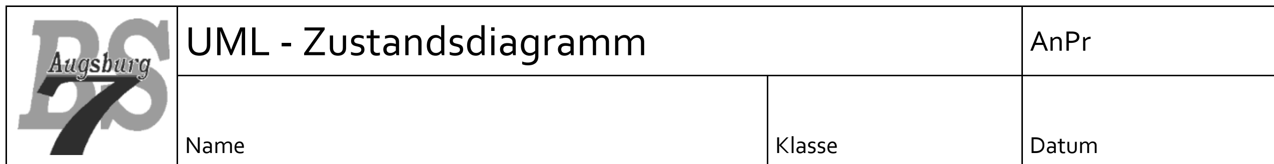


1 Allgemeines

Die Zustandsdiagramme in UML basieren im Wesentlichen auf den Statecharts von David Harel. Der Grundgedanke ist, das Verhalten eines endlichen Zustandsautomaten grafisch zu modellieren. Bei Zustandsautomaten gelten folgende Grundregeln:

- der Zustandsautomat befindet sich zu jeder Zeit in einem der definierten Zustände
- der Zustandsautomat kann niemals in zwei Zuständen gleichzeitig sein (gilt nicht für UML!)
- es gibt eine endliche Menge an Zuständen
- jeder Zustand kann eingenommen werden

Im Rahmen von UML wurde das Statechart jedoch **erweitert**, um das **Zustandsdiagramm** breiter nutzbar zu machen. So ist es im UML Zustandsdiagramm durchaus möglich, dass ein Zustandsdiagramm zwei Zustände gleichzeitig darstellen kann, wobei man dies dergestalt interpretiert, dass man einen inneren Prozess betrachtet, der in sich zwei eigenständige Systeme aufweist („nebenläufige Teilzustände“). Da dies jedoch im Regelfall für Verwirrung sorgt, wird dieses Feature **selten** genutzt. Aus diesem Grunde werden wir uns an dieser Stelle „nur“ mit den elementaren Symbolen des Zustandsdiagramms beschäftigen.

Das Ziel von Zustandsdiagrammen ist es Systemzustände, deren Übergänge und die damit verbundenen Aktionen zu modellieren. Damit verbunden sind folgende Eigenschaften:

- sie modellieren das dynamische Verhalten eines Systems
- sie fokussieren sich auf die Systemreaktionen basierend auf Änderungen
- sie zeigen die mögliche bzw. erwartete Reihenfolge von Ereignissen auf

Kurz, sie zeigen uns das Verhalten des Systems auf – also was macht das System basierend auf welchen Ereignissen. Zustandsdiagramme, welche primär dieses Verhalten modellieren nennt man „**Verhaltensautomat**“.

Daneben gibt es auch Modelle, welche sich primär darauf konzentrieren, welche Ereignisse in welchem Zustand erlaubt bzw. erwartet sind. Solche Modelle sind „**Protokollmaschinen**“ oder Protokollautomaten“. Laut Wikipedia wir ein Protokoll wie folgt definiert:

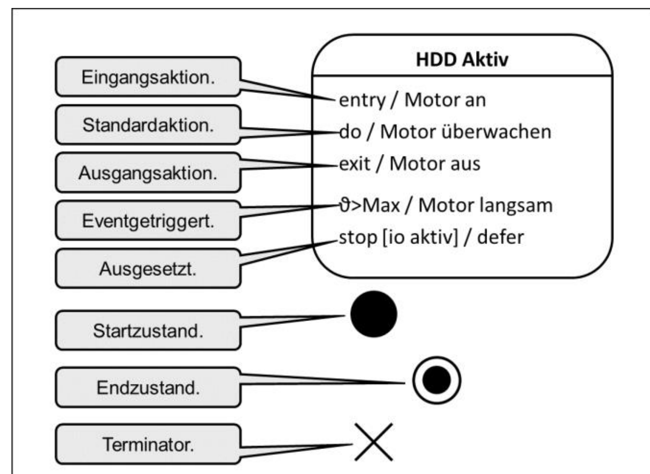
„Ein Protokoll zeichnet auf, hält fest oder schreibt vor, zu welchem Zeitpunkt oder in welcher Reihenfolge welcher Vorgang durch wen oder durch was veranlasst wurde oder wird.“

Hier ist der Fokus also nicht das innere Verhalten, sondern die Nutzung von außen.

2 UML-Zustandsdiagramm, Notation

Hier die wichtigsten Elemente des Zustandsdiagramms.

2.1 Zustand



Aus der Sichtweise eines Programmierers ist ein Zustand nichts anderes als die Belegung einer Zustandsvariable mit fest definierten Werten, also eher eine virtuelle, als eine in realen Systemen sichtbare Eigenschaft.

Die rechts dargestellte Notation geht von der Existenz möglicher **Aktionen** aus, weshalb der **waagerechte Strich** diese vom Zustandsnamen abgrenzt. Ohne diese Aktionen kann man auf den Strich verzichten.

Grundsätzlich finden wir vier verschiedene Typen von Zuständen:

- Endzustand
- Terminator
- Startzustand
- „normaler" Zustand – dies ist der Standardfall

Der **Endzustand** terminiert die dargestellte Ebene von Zuständen. Gibt es in dem Zustandsautomat nur eine Ebene, so terminiert der gesamte Automat. Sind mehrere Ebenen übereinandergestapelt, so wird die Verarbeitung an die darunterliegende (aufrufende) Ebene weitergereicht. **Terminatoren** beenden immer den gesamten Automat. Der Endzustand ist somit vergleichbar mit dem „return Statement" in Programmiersprachen, welches das aktuelle Unterprogramm terminiert und im Falle von „main" das gesamte Programm. Der Terminator wäre dann vergleichbar mit „System.exit(0)".

Der **Startzustand** wiederum stellt den Einstiegspunkt in die Ebene dar. Es darf nur einen einzigen geben. Weiterhin muss der Automat in der Lage sein, vom Startzustand auf jeden Fall in einen normalen Zustand zu wechseln – er darf also nicht im Startzustand verharren. Dies wird normalerweise dadurch erreicht, dass eine Transition ohne Bedingungen existiert, welche in einen normalen Zustand wechselt.

Der normale Zustand besitzt einen Namen und optional eine Liste von Aktionen. Hierbei unterscheiden wir zum einen die Fälle basierend auf dem **zeitlichen Ablauf**:

- **Entry - Aktion**: Wird als erstes ausgeführt, sobald der Zustand eingenommen wird.
- **Do – Aktion**: Wird ausgeführt, wenn der Zustand eingenommen wurde.
- **Exit – Aktion**: Wird ausgeführt, wen der Zustand verlassen wird – also als letztes.

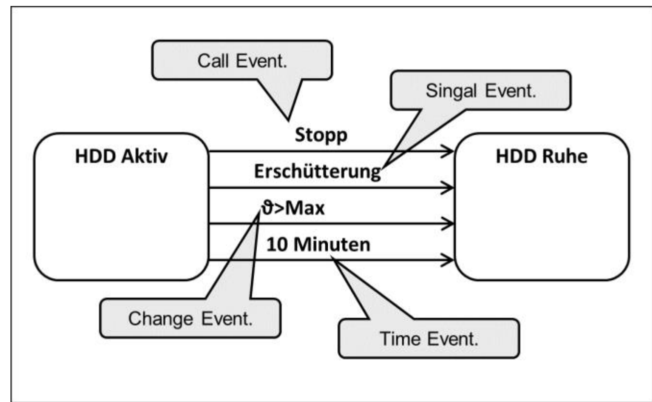
Weiterhin werden basierend auf Events Aktionen ausgeführt:

- **Eventgetriggerte Aktion**: Wird Ausgeführt, wenn das Event eintrifft.

Events können jedoch auch ausgesetzt werden:

- **Deferred Events**: Das Event wird nicht ausgewertet und steht für andere Trigger zur Verfügung – es wird also nicht „verbraucht".

2.2 Transitionen



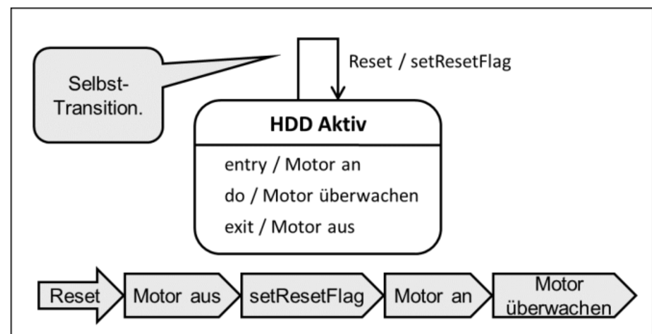
Eine Transition stellt einen Statuswechsel dar – womit sie gerichtet ist. Die Auslöser einer Transition sind:

- **Call Event**: Wenn ein Befehl oder inneres Kommando auftritt.
- **Signal Event**: Asynchrones Signal (meist von außen)
- **Change Event**: Zustandsänderung einer Systemeigenschaft (Prüfergebnis ist boolean)
- **Time Event**: Entweder Zeitpunkt oder Zeitspanne

Sobald das Event aufgetreten ist (und es nicht vom Zustand verbraucht wurde), wird die Transition ausgeführt. Von einem Zustand können mehrere Transitionen ausgehen, welche vom Event her disjunkt sein müssen. Alternativ zu mehreren Pfeilen können die Events auch kommasepariert auf einen Pfeil geschrieben werden.

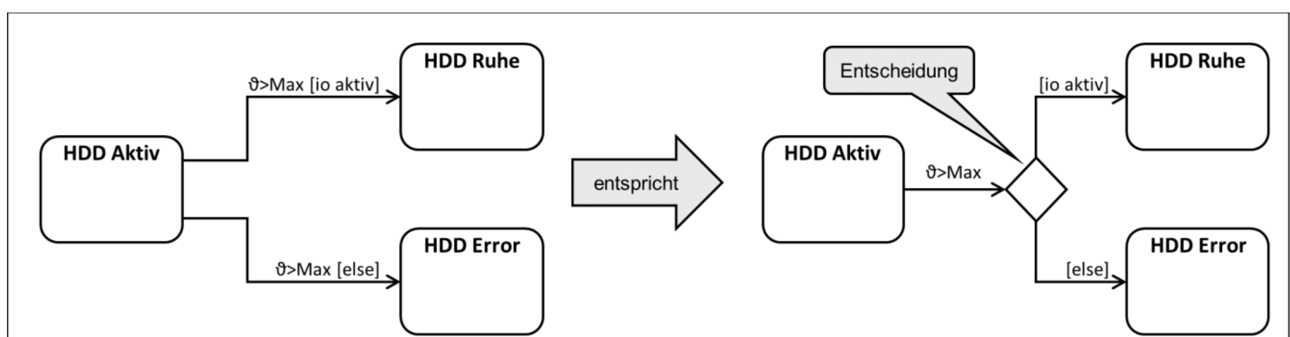
Events können auch einen **Guard** aufweisen (also einen Wächter), welcher eine **Bedingung** für die Ausführung darstellt. Bspw. würde *Stopp[io inaktiv]* bedeuten, dass die Transition nur dann durchlaufen wird, wenn die HDD gerade inaktiv ist (also weder schreibt, noch liest). Guards aller Transitionen aus einem Status müssen disjunkt sein (sie dürfen also keine Schnittmenge haben).

Weiterhin können auf einer Transition **Aktionen** hinterlegt werden, welche bei Durchlaufen der Transition ausgeführt werden – bspw. *ϑ>Max / Motor langsam* – also, wenn der Temperaturwert ϑ über den Maximalwert steigt, soll der Motor langsam werden.

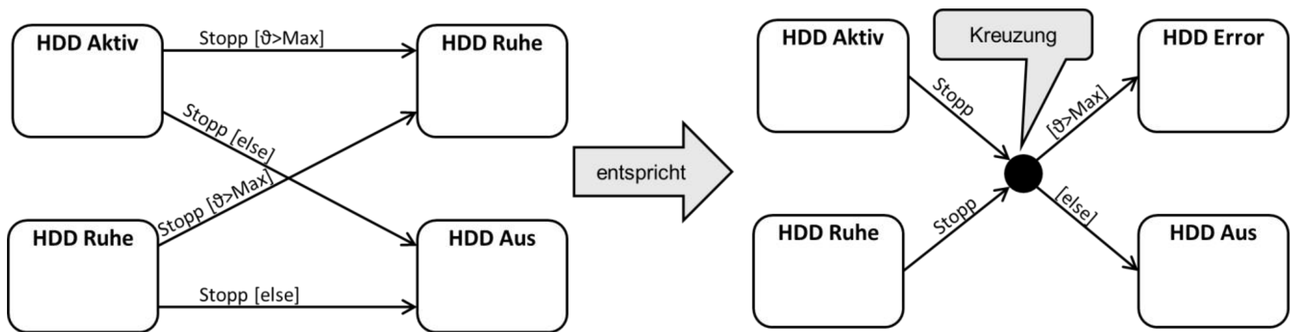


Selbsttransitionen gehen von einem Zustand aus und gehen wieder in denselben Zustand hinein. Wichtig zu verstehen ist hierbei, in welcher Reihenfolge welche Aktionen ausgeführt werden – vor allem im Zusammenhang mit Aktionen, welche innerhalb des Zustands definiert wurden.

Neben den klassischen Elementen gibt es noch zwei Notationselemente, welche die Zeichenkomplexität des Diagramms vereinfachen sollen. Hier ist zum einen die **Entscheidung**. Im unteren Bild sind beide Diagramme funktional identisch, wobei das rechte Bild schneller zu erfassen ist:

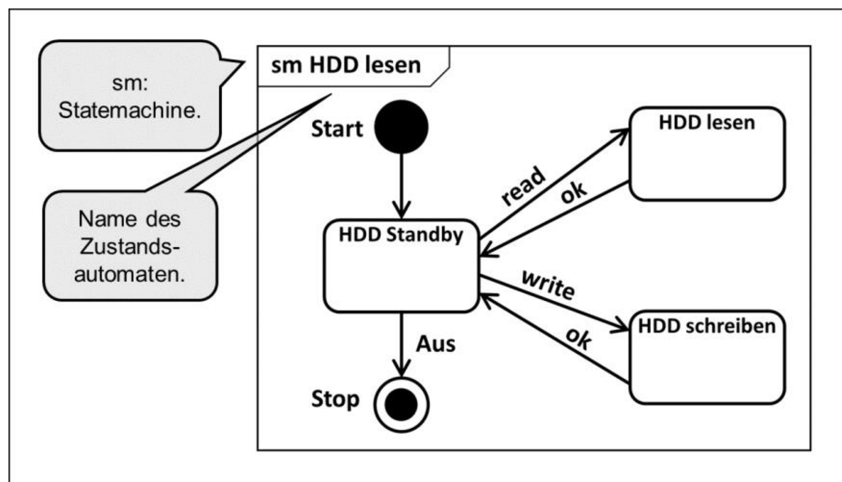


Die **Kreuzung** im unteren Bild vereinfacht ebenfalls die Interpretation. Hier sei angemerkt, dass hier die beiden Aktionen aus *HDD Aktiv* und *HDD Ruhe* nicht zwingend die gleichen sein müssen:



2.3 Rahmen

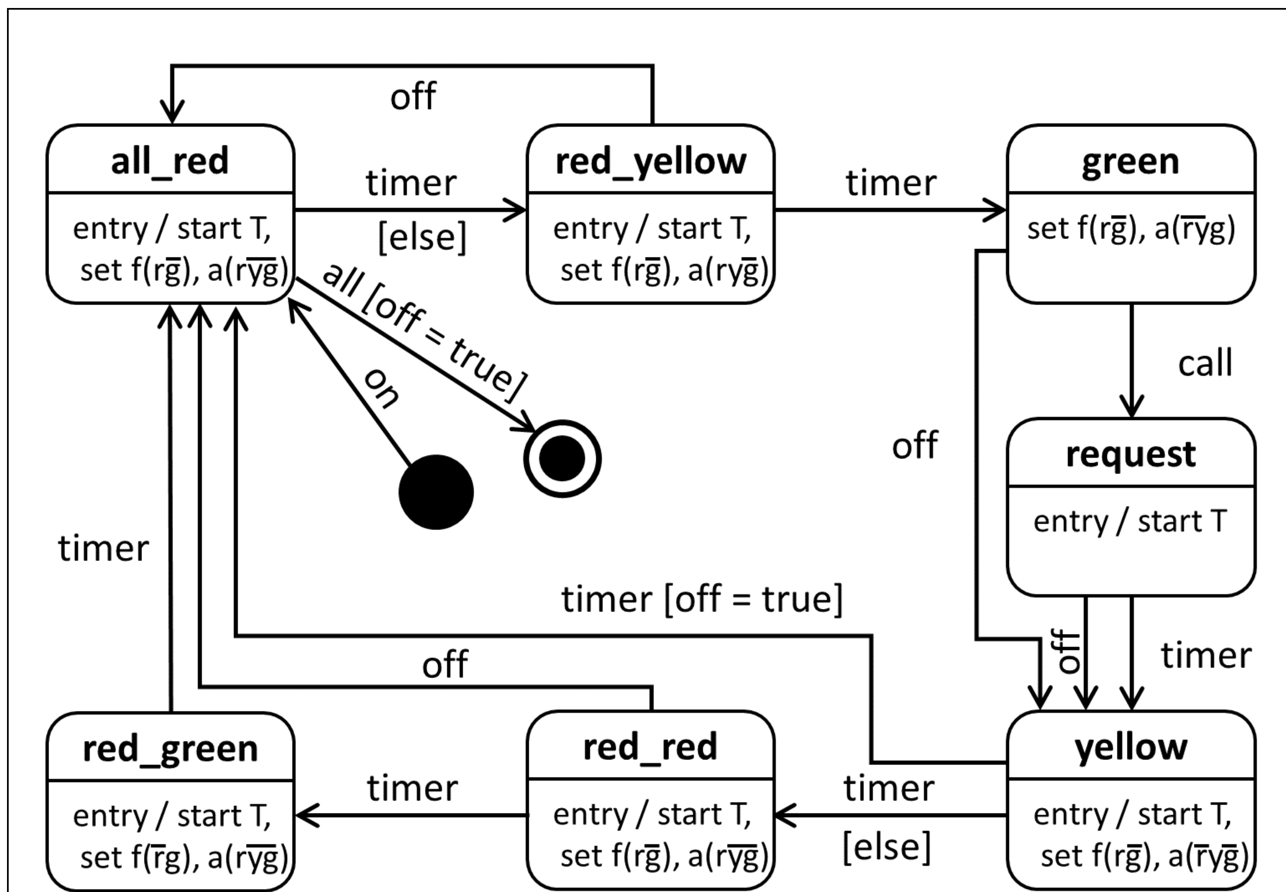
Die Zusammenfassung der zusammenhängenden Zustände und Transitionen wird in einem Rahmen erledigt. Dieser kann auch als Rahmen einer Ebene fungieren, welcher von anderen Ebenen aufgerufen wird. Herbei müssten noch Ein- und Ausgänge definiert werden, wobei dies hier nicht näher behandelt wird:



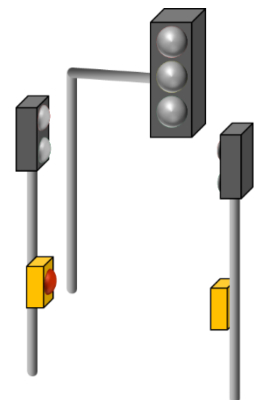
.

3 Beispiel

Das folgende Beispiel soll eine Fußgängerampel modellieren:



Die erwarteten Events sind:



- ON: Das System wird eingeschaltet.
- OFF: Das System wird ausgeschaltet. Hierbei soll nicht sofort alles ausgehen, sondern zuerst kontrolliert alles auf „*rot*“ und dann erst ausgehen.
- TIMER: Der Timer ist abgelaufen. Es wird in diesem Beispiel nur ein Timer verwendet, der aber unterschiedliche Laufzeiten umsetzen kann (auf die Darstellung der Laufzeiten wurde hier verzichtet).
- CALL: Der Fußgänger hat auf den Anfrageknopf für *grün* gedrückt.

Weiterhin wird in den Zuständen mit „*start T*“ der Timer gestartet, so dass er nach der eingestellten Zeit das TIMER Event auslöst. Die Aufrufe „*set f(rg)*“ setzt die Farben rot und grün der Fußgängerampel auf an (kein Hochstrich - Negation) oder aus (Negationsstrich). Gleiches gilt für die Autoampel mit den Farben *rot*, *grün* und *gelb* („yellow“).

Schließlich wird bei Eintreffen von „OFF“ noch ein inneres Flag gesetzt, welches in den Guards ausgewertet werden kann.

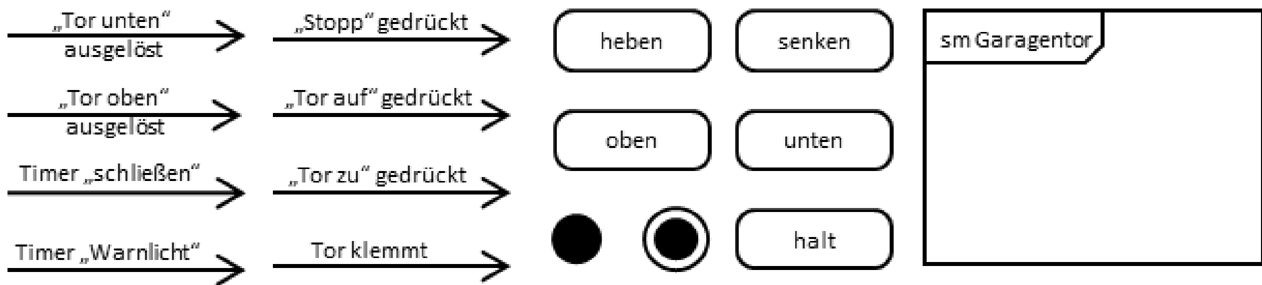
Die Funktionsweise und die JAVA Umsetzung wird in meinem YouTube Kanal nochmal erklärt:

https://www.youtube.com/channel/UCHuJP8SW1Rx6QhG7235E8-w

4 Aufgabenstellung

Zeichnen Sie ein UML Zustandsdiagramm für ein Garagentor, bei dem mit Hilfe von drei Tastern das Tor gesteuert werden soll. Beim Knopf „Tor auf" bzw. „Tor zu" soll es sich heben bzw. senken. Mit dem Knopf „Stopp" soll das Tor in den Haltezustand gehen. Weiterhin gibt es drei Endschalter, welche automatisch ein Signal geben wenn das Tor oben bzw. unten ist bzw. wenn das Tor mechanisch klemmt. Schließlich existieren noch zwei Timer. Der Timer „schließen" wird ausgewertet, wenn das Tor im Zustand „oben" ist. Er soll verhindern, dass das Tor zu lange offen bleibt. Der Timer „Warnlicht" löst aus, wenn das Tor im Zustand „unten" ist und soll die gesamte Anlage ausschalten (und somit auch ein Warnlicht, welches immer leuchtet, wenn die Anlage an ist).

Zeichnen sie das Zustandsdiagramm (als Protokollmaschine) und verwenden sie dabei folgende Elemente (wobei einige Pfeile mehrfach verwendet werden müssen):



5 Lizenz

Diese(s) Werk bzw. Inhalt von Maik Aicher (www.codeconcert.de) steht unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 3.0 Unported Lizenz.