

## Objektorientierte Modellierung

### Zustandsdiagramm



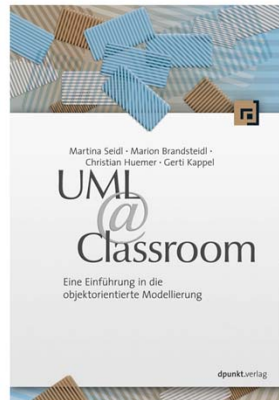
**Business Informatics Group**  
Institute of Software Technology and Interactive Systems  
Vienna University of Technology  
Favoritenstraße 9-11/188-3, 1040 Vienna, Austria  
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896  
office@big.tuwien.ac.at, www.big.tuwien.ac.at

Ich begrüße Sie zu unserer nächsten Vorlesung aus Objektorientierter Modellierung. Heute geht es um das Zustandsdiagramm.

## Literatur

---

- Die Vorlesung basiert auf folgendem Buch:



**UML @ Classroom:**  
**Eine Einführung in die objektorientierte Modellierung**  
*Martina Seidl, Marion Brandsteidl,  
Christian Huemer und Gerti Kappel*

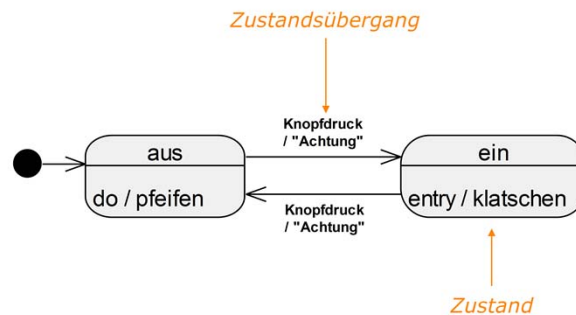
dpunkt.verlag

Juli 2012

ISBN 3898647765

- Anwendungsfalldiagramm
- Strukturmodellierung
- **Zustandsdiagramm**
- Sequenzdiagramm
- Aktivitätsdiagramm

## Bsp.: Zustandsdiagramm einer Lampe



Zu Beginn dieser Einheit möchte ich ein kleines Experiment mit Ihnen durchmachen. Ich werde Ihnen eine ganz einfache Zustandsmaschine vorstellen: Die Schreibtischlampe. Sie wechselt bei Knopfdruck von hell auf dunkel bzw. in die andere Richtung. Zusätzlich wird während der dunkel-Phase gepfiffen und beim Eingang in die hell-Phase wird kurz geklatscht. Ich werde jeden Knopfdruck mit einem „Achtung“ begleiten. Sehen wir uns das Ganze an:

Kurzes Klatschen (dann Stille) – „Achtung“ (Knopfdruck) – Pfeifen – „Achtung“ (Knopfdruck) – Kurzes Klatschen (dann Stille) – „Achtung“ (Knopfdruck) ...

Nun sehen wir uns das Zustandsdiagramm zu unserem Experiment an. In dem dargestellten Zustandsdiagramm gibt es zwei **Zustände**, nämlich das Licht ist „aus“ und das Licht ist „ein“. Zwischen diesen Zuständen gibt es **Zustandsübergänge**, die durch das Event „Knopfdruck“ ausgelöst werden. Ich kann im Vorhinein allerdings nicht sagen was passiert, wenn ich den Lichtschalter drücke, denn das ist abhängig davon ob das Licht „ein“ oder „aus“ ist. Im Zustandsdiagramm muss man sich also fragen, in welchem Zustand man sich befindet und in welchen Zustand man kommt, wenn ein gewisses Event eintritt. D.h. befinde ich mich im Zustand „aus“ wo es dunkel ist und drücke den Knopf, so wird es hell, ich gelange also in den Zustand „ein“. Ist es jedoch hell, d.h. die Lampe ist im Zustand „ein“ und ich drücke den Knopf, dann wird es dunkel, ich gehe also in den Zustand „aus“. D.h. dasselbe Event „Knopfdruck“ kann in unserem Beispiel in unterschiedlichen Zuständen vorkommen und führt auch jeweils in unterschiedliche Zustände. D.h. im Zeitpunkt des Events ist es wichtig in welchem Zustand man sich befindet, um daraus zu schließen, was als nächstes passiert.

