, anstatt $24 \cdot 60 = 1440$ Zustände zu zeichnen.

Zustandsdiagramme

So sieht das modifizierte Zustandsdiagramm für die Uhr jetzt aus:



Zustandsdiagramme

Bemerkungen:

- Es gibt jetzt zwei Startzustände, die beide zu Beginn betreten werden (Uhrzeit: 0:00).
- Da Stunden- und Minutenanzeige nicht vollkommen unabhängig voneinander arbeiten, ist eine Synchronisation eingebaut. Die Transition in den Minutenzustand 0 löst einen Effekt h (h für "hour") aus.

  Dieser Effekt ist dann ein Trigger, der das entsprechende Ereignis triggert und den Übergang in den nächsten Stundenzustand verursacht. Die beiden Transitionen werden synchronisiert und finden gleichzeitig statt.

Zustandsdiagramme

Bemerkungen:

- Transitionen verbrauchen im allgemeinen keine Zeit (im Gegensatz zum Aufenthalt in Zuständen).
- Neben Triggern, die direkt ausgeführt werden, gibt es auch Events, die zunächst in einer Event Queue (= Warteschlange) abgelegt werden. Diese Warteschlange wird schrittweise abgearbeitet, wobei die entsprechenden Ereignisse ausgelöst werden. Damit kann asynchrone Kommunikation beschrieben werden.
- Effekte können direkte Kommunikation bedeuten (beispielsweise Methodenaufrufe), können aber auch sogenannte Broadcasts (= Rundrufe) sein. Diese sind überall im Zustandsdiagramm sichtbar. (Die ursprüngliche Statecharts-Semantik von Harel verwendete nur Broadcasts.)

Zustandsdiagramme

Nun soll noch die Möglichkeit hinzugefügt werden, das Alarmsignal zur vollen Stunden aus- und wieder einzuschalten. Dazu führen wir weitere Zustände (Alarm on, off) ein, in die man durch Drücken von *a* gelangt.

Problem: wir brauchen mindestens 84 mit *a* beschriftete Transitionen, die aus den Stunden-/Minutenzuständen ausgehen! Das sind ziemlich viele . . .

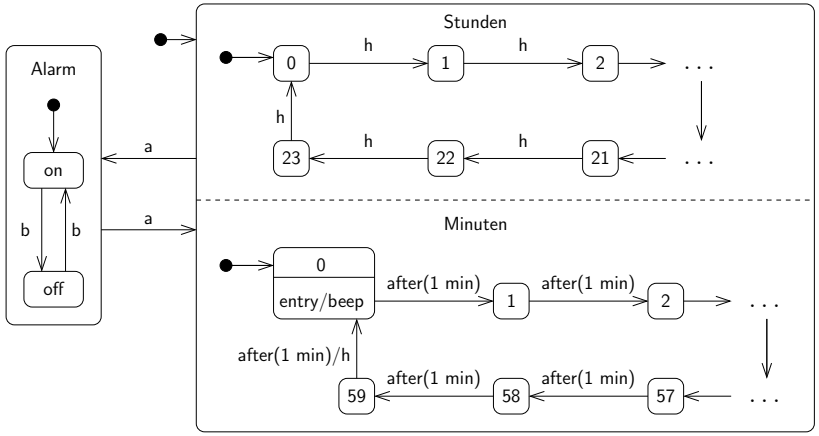
Dafür bieten Zustandsdiagramme folgende Lösung:

Zusammengesetzte Zustände

Zusammengesetzte Zustände dienen dazu, um Hierarchien von Zuständen zu modellieren und damit ein- und ausgehende Transitionen zusammenzufassen.

Zustandsdiagramme

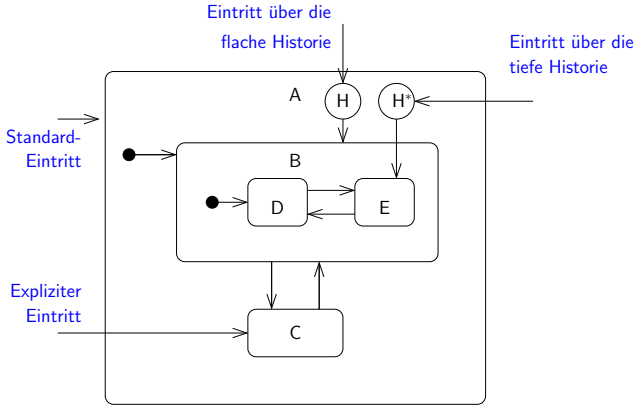
Damit ergibt sich folgendes Diagramm:



Zustandsdiagramme

Um eine gewisse Flexibilität bei der Modellierung zu haben, werden verschiedene Eintritts- und Austrittsmöglichkeiten bereitgestellt.

Eintrittsmöglichkeiten in einen Zustand (graphisch)



Zustandsdiagramme

Beschreibung der Eintrittsmöglichkeiten:

- Standard-Eintritt: dabei wird der Startzustand des zusammengesetzten Zustands angesprungen.

  (Fortsetzung bei B, was schließlich zu einer Fortsetzung bei D führt)
- Expliziter Eintritt: es wird bei dem explizit angegebenen Folgezustand fortgesetzt.

  (Fortsetzung bei C.)

Zustandsdiagramme

Beschreibung der Eintrittsmöglichkeiten (Fortsetzung):

- Eintritt über die tiefe Historie: Wurde der zusammengesetzte Zustand bereits früher besucht, so wird der letzte vor dem Verlassen des Zustands aktive Unterzustand *der tiefstmöglichen Ebene* betreten.

  (Falls also der zusammengesetzte Zustand das letzte Mal von E aus verlassen wurde, so wird jetzt wieder bei E fortgesetzt.)

  Falls man noch niemals zuvor diesen zusammengesetzten Zustand betreten hat, so wird der Zustand betreten, der mit der Kante gekennzeichnet ist, die aus dem H*-Knoten ausgeht.

Zustandsdiagramme

Beschreibung der Eintrittsmöglichkeiten (Fortsetzung):

- Eintritt über die flache Historie: Wurde der zusammengesetzte Zustand bereits früher besucht, so wird der letzte vor dem Verlassen des Zustands aktive Unterzustand *der obersten Ebene* betreten.

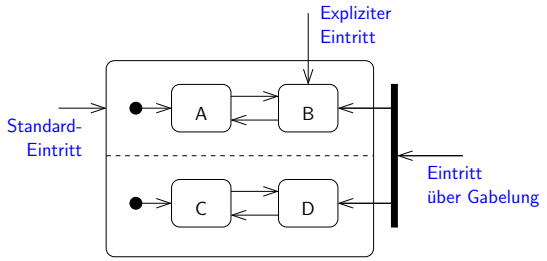
  (Falls also der zusammengesetzte Zustand das letzte Mal von E aus verlassen wurde, so wird jetzt bei B fortgesetzt, was letztendlich zu einer Fortsetzung bei D führt.)

Außerdem: Eintritt über einen Eintrittspunkt (wird hier nicht behandelt).

Zustandsdiagramme

Wenn ein Zustand in mehrere Regionen unterteilt ist, so ergeben sich bei den Eintrittsmöglichkeiten einige Besonderheiten.

Eintrittsmöglichkeiten bei Regionen (graphisch)



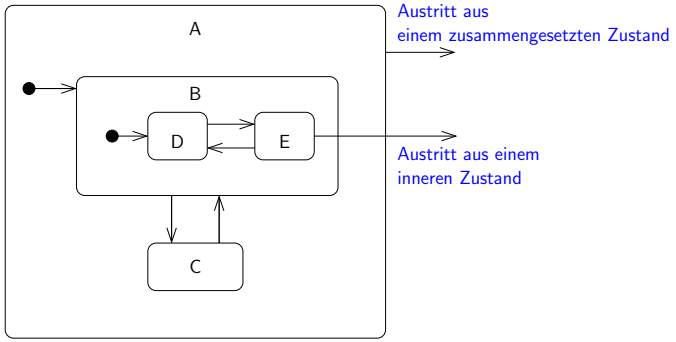
Zustandsdiagramme

Beschreibung der Eintrittsmöglichkeiten bei Regionen:

- Standard-Eintritt: dabei werden die jeweiligen Startzustände der Regionen angesprungen.
  (Fortsetzung bei A und C)
- Expliziter Eintritt: ein Zustand einer Region wird direkt angesprungen. In der anderen Region wird bei dem entsprechenden Startzustand fortgesetzt.
  (Fortsetzung bei B und C.)
- Eintritt über Gabelung: ähnlich wie bei Aktivitätsdiagrammen wird eine Gabelung eingesetzt, um die beiden anzuspringenden Zustände in den Regionen zu kennzeichnen.
  (Fortsetzung bei B und D.)

Zustandsdiagramme

Austrittsmöglichkeiten aus einem Zustand (graphisch)



Zustandsdiagramme

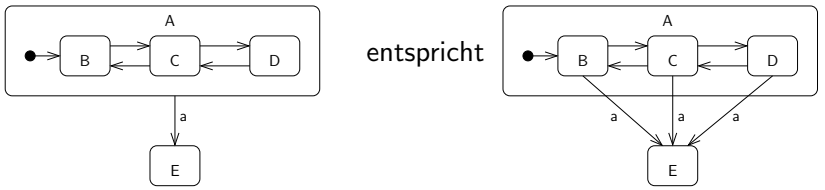
Beschreibung der Austrittsmöglichkeiten:

- Austritt aus einem zusammengesetzten Zustand: sobald das mit der Transition assoziierte Ereignis stattfindet, wird jeder beliebige (Unter-)Zustand von A verlassen.
- Austritt aus einem inneren Zustand: die Transition wird nur genommen, wenn man sich gerade im entsprechenden Zustand (hier: Zustand E) befindet (und das entsprechende Ereignis stattfindet).

Außerdem: Austritt über einen Austrittspunkt, über einen Endzustand oder Terminator (wird hier nicht behandelt).

Zustandsdiagramme

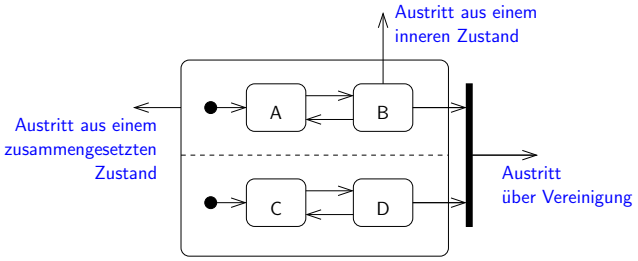
Beispiel zu Austrittsmöglichkeiten:



Zustandsdiagramme

Auch für Austrittsmöglichkeiten müssen wir untersuchen, was sich bei Regionen ändert.

Austrittsmöglichkeiten bei Regionen (graphisch)



Zustandsdiagramme

Beschreibung der Austrittsmöglichkeiten bei Regionen:

- Austritt aus einem zusammengesetzten Zustand: dabei wird der zusammengesetzte Zustand verlassen, egal in welchen Unterzuständen man sich gerade befindet.
- Austritt aus einem inneren Zustand: der zusammengesetzte Zustand wird nur verlassen, wenn man sich in der jeweiligen Region in dem Zustand befindet, der durch den Pfeil verlassen wird. In den anderen Regionen kann man sich in beliebigen Zuständen befinden.

  (Austritt nur, wenn man sich in der ersten Region in B befindet.)

Zustandsdiagramme

Beschreibung der Austrittsmöglichkeiten bei Regionen (Fortsetzung):

- Austritt über Vereinigung: ähnlich wie bei Aktivitätsdiagrammen wird eine Vereinigung eingesetzt, um mehrere Regionen zusammenzuführen. Der zusammengesetzte Zustand kann nur verlassen werden, wenn man sich in den Zuständen befindet, von denen Pfeile in die Vereinigung hineinführen.

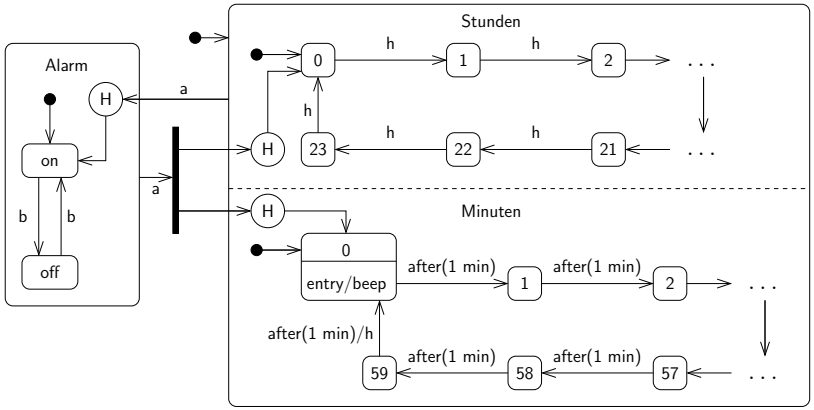
  (Austritt nur, wenn man sich in B und D befindet.)

Zustandsdiagramme

Wieder zurück zur Arbanduhr. Es gibt ein weiteres Problem: wenn man aus der Alarmeinstellung zurückkehrt, ist die Zeit auf 0:00 zurückgesetzt!

Lösung: Verwendung des Eintritts über die (flache) Historie.
Da man im Fall der Zeitanzeige in zwei Regionen gleichzeitig eintreten muss, verwenden wir eine Gabelung.

Zustandsdiagramme



Zustandsdiagramme

Weiteres Problem: wenn man längere Zeit im Alarm-Zustand verbringt, so wird in dieser Zeit die Minuten-/Stundenanzeige nicht entsprechend aktualisiert. Das Time Event (after(1 min)) bezieht sich nur auf die Zeit, die seit dem Eintritt in den entsprechenden Zustand vergangen ist.

Die Zeitanzeige müsste deshalb entsprechend aktualisiert werden. Dieses Problem wird hier nicht gelöst.

Zustandsdiagramme

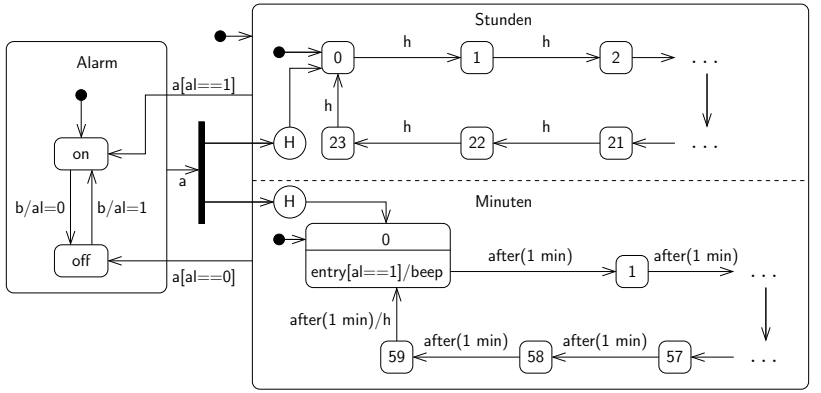
Was fehlt noch?

$\rightsquigarrow$ Beim Wechseln zwischen den Alarm-Zuständen (on,off) muss ein Flag (genannt al) gesetzt werden, um damit den beep-Effekt zu kontrollieren.

Dieses Flag muss mit Hilfe einer Bedingung im Minutenzustand 0 abgefragt werden.

Außerdem betreten wir den Zustand Alarm nun nicht mehr über die flache Historie, sondern fragen mit Hilfe von Bedingungen ab, wie al belegt ist. (Damit ist die Markierung des Anfangszustands eigentlich überflüssig geworden.)

Zustandsdiagramme



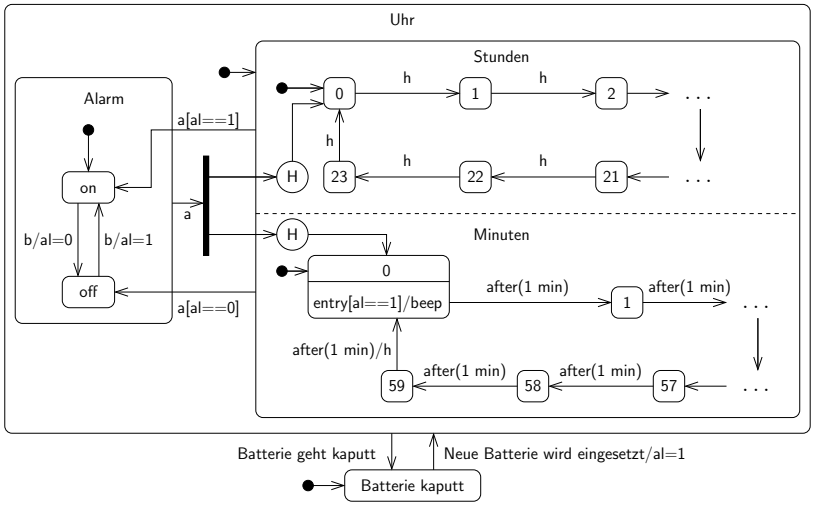
Zustandsdiagramme

Wir modellieren, dass die Batterie der Uhr kaputtgehen kann und gewechselt werden muss.

In diesem Fall will man den zusammengesetzten Zustand nicht über die flache oder tiefe Historie betreten! Es wird also hier tatsächlich die Zeit auf 0:00 zurückgesetzt.

Außerdem setzen wir das Flag al beim Einsetzen der Batterie zurück auf den Anfangswert 1.

Zustandsdiagramme



Zustandsdiagramme

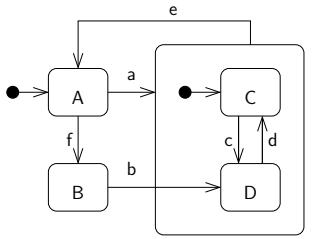
Ein großer Teil der Modellierungsmöglichkeiten von Zustandsautomaten dient dazu, Zustandsdiagramme kompakt und übersichtlich zu notieren.

Oft kann man Zustandsdiagramme “flachklopfen” und zusammengesetzte Zustände auflösen, wodurch man äquivalente Zustandsdiagramme erhält, die die gleichen Übergänge erlauben. Dabei erhält man jedoch im allgemeinen mehr Zustände und/oder Transitionen.

Wir sehen uns einige Beispiele an . . .

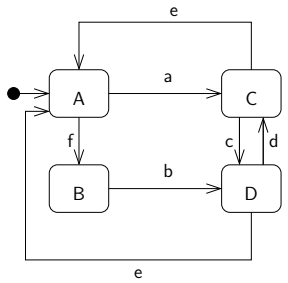
Zustandsdiagramme

Beispiel 1: Wandeln Sie folgendes Zustandsdiagramm in ein “flaches” Zustandsdiagramm um:



Zustandsdiagramme

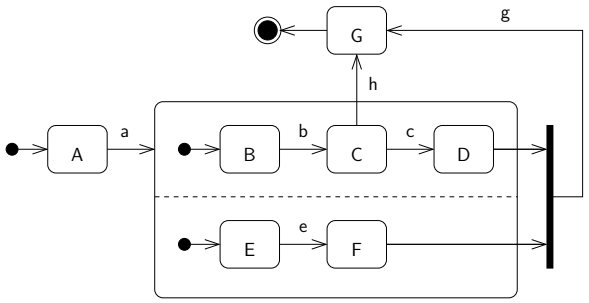
Lösung zu Beispiel 1:



Idee: die Transition, die von dem zusammengesetzten Zustand wegführt, durch mehrere Transitionen ersetzen, die von den inneren Zuständen ausgehen.

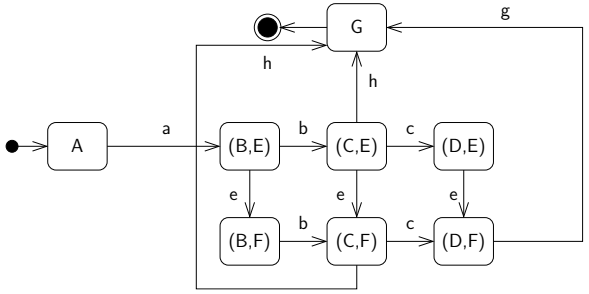
Zustandsdiagramme

Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein "flaches" Zustandsdiagramm um:



Zustandsdiagramme

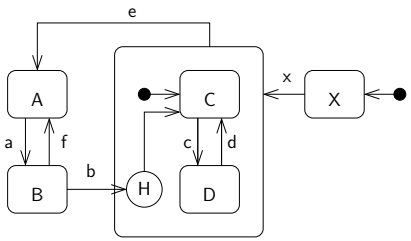
Lösung zu Beispiel 2:



Idee: Kreuzprodukt der Zustandsmengen der Regionen bilden.

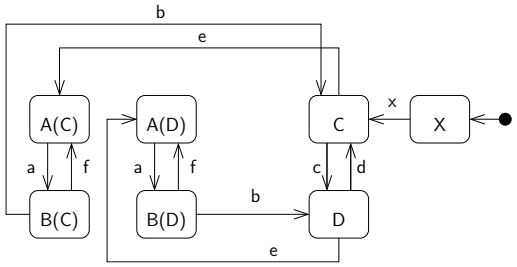
Zustandsdiagramme

Beispiel 3: Wandeln Sie folgendes Zustandsdiagramm in ein "flaches" Zustandsdiagramm um:



Zustandsdiagramme

Lösung zu Beispiel 3:



Idee: in den äußeren Zuständen muss man sich merken, aus welchem Zustand man den zusammengesetzten Zustand verlassen hat. Dies führt zu zusätzlichen Zuständen.

Zustandsdiagramme

Weitere Elemente von Zustandsdiagrammen:

- Unterscheidung zwischen verschiedenen Arten von Ereignissen: call event, signal event, change event, time event, any receive event
- Verzögern und Ignorieren von Ereignissen
- Entscheidungen und Kreuzungen
- Rahmen und Wiederverwendung von Zustandsdiagrammen

Außerdem: Generalisierung und Spezialisierung von Zustandsdiagrammen

Zustandsdiagramme

Zusammenfassung:
Nach Harel werden Zustandsdiagramme bzw. deren Eigenschaften durch folgende "Formel" beschrieben:

UML-Zustandsdiagramme =
Zustandsübergangsdiagramme + Tiefe + Orthogonalität
(+ Broadcast-Kommunikation)

Dabei steht "Orthogonalität" für die Parallelität, die durch Regionen erreicht werden kann.

Sequenzdiagramme

Sequenzdiagramme (sequence diagrams) sind die bekanntesten Vertreter von Interaktionsdiagrammen in UML.

Sie dienen dazu, um Kommunikation und Interaktion zwischen mehreren Kommunikationspartnern zu modellieren und beruhen auf dem Basiskonzept der Interaktion:

Interaktion

Eine Interaktion ist das Zusammenspiel von mehreren Kommunikationspartnern.

Typische Beispiele: Versenden von Nachrichten, Datenaustausch, Methodenaufruf

Sequenzdiagramme

Sequenzdiagramme waren bereits vor Aufnahme in die UML unter dem Namen message sequence charts bekannt.

Im Gegensatz zu Aktivitätsdiagrammen oder Zustandsdiagrammen beschreiben sie im allgemeinen nicht alle Abläufe eines Systems, sondern nur einen oder mehrere mögliche Abläufe.

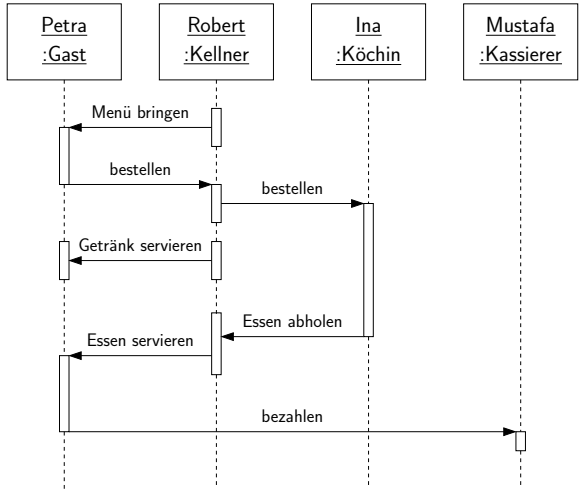
Sequenzdiagramme

Sequenzdiagramme beschreiben Interaktionen in zwei Dimensionen:

- Von links nach rechts: Anordnung der Kommunikationspartner als Lebenslinien
  Oft wird der Partner, der den Ablauf initiiert ganz links angegeben.
- Von oben nach unten: Zeitachse

Sequenzdiagramme

Beispiel Restaurant



Sequenzdiagramme

Kommunikationspartner

Die Kommunikationspartner in einem Sequenzdiagramm werden ähnlich wie in Objektdiagrammen als Rechtecke notiert.

Manchmal werden die Rechtecke auch weggelassen. Menschliche Partner werden auch durch ein Strichmännchen symbolisiert:

Von jedem Kommunikationspartner führt eine gestrichelte Lebenslinie nach unten.

Sequenzdiagramme

Ausführungsbalken

Aktivitäten eines Kommunikationspartners werden durch sogenannte Ausführungsbalken dargestellt.

Parallele Tätigkeiten eines Kommunikationspartners werden dabei durch übereinander liegende Ausführungsbalken beschrieben (siehe rechts oben).

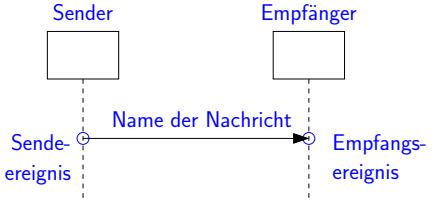
Während die Balken aktive Zeit anzeigen, symbolisieren die gestrichelten Linien passive Zeit.

Sequenzdiagramme

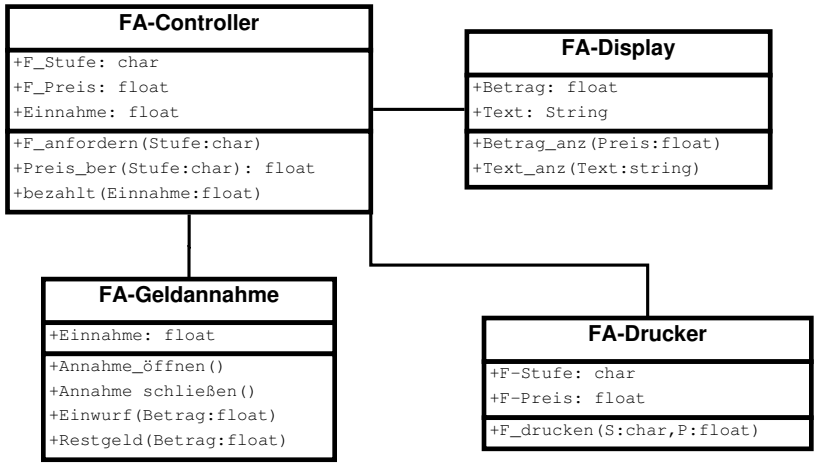
Nachrichten

Die Nachrichten beschreiben die Kommunikationen bzw. Interaktionen der Kommunikationspartner und werden durch Pfeile dargestellt. Eine Nachricht hat einen Sender und einen Empfänger.

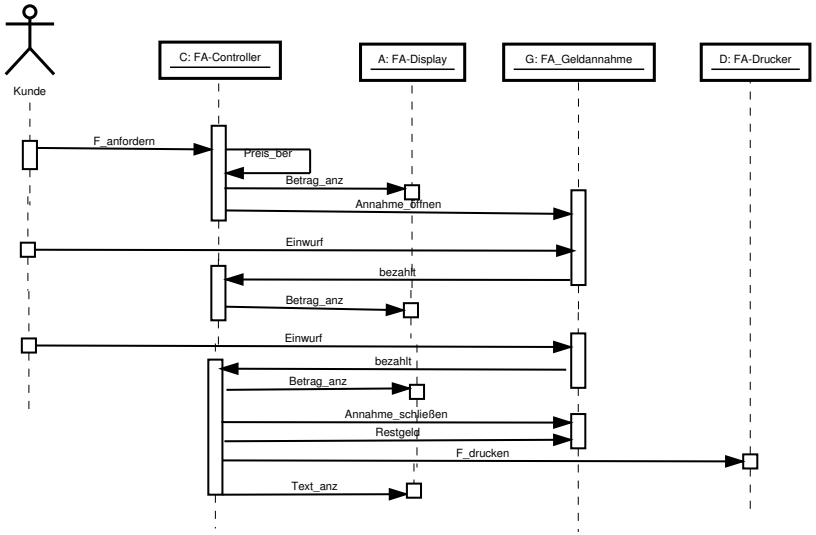
Die Stellen, an denen die Pfeile auf den Lebenslinien auftreffen, nennt man auch Sendeereignis und Empfangsereignis.



Fallstudie: Fahrkartenautomat - Klassendiagramm



Fallstudie: Fahrkartenautomat - Sequenzdiagramm



Synchrone und asynchrone Nachrichten

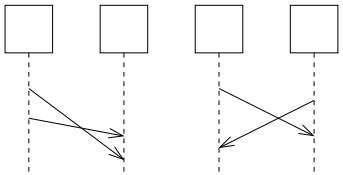
Bei synchroner Kommunikation warten Sender und Empfänger aufeinander. Der Sender macht erst dann weiter, wenn er weiß, dass der Empfänger die Nachricht erhalten hat. Sie wird durch eine schwarze ausgefüllte Pfeilspitze dargestellt.

Synchrone und asynchrone Nachrichten (Fortsetzung)

Bei asynchroner Kommunikation wartet der Sender nicht darauf, dass der Empfänger die Nachricht erhalten hat. Er arbeitet einfach weiter. Die Nachricht wird durch eine einfache Pfeilspitze wird dargestellt.

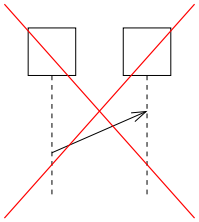
Bei asynchronen Nachrichten kann es zu einer Zeitverzögerung zwischen Sende- und Empfangsereignis kommen, die durch einen geneigten Pfeil beschrieben wird (siehe oben rechts).

Bei asynchroner Kommunikation (aber *nicht* bei synchroner Kommunikation) kann auch der Fall eintreten, dass sich Nachrichten überholen oder kreuzen.



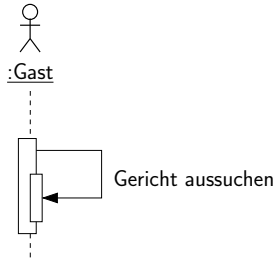
Das Überholen der Nachrichten (oben links) ist nur dann nicht möglich, wenn man explizit einen FIFO-Kanal fordert.

Was jedoch nicht möglich ist, ist eine Nachricht, die “rückwärts” in der Zeit läuft und vor dem Senden ankommt.



Sequenzdiagramme

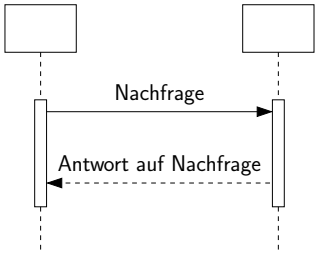
Es ist jedoch möglich, eine Nachricht an sich selbst zu schicken.



Bei einer synchronen Nachricht sollte man dabei parallele Ausführungsbalken verwenden, da das Sende- und Empfangsereignis parallel stattfinden müssen.

Sequenzdiagramme

Nachrichten, die als Antworten auf frühere Nachrichten gedacht sind, werden durch gestrichelte Pfeile notiert. Dabei sollte der Name der ursprünglichen Nachricht wiederholt werden.



Sequenzdiagramme

Wir machen uns nun Gedanken über die Reihenfolge der Ereignisse in einem Sequenzdiagramm. Dazu ist es nützlich sich klarzumachen, was eine (partielle) Ordnung ist.

Partielle Ordnung

Gegen sei eine Menge X. Eine partielle Ordnung auf X ist eine Relation $R \subseteq X \times X$ mit folgenden Eigenschaften:

- Reflexivität: für jedes $x \in X$ gilt $(x, x) \in R$
- Transitivität: aus $(x_1, x_2) \in R$ und $(x_2, x_3) \in R$ für $x_1, x_2, x_3 \in X$ folgt $(x_1, x_3) \in R$.
- Antisymmetrie: aus $(x_1, x_2) \in R$ und $(x_2, x_1) \in R$ für $x_1, x_2 \in X$ folgt $x_1 = x_2$.

Sequenzdiagramme

Weitere Bezeichnungen:

- Partielle Ordnung bezeichnet man oft mit dem Zeichen $\leq$ und notiert in Infix-Schreibweise ($x_1 \leq x_2$ statt $(x_1, x_2) \in R$).
- Wir schreiben $x_1 < x_2$, falls $x_1 \leq x_2$ und $x_1 \neq x_2$.

Vorsicht: Die Relation $<$ ist selbst keine partielle Ordnung.

Totale Ordnung

Eine partielle Ordnung $\leq$ heißt total, wenn für zwei beliebige Elemente $x_1, x_2 \in X$ immer $x_1 \leq x_2$ oder $x_2 \leq x_1$ gilt.

Beispiele:

- Die $\leq$-Relation auf den ganzen Zahlen $\mathbb{Z}$ ist eine totale Ordnung.
- Ein typisches Beispiel für eine partielle Ordnung, die nicht total ist, ist die Inklusionsordnung $\subseteq$ auf Mengen.

Reflexiv-transitive Hülle

Für eine gegebene Relation $R \subseteq E \times E$ beschreiben wir deren reflexiv-transitive Hülle R^*. Man erhält sie aus R, indem man

- alle Paare (e, e) mit $e \in E$ zu R hinzufügt *und*
- bei $(e_1, e_2), (e_2, e_3) \in R$ auch (e_1, e_3) zu R hinzufügt. Dieser Prozess wird so lange wiederholt, bis keine neuen Paare mehr hinzukommen.

- Die reflexiv-transitiv Hülle R^* einer Relation R ist immer reflexiv und transitiv. Sie ist jedoch nicht immer antisymmetrisch und daher nicht notwendigerweise eine Ordnung.
- Die Relation R^* ist die kleinste reflexive und transitive Relation, die R enthält.

Beispiel:

Die reflexiv-transitive Hülle von

$$R = \{(1,2), (2,3), (3,4), (3,5)\}$$

ist

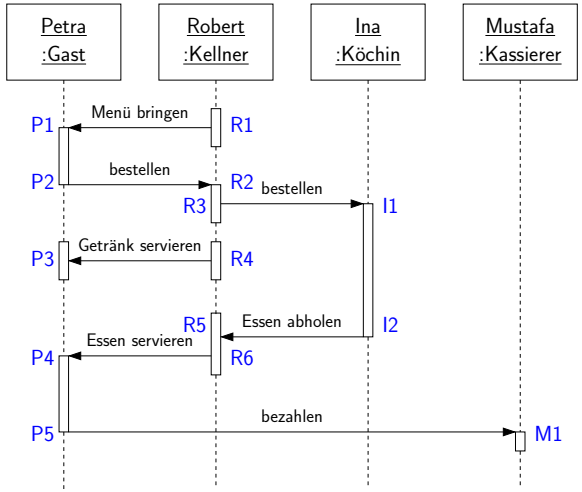
$$\begin{aligned} R^* \quad = \quad & \{(1,2), (2,3), (3,4), (3,5), \\ & (1,1), (2,2), (3,3), (4,4), (5,5), \\ & (1,3), (1,4), (1,5), (2,4), (2,5)\} \end{aligned}$$

Die Elemente von Sequenzdiagrammen, die nun geordnet werden, sind die Ereignisse, d.h., Sende- und Empfangsereignisse.

In den Sequenzdiagrammen, wie wir sie bisher kennengelernt haben, sind die Ereignisse immer total geordnet, nach folgendem Schema:

- (Sendeereignis der Nachricht N) $<$ (Empfangsereignis von N)
- Die restlichen Ereignisse sind entsprechend dem zeitlichen Verlauf geordnet.

Sequenzdiagramme

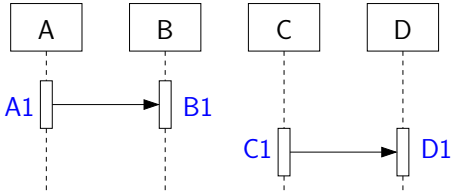


$R1 < P1 < P2 <$
$R2 < R3 < I1 <$
$R4 < P3 < I2 <$
$R5 < R6 < P4 <$
$P5 < M1$

Sequenzdiagramme

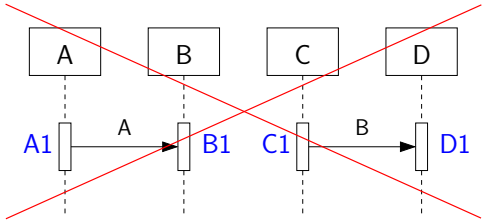
In einem einfachen Sequenzdiagramm (Erweiterungen werden noch besprochen) werden daher die Ereignisse immer in eine Reihenfolge gebracht. Die Ordnung ist daher total. Im folgenden Diagramm gilt beispielsweise:

$$A1 < B1 < C1 < D1$$



Sequenzdiagramme

Nicht so günstig sind Diagramme, aus denen die Reihenfolge von Nachrichten nicht klar hervorgeht:



Insbesondere ist die Darstellung echter Parallelität in Sequenzdiagrammen nicht vorgesehen.

Sequenzdiagramme

Äquivalenz von Sequenzdiagrammen

Zwei Sequenzdiagramme sind äquivalent, wenn sie die gleichen Ereignisse enthalten und die Reihenfolge der Ereignisse identisch ist.

Dabei können die Diagramme durchaus verschieden gezeichnet sein (andere Reihenfolge der Kommunikationspartner, verschiedene Abstände zwischen den Ereignissen, . . .).

Sequenzdiagramme

Bisher haben wir gesehen, wie man mit einem Sequenzdiagramm *einen* möglichen Ablauf beschreiben kann. In manchen Fällen möchte man jedoch *mehrere* (vielleicht sogar alle) Abläufe beschreiben.

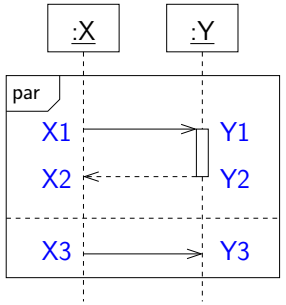
Dazu gibt es die Möglichkeit, kombinierte Fragmente zu verwenden, bei denen mehrere (Interaktions-)Operanden (= Teil-Sequenzdiagramme) mit Hilfe von Interaktions-Operatoren zusammengesetzt werden.

Wir betrachten im folgenden einige dieser Interaktions-Operatoren.

Sequenzdiagramme

Interaktionsoperator par (Parallelität)

Hier sind die Operanden in beliebiger Reihenfolge ausführbar. Die Reihenfolge der Ereignisse *in* den Operanden muss aber gewahrt werden, ansonsten gibt es keine Bedingungen.

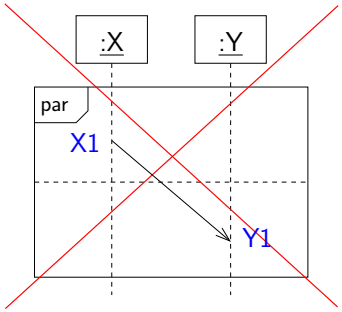


Dabei wird der Operator par links oben in der Ecke angegeben.

Eine gestrichelte waagrechte Linie trennt die verschiedenen Operanden.

Sequenzdiagramme

Nachrichten, die von einem Operanden in einen anderen laufen, sind nicht zugelassen.



Sequenzdiagramme

Ordnung auf den Ereignissen:

- X1 $<$ Y1 $<$ Y2 $<$ X2 (Operand 1)
- X3 $<$ Y3 (Operand 2)

Ansonsten gibt es keine weiteren Einschränkungen

Dieses Sequenzdiagramm beschreibt insgesamt fünfzehn Abläufe, beispielsweise:

- X1, Y1, X3, Y3, Y2, X2
- X3, X1, Y1, Y2, X2, Y3
- ...

Sequenzdiagramme

Weitere Bemerkungen: um die Ereignis-Ordnung innerhalb eines par-Operators zu bestimmen ...

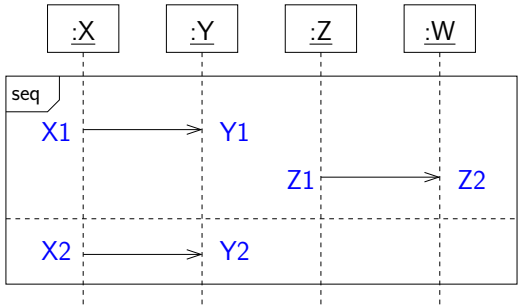
- verwendet man die partiellen Ordnungen der einzelnen Operanden ($\leq_1$, $\leq_2$)
- *und* vereinigt diese. Die Vereinigung $\leq_1 \cup \leq_2$ ist ebenfalls eine partielle Ordnung und beschreibt alle möglichen Reihenfolgen von Ereignissen.

Jede Ereignisreihenfolge, bei der ein Ereignis X immer vor einem Ereignis Y auftritt, falls X<Y gilt, ist eine mögliche Reihenfolge.

Sequenzdiagramme

Interaktionsoperator seq (schwache Sequenz)

Die Reihenfolge der Ereignisse *in* den Operanden muss wieder gewahrt werden, außerdem ist die Reihenfolge der Ereignisse auf den Lebenslinien (von oben nach unten) festgelegt.



Sequenzdiagramme

Ordnung auf den Ereignissen:

- X1 $<$ Y1 $<$ Z1 $<$ Z2 (Operand 1)
- X2 $<$ Y2 (Operand 2)
- X1 $<$ X2 (Lebenslinie von X)
- Y1 $<$ Y2 (Lebenslinie von Y)

Dieses Sequenzdiagramm beschreibt neun verschiedene Abläufe, beispielsweise:

- X1, X2, Y1, Z1, Z2, Y2
- X1, Y1, X2, Y2, Z1, Z2
- ...

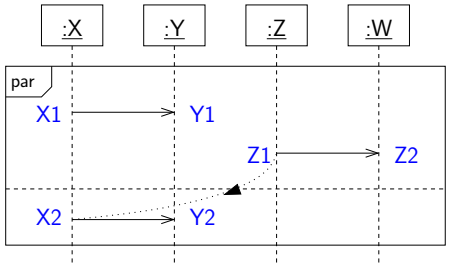
Sequenzdiagramme

Weitere Bemerkungen: um die Ereignis-Ordnung innerhalb eines seq-Operators zu bestimmen ...

- verwendet man die partiellen Ordnungen der einzelnen Operanden ($\leq_1$, $\leq_2$),
- außerdem die partiellen Ordnungen der einzelnen Lebenslinien ($\leq_X$, $\leq_Y$, ...),
- vereinigt diese und bestimmt die reflexiv-transitive Hülle. Dabei entsteht eine partielle Ordnung, die alle möglichen Reihenfolgen von Ereignissen beschreibt.

Sequenzdiagramme

Bei Interaktions-Operanden ist es außerdem möglich, zusätzliche Ordnungsbeziehungen einzuführe, die ansonsten nicht abgedeckt sind. (Durch gestrichelte Linien mit Pfeil in der Mitte.)



Die neue partielle Ordnung kann man hier dadurch bestimmen, indem man die neuen Paare hinzufügt und die reflexiv-transitive Hülle der Gesamtrelation bildet.

Sequenzdiagramme

Weitere Interaktions-Operatoren (Liste ist nicht vollständig):

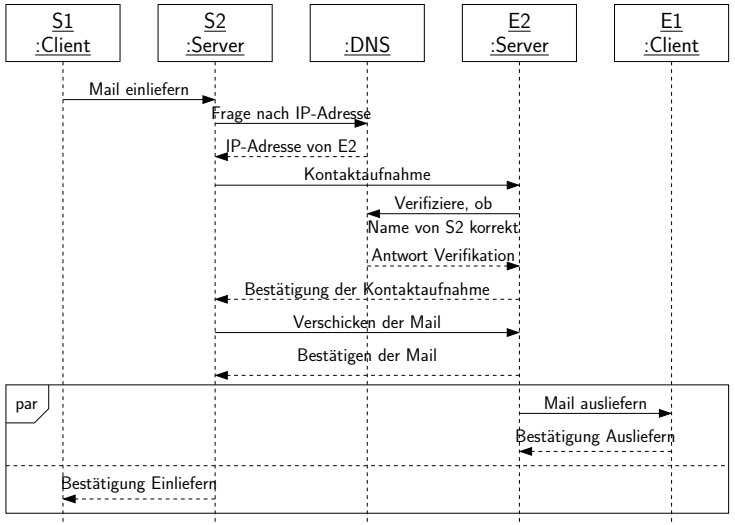
- strict (Strikte Sequenz): Die Reihenfolge der Ereignisse wird von oben nach unten strikt eingehalten (wie in einem einzelnen Operanden).
- alt (Alternative): entsprechend einer If-Then-Else-Anweisung (mit Bedingungen)
- neg (Negation): beschreibt eine ungültige Nachrichtenreihenfolge
- loop (Schleife): beschreibt die Wiederholung eines Operanden

Sequenzdiagramme

Beispiel: wir modellieren ein Protokoll, bei dem ein Client-Rechner S einen E-Mail an einen anderen Client R verschickt.

Weitere Beteiligte sind der Mail-Server von S, der Mail-Server von R und der DNS-Server, der benötigt wird, um Mail-Adressen in die IP-Adressen des entsprechenden Servers umzuwandeln (DNS = domain name system).

Sequenzdiagramme



Sequenzdiagramme

Bemerkung: dieses Sequenzdiagramm ist an den Ablauf im SMTP-Protokoll angelehnt (SMTP = send mail transport protocol), ist jedoch erheblich vereinfacht.

Sequenzdiagrammen zu vielen TCP/IP-Netzwerkprotokollen findet man unter:

`http://www.eventhelix.com/Realtimemantra/Networking/`

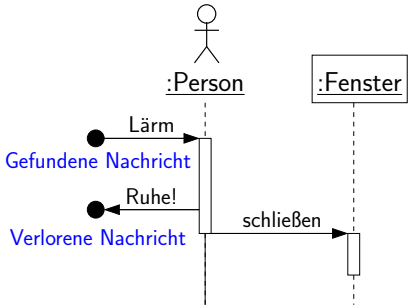
Sequenzdiagramme

Es gibt noch einige weitere Arten von Nachrichten:

Gefundene und verlorene Nachrichten

Bei gefundenen und verlorenen Nachrichten werden das Sende- bzw. das Empfangsereignis nicht explizit modelliert. Die Nachrichten tauchen quasi aus der Umgebung auf und verschwinden wieder dorthin.

Solche Nachrichten werden benötigt, wenn die entsprechenden Kommunikationspartner nicht mitmodelliert werden.

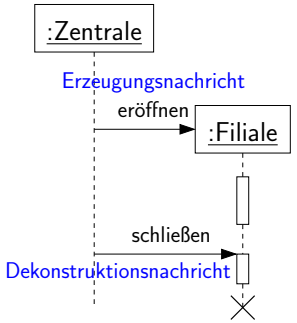


Sequenzdiagramme

Erzeugung und Dekonstruktion von Objekten

Außerdem kann es passieren, dass Objekte nicht während des ganzen Ablaufs zur Verfügung stehen. Sie können während des Ablaufs dynamisch erzeugt und wieder zerstört werden.

Dies erfolgt zumeist durch sogenannte Erzeugungs- und Dekonstruktionsnachrichten und wird folgendermaßen dargetellt:



Kommunikationsdiagramme

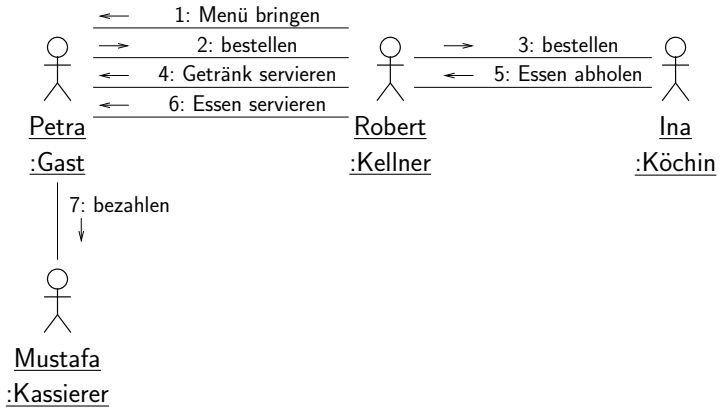
Kommunikationsdiagramme enthalten dieselbe Information wie Sequenzdiagramme, werden jedoch anders dargestellt.

Während bei Sequenzdiagrammen der Fokus eher auf dem zeitlichen Ablauf liegt, heben Kommunikationsdiagramme eher die Kommunikationsbeziehungen der Teilnehmer hervor.

Kommunikationsdiagramme gehören – genau wie Sequenzdiagramme – zur Klasse der Interaktionsdiagramme.

Kommunikationsdiagramme

Das Sequenzdiagramm Restaurantbesuch Diagramm wird folgendermaßen als Kommunikationsdiagramm dargestellt:



Kommunikationsdiagramme

Dabei werden die Kommunikationspartner wie bisher durch Rechtecke oder Strichmännchen dargestellt. Es wird jedoch kein zeitlicher Ablauf mehr dargestellt.

Die Interaktionen bzw. Nachrichten werden durch Linien notiert, an denen die Namen der Nachrichten und die Senderichtung ($\rightarrow$) stehen.

Die Nummerierung der Nachrichten (1,2,3,...) gibt die Reihenfolge an. Durch Buchstaben hinter den Nummern (2a,2b,...) beschreibt man parallele Nachrichten, die in beliebiger Reihenfolge angeordnet sein können.

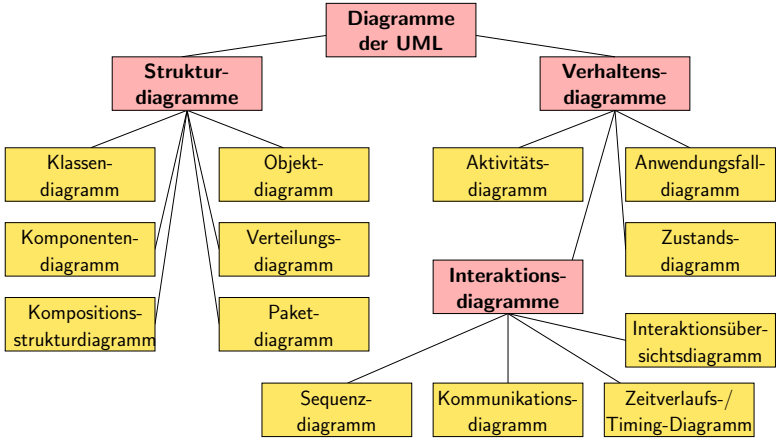
Überblick über weitere UML-Diagramme

Wir schließen die Vorlesung mit einem kurzen Überblick über die noch fehlenden Typen von UML-Diagrammen ab.

Zuletzt gibt es dann noch ein paar Abschlussbemerkungen.

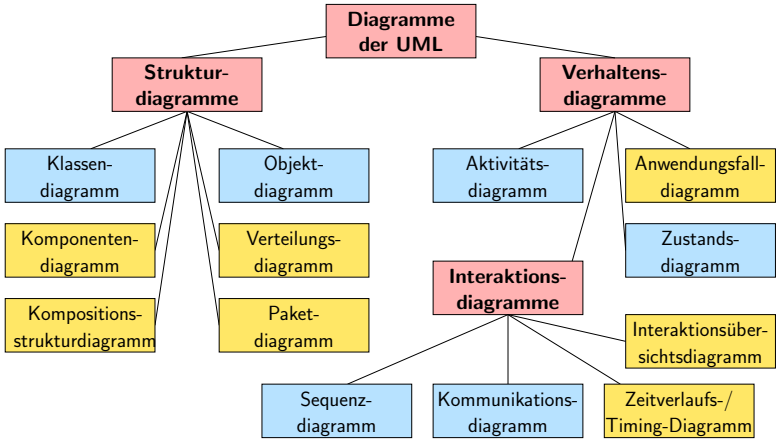
Überblick über weitere UML-Diagramme

UML-Diagramme Liste



Überblick über weitere UML-Diagramme

UML-Diagramme (blau: bereits behandelte Diagramme)



Überblick über weitere UML-Diagramme

Wir beginnen zunächst mit den beiden noch fehlenden Arten von Interaktionsdiagrammen:

- Timing-Diagramme bzw. Zeitverlaufsdiagramme
- Interaktionsübersichtsdiagramme

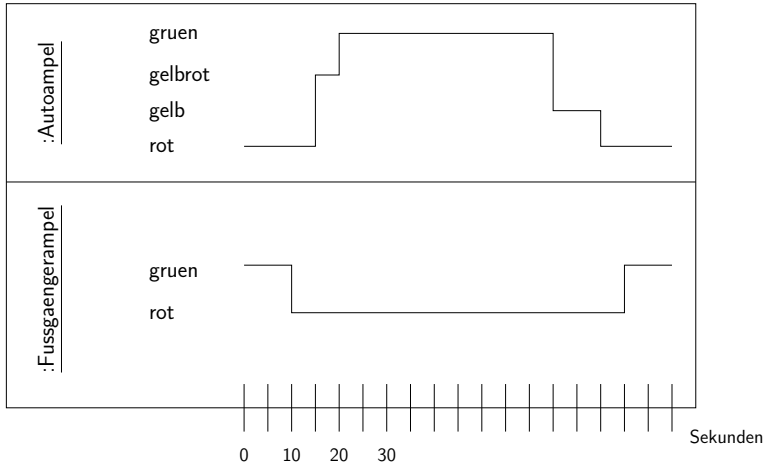
Timing-Diagramme

Timing-Diagramme sind in der Elektrotechnik weit verbreitet und zeigen an, zu welchem Zeitpunkt welcher Kommunikationspartner welchen Zustand einnimmt.

Dabei wird von links nach rechts (horizontal) die Zeit aufgetragen und von oben nach unten werden die Kommunikationspartner und deren Zustände aufgetragen.

Timing-Diagramme

Beispiel: Ampelschaltung



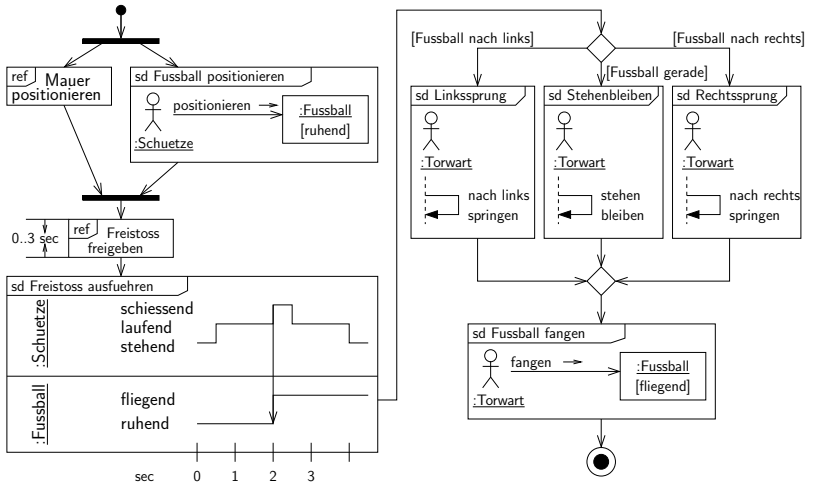
Interaktionsübersichtsdiagramme

Interaktionsübersichtsdiagramme sind im wesentlichen Aktivitätsdiagramme, die – anstatt der Aktionen – hierarchisch weitere Interaktionsdiagramme (Sequenzdiagramme, Kommunikationsdiagramme, Timing-Diagramme) enthalten können.

Sie werden eingesetzt, wenn es eine größere Menge verschiedener Arten von Aktionen gibt, über die man sonst nur schwer den Überblick behalten kann.

Als Beispiel betrachten wir ein Interaktionsübersichtsdiagramm, das den Ablauf eines Freistoßes in einem Fussballspiel modelliert.

Interaktionsübersichtsdiagramme



Interaktionsübersichtsdiagramme

Bemerkungen:

- Interaktionsreferenzen (Schlüsselwort ref) werden verwendet, um an anderer Stelle definierte Interaktionsdiagramme wiederzuverwenden.
- Anstatt der Aktionen wie in Aktivitätsdiagrammen werden ganze Interaktionsdiagramme verwendet, die alle sd als Diagrammtyp für Interaktionsdiagramme erhalten.

Anwendungsfalldiagramme

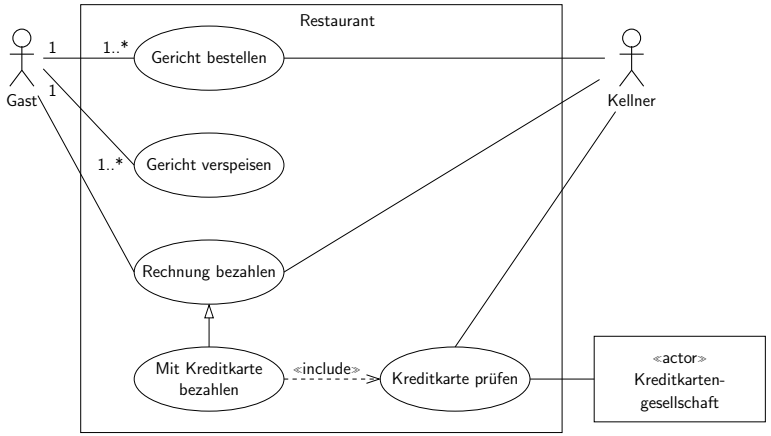
In frühen Stadien der Entwicklung und bei der Kommunikation mit dem Auftraggeber spielen auch Anwendungsfalldiagramme (engl. use case diagrams) eine grosse Rolle.

Anwendungsfalldiagramme modellieren die Funktionalität des Systems

- auf einem hohen Abstraktionsniveau;
- aus der Black-Box-Sicht des Anwenders (d.h., nur das von außen Sichtbare soll beschrieben werden, nicht die interne Realisierung);
- durch Spezifikation der Schnittstellen.

Die Anwender bzw. Nutzer tauchen als sogenannte Akteure in den Diagrammen auf.

Beispiel: Restaurant



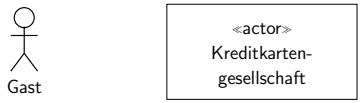
Anwendungsfalldiagramme bestehen aus folgenden Komponenten:

Systemgrenze

Die Systemgrenze ist ein Rechteck, das beschreibt, was sich außerhalb und innerhalb des zu erstellenden Systems befindet.

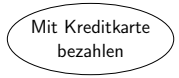
Akteur

Ein Akteur ist ein Typ oder eine Rolle, die ein externer Benutzer oder ein externes System während der Interaktion mit dem System einnimmt. Menschliche Akteur werden durch Strichmännchen symbolisiert, andere Akteure durch ein Rechteck, das mit dem Schlüsselwort «actor» gekennzeichnet ist



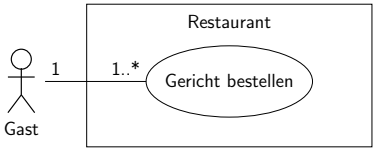
Anwendungsfall

Ein Anwendungsfall ist eine Menge von Aktionen, die von dem System bereitgestellt werden und die einen Nutzen für einen oder mehrere Akteure bringen. (D.h., es handelt sich dabei um eine Art "Service" des Systems.) Ein Anwendungsfall wird durch eine Ellipse dargestellt.



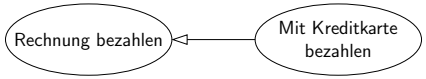
Assoziation

Wie bei Klassendiagrammen gibt es Assoziationen, vor allem zwischen Akteuren und Anwendungsfällen (welcher Akteur kann welchen Anwendungsfall nutzen?).
Assoziationen dürfen die üblichen Beschriftungen besitzen (Multiplizitäten, etc.).



Generalisierung/Spezialisierung

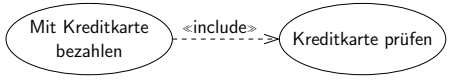
Anwendungsfälle können andere Anwendungsfälle spezialisieren, d.h., sie können von ihnen erben. Dabei werden – wie bei Klassen – auch alle Assoziationen geerbt.



Auch Spezialisierungsbeziehungen zwischen Akteuren sind möglich.

Include-Beziehung

Bei einer Include-Beziehung wird modelliert, dass ein Anwendungsfall die Funktionalität eines anderen Anwendungsfalles auf jeden Fall nutzt. D.h., der zweite Anwendungsfall wird immer als eine Art "Unterprozedur" aufgerufen.



Es gibt auch sogenannte Extend-Beziehungen, bei denen ein Anwendungsfall nur unter bestimmten Bedingungen in einen anderen Anwendungsfall eingebunden wird.

Bemerkungen:

- Anwendungsfalldiagramme sollten nicht zu viele Details enthalten.
- Sie sind ein einfaches Mittel, um Anwenderwünsche zu diskutieren und sollten nur eine grobe Sicht auf die Funktionalität des Systems darstellen.
- Bei Bedarf müssen bestimmte Anwendungsfälle dann noch textuell oder mit Hilfe anderer UML-Diagramme genauer beschrieben werden.

Strukturierte, textuelle Beschreibung eines Anwendungsfalls

Af.-Nr. **Name des Anwendungsfalles**

- Vorbedingungen:
- Nachbedingungen:
- Nicht-funktionale Anforderungen:
- Beschreibung: ..
- Variationen:
- Regeln:
- Services:
- Ansprechpartner:
- Anmerkungen / offene Fragen:
- Dialogbeispiele oder - muster:
- Diagramme:

Beispiel: Formular zur Beschreibung eines Anwendungsfalls

Quantifizierte Anforderungen	an die Kommunikation zwischen Akteur und Anwendungsfall [Hier klicken und Text eingeben]
Antwortzeit-verhalten	Wie schnell erwartet der Akteur das Ergebnis des Anwendungsfalls? Quantifizieren Sie hier das Antwortzeitverhalten, sofern Sie es nicht bereits unter den Zusatzinformationen zum Anwendungsfall getan haben. [Hier klicken und Text eingeben]
Systemverfügbarkeit	Von wann bis wann erwartet der Akteur die Verfügbarkeit des Systems? [Hier klicken und Text eingeben]
Zuverlässigkeit	Welche Zeit zwischen dem Auftreten von Fehlern ist für den Akteur akzeptabel? [Hier klicken und Text eingeben]
Sicherheit	Welche Backup- und Recovery-Mechanismen sind für die erfolgreiche Arbeit des Akteurs unerläßlich? [Hier klicken und Text eingeben]
Schutz	In welchem Umfang ist Schutz vor unberechtigten Benutzern im Rahmen des Anwendungsfalls vorzusehen? [Hier klicken und Text eingeben]

Formular zur Beschreibung eines Anwendungsfalls (2)

Ressourcenbelegung	Welche Ressourcen wie Hauptspeicher, Plattenplatz, Leitungen etc. stehen einem Akteur zu bzw. dürfen im Rahmen des Anwendungsfalls in welchem Umfang belegt werden? [Hier klicken und Text eingeben]
Dokumentation	In welchem Umfang benötigt ein Akteur Hilfe zu einem Anwendungsfall und in welcher Form (Online-Hilfe, Referenzkarte, Handbuch etc.) soll sie angeboten werden? [Hier klicken und Text eingeben]
Erweiterbarkeit	Welche zukünftigen Erweiterungen des Anwendungsfalls müssen in Hinsicht auf diese Kommunikationsbeziehung berücksichtigt werden? [Hier klicken und Text eingeben]
Portabilität	Auf welchen Systemen muß der Anwendungsfall ablauffähig sein? [Hier klicken und Text eingeben]
Sonstige	[Hier klicken und Text eingeben]

Überblick über weitere UML-Diagramme

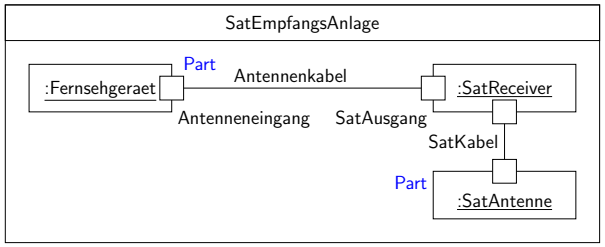
Wir sehen uns nun noch (ganz kurz) die fehlenden vier Typen von Strukturdiagrammen an. UML-Übersicht

- Kompositionsstrukturdiagramme
- Paketdiagramme
- Verteilungsdiagramme
- Komponentendiagramme

Im weitesten Sinne dienen sie alle dazu die (übergeordnete) Struktur bzw. Architektur eines Systems darzustellen.

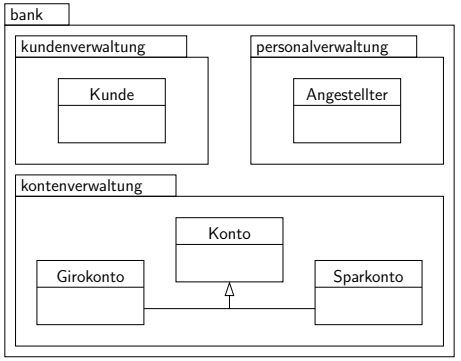
Kompositionsstrukturdiagramme

Kompositionsstrukturdiagramme (engl. composite structure diagrams) beschreiben die interne Struktur von Komponenten (z.B. Klassen) in einer White-Box-Darstellung. Instanzen von durch Komposition verbundenen Klassen werden dabei als sogenannte Parts innerhalb der Klasse dargestellt.



Paketdiagramme

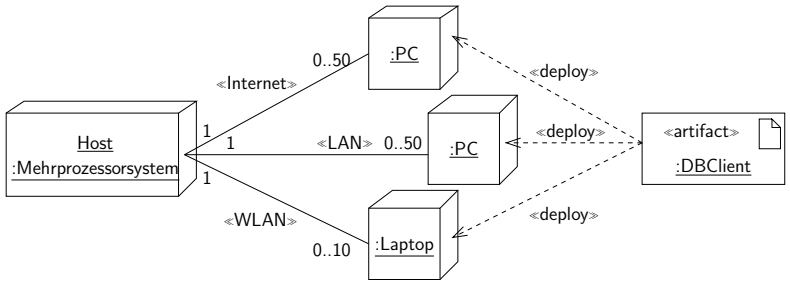
Ein großes Softwaresystem muss in Pakete bzw. Module gegliedert werden. Paketdiagramme (engl. package diagrams) beschreiben die statische Struktur eines großen Systems durch Zusammenfassen von Klassen in Paketen.



Verteilungsdiagramme

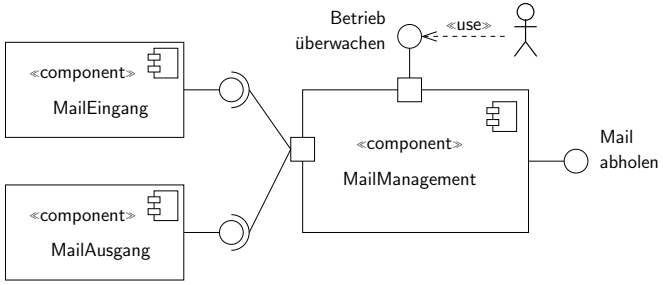
Verteilungsdiagramme (engl. deployment diagrams) beschreiben die Hardware-Komponenten, die in einem System benutzt werden und wie diese in Beziehung stehen.

Sie können auch konkrete physische Informationseinheiten (Dateien, etc.) enthalten, die als Artefakte bezeichnet werden.



Komponentendiagramme

Komponentendiagramme (engl. component diagrams) beschreiben im Gegensatz dazu die Software-Architektur eines Systems. Es beantwortet die Frage, wie Klassen zur Laufzeit zu größeren Komponenten zusammengefasst werden und welche Schnittstellen angeboten und genutzt werden.



UML-Vorgehensweise

a) Von den Anforderungen zu Klassendiagrammen
   - und dann über Anwendungsfalldiagrammen zu Interaktionsdiagrammen zu Zustandsdiagrammen
   - parallel Übergang zu OO-Design mit Hilfe von Komponentendiagrammen
   - Aktivitätsdiagramme nahezu beliebig zur Ablaufveranschaulichung

b) Von den Anforderungen zu Use Case-Diagrammen
   - und dann Aktivitätsdiagrammen zur Detailbeschreibung von Anwendungsfällen
   - und dann über Klassendiagramme zu Interaktionsdiagrammen
   - und dann zu Zustandsdiagrammen
   - parallel Übergang zum OO-Design mit Hilfe von Komponentendiagrammen

Beide Ansätze nur als grobe Richtlinien, nicht als Dogmen!

Abschließende Bemerkungen

UML als semi-formale Modellierungsmethode

UML ist eine semi-formale Modellierungsmethode, d.h., nicht bei jedem Sprachelement gibt es vollständige Einigkeit über die Bedeutung.

Trotz dieser Kritik ist die UML ein großer Fortschritt, da sie vereinheitlichte Diagramme bereitstellt, die von jedem Beteiligten am Softwareentwicklungsprozess verstanden werden können.

Abschließende Bemerkungen

Konsistenz von Modellen

Bei der Beschreibung eines großen Systems durch mehrere Modelle ist auch darauf zu achten, dass die Modelle untereinander konsistent sind. (Beispielsweise: Klassennamen, die in einem Sequenzdiagramm verwendet werden, tauchen auch im Klassendiagramm auf.)

Es gibt Ansätze, solche Konsistenz (halb-automatisch) zu erreichen, indem man sogenannte Modelltransformationen durchführt und verschiedene Diagramme ineinander übersetzt.

Abschließende Bemerkungen

Es gibt noch viele Aspekte in der UML, die wir nicht betrachtet haben. Ein wichtiger Teil der UML ist beispielsweise die OCL (Object Constraint Language), eine formale Beschreibungssprache, in der man zusätzliche Bedingungen und Invarianten ausdrücken kann.

Weitergehende Informationen über UML gibt es in zahllosen Büchern (siehe Literaturliste) und auf den Seiten der OMG (Object Management Group):

```
http://www.omg.org/technology/documents/formal/uml.htm
```

Und natürlich gibt es neben UML und Petri-Netzen noch zahlreiche andere Modellierungsmethoden!