Ein wesentliches Merkmal von Zustandsdiagrammen ist die Tatsache, dass es unbedeutend ist, wie man in einen gewissen Zustand gekommen ist. Sondern alleine aus der Tatsache, dass ich weiß, dass ich mich in einem bestimmten Zustand befinde, kann ich alle anderen Ereignisse, die bis dahin schon passiert sind, vergessen. Wenn ein gewisses Event eintritt, weiß ich auf Grund des aktuellen Zustands, was passiert. Wenn es dunkel ist und das Event „Knopfdruck“ tritt ein, wird es hell, egal wie ich zum Zustand dunkel bzw. „aus“ gekommen bin.

Wir haben also die wichtigsten Dinge besprochen: Es gibt Zustände und Zustandsübergänge die durch Events ausgelöst werden.

Des Weiteren kann es noch sein, dass im Rahmen von Zuständen oder Zustandsübergängen **Aktivitäten** ausgeführt werden.

Wenn das Licht „aus“ war, haben Sie alle gepfiffen. Das kann ich ausdrücken mit der Aktivität „Do/ pfeifen“. Das „Do/“ bedeutet: Während Sie in diesem Zustand sind, führen Sie die angegebene Aktivität kontinuierlich durch. Sie haben also die ganze Zeit gepfiffen. Sobald wir allerdings in den Zustand „ein“ übergegangen sind, haben Sie nur beim Eintritt, also nur einmal und nicht die ganze Zeit geklatscht. Das drückt man mit der Aktivität „Entry/ klatschen“ aus. Und außerdem habe ich jedes Mal beim Knopfdruck „Achtung“ gesagt. Das kann man mit „/Achtung“ beim Zustandsübergang „Knopfdruck“ ausdrücken.

Damit haben wir jetzt die wesentlichsten Elemente bereits kennengelernt: Zustände, Zustandsübergänge, Events die diese Zustandsübergänge auslösen und Aktivitäten die im Rahmen von Zuständen bzw. Zustandsübergängen durchgeführt werden.

## Inhalt

---

- Einführung
- Zustände und Aktivitäten
- Zustandsübergänge
- Ereignisarten
- Innere Transitionen
- Komplexe Zustände
  - Orthogonale Zustände
  - History-Zustände



© BIG / TU Wien



3

Hier haben wir einen Überblick über die Themen, die wir heute durchnehmen: Zuerst kommt eine kurze **Einführung** in die Zustandsdiagramme. Danach lernen wir **Zustände** und **Aktivitäten** die ausgeführt werden kennen. Anschließend folgen die **Zustandsübergänge**, die verschiedenen **Arten von Ereignissen** und dann noch **innere Transitionen**. Eine innere Transition bedeutet, dass ich den Zustand nicht verlasse, aber eine Aktivität durchführe wenn ein Event auftritt. Und dann lernen wir noch **komplexe Zustände** kennen was bedeutet, dass ich Zustände schachteln kann und zwar mittels AND oder OR. Auch die History-Zustände werden wir durchnehmen. Ein History-Zustände bedeuten folgendes: Wenn ich einen Zustand verlasse, der Subzustände besitzt, merke ich mir in welchem dieser Subzustände ich zuletzt war, bevor ich den übergeordneten Zustand verlassen habe und setze, falls ich in diesen Zustand wieder eintrete, bei jenem Subzustand fort.

## Einführung

- Ein Zustandsdiagramm (State Machine Diagram) beschreibt die möglichen **Folgen von Zuständen** eines **Modell-Elements**, i.A. eines Objekts einer bestimmten Klasse
  - Während seines **Lebenslaufs** (Erzeugung bis Destruktion)
  - Während der **Ausführung** einer **Operation** oder **Interaktion**
- Modelliert werden
  - Die **Zustände**, in denen sich die Objekte einer Klasse befinden können
  - Die möglichen **Zustandsübergänge** (Transitionen) von einem Zustand zum anderen
  - Die **Ereignisse**, die Transitionen auslösen
  - **Aktivitäten**, die in Zuständen bzw. im Zuge von Transitionen ausgeführt werden



Ein **Zustandsdiagramm** oder auch State Machine Diagram, Zustandsautomat oder Zustandsmaschine genannt, beschreibt die **mögliche Folge von Zuständen eines Modell-Elements**. In der Objektorientierung handelt es sich normalerweise um die **Zustände eines Objekts** einer gewissen Klasse. Dementsprechend ordne ich ein Zustandsdiagramm auch der Klasse zu von der der typische Lebenszyklus eines Objekts beschrieben wird. Sehr oft beschreibt man den **Lebenszyklus** eines Objekts von der Konstruktion bis zur Destruktion. Man könnte aber auch die Zustände des Objekts während der **Durchführung einer einzelnen Operation oder Interaktion** beschreiben.

Modelliert werden die **Zustände**, wie „ein“ oder „aus“; die **Zustandsübergänge** wie von „ein“ nach „aus“; die **Ereignisse**, die die Transition auslösen, wie „Knopfdruck“; und **Aktivitäten** wie „pfeifen“, „klatschen“ oder „Achtung sagen“.

## Einführung – Beispiel Digitaluhr (1/2)

- Modelliert werden sollen die Zustände, die eine Digitaluhr beim Stellen der Uhr einnehmen kann.
- Die Uhr kann 3 Zustände einnehmen:
  - Zeit anzeigen
  - Stunden einstellen
  - Minuten einstellen



Digitaluhr	
-	min: int
-	stunden: int
+	set() : void
+	inc() : void

- Durch die Betätigung des Einstellungsknopfes wird von "Zeit anzeigen" in "Stunden einstellen" gewechselt, von "Stunden einstellen" in "Minuten einstellen" und schließlich von "Minuten einstellen" wieder in "Zeit anzeigen".
- Wird in "Stunden einstellen" gewechselt, piepst die Uhr und zeigt die Stunden an. Durch die Betätigung des inc-Buttons, wird die Stundenanzahl um 1 erhöht.
- "Minuten einstellen" funktioniert analog.
- Am Anfang befindet sich die Uhr im Zustand "Stunden einstellen".

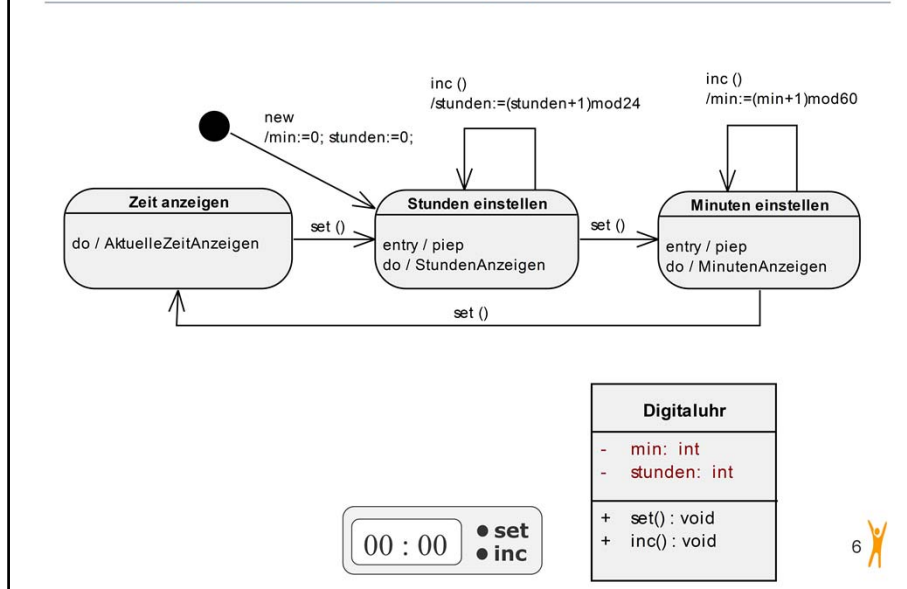


Wir kommen zu einem einfachen Beispiel.

Wir haben hier eine Klasse „Digitaluhr“. Diese Klasse besitzt zwei Attribute „Minuten“ und „Stunden“, diese sind „private“. Und auf der Uhr gibt es zwei Knöpfe, einen „set“-Knopf mit dem ich den Modus wechsele und einen „increment“-Knopf mit dem ich die Stunden oder die Minuten erhöhen kann. Mehr Knöpfe habe ich auf dieser einfachen Digitaluhr nicht zur Verfügung. Dementsprechend hat die Klasse nur die beiden Operationen „set()“ und „inc()“, die dem Drücken der entsprechenden Knöpfe entsprechen.

Wir kennen drei verschiedene Modi bzw. Zustände, nämlich „Zeit anzeigen“, „Stunde einstellen“ und „Minuten einstellen“. Jedes Mal wenn ich den Knopf „set“ drücke, wechsele ich zwischen diesen drei Zuständen. Jedes Mal wenn ich in den Zustand „Stunden einstellen“ oder „Minuten einstellen“ wechsele, soll die Uhr piepsen. Wenn ich mich im Zustand „Stunden einstellen“ befinde und „increment“ drücke, wird die Stunde um 1 erhöht. Befinde ich mich im Zustand „Minuten einstellen“, werden die Minuten um 1 erhöht. Wenn ich in „Zeit anzeigen“ bin und „increment“ aufrufe, soll nichts passieren. D.h. es gibt auch Events die verloren gehen, wenn sie auftreten. Am Anfang befindet sich die Uhr im Zustand „Stunden einstellen“.

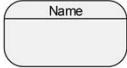


## Einführung – Beispiel Digitaluhr (2/2)



Wir haben hier im Zustandsdiagramm also drei Zustände, „Zeit anzeigen“, „Stunden einstellen“ und „Minuten einstellen“. Wir beginnen mit dem Zustand „Stunden einstellen“. Das wird markiert mit einem Anfangszustand, einem schwarzen Kreis. Der Anfangszustand ist ein Pseudozustand, denn ich kann in diesem nicht verweilen. D.h. unmittelbar nachdem der Zustandsautomat aktiv wird, gehe ich in den Zustand „Stunden einstellen“ und beim Übergang werden die Minuten und Stunden auf 0 gesetzt. D.h. nach dem Slash (/) werden immer die Aktivitäten angegeben, die ausgeführt werden sollen. Ich gehe also zunächst in „Stunden einstellen“ und beim Eintritt „entry“ piepst die Uhr. Während ich in „Stunden einstellen“ bin, werden permanent die Stunden angezeigt. Wenn ich den Knopf „increment“ drücke, verlasse ich den Zustand, kehre aber wieder in denselben Zustand zurück. Das Ausführen der Operation „increment“ ist ein sogenanntes CallEvent, wir lernen die Eventarten noch kennen, und im Zuge des Zustandsübergangs wird die Aktivität „stunden = (stunden + 1) mod 24“ ausgeführt. D.h. jedes Mal wenn ich hier „increment“ drücke, wird die Stunde um 1 erhöht und ich kehre in „Stunden einstellen“ zurück, was auch bedeutet, dass jedes Mal wenn ich „increment“ drücke, die Uhr piepst und die Stunden angezeigt werden. Drücke ich im Zustand „Stunde einstellen“ die Taste „set“, gehe ich in den Zustand „Minuten einstellen“ über. Dieser Zustand funktioniert genauso wie „Stunde einstellen“, d.h. beim Eintritt wird gepiepst, die Minuten werden angezeigt und beim Drücken von „increment“ werden die Minuten um 1 erhöht und wieder kommt man in den Zustand „Minuten einstellen“, es wird gepiepst etc. Und wenn ich hier „set“ aufrufe, gehe ich über in den Zustand „Zeit anzeigen“. Wenn hier „increment“ aufgerufen wird, passiert nichts. Denn im Zustand „Zeit anzeigen“ ist mit dem Event „increment“ kein Zustandsübergang verbunden. Erst wenn ich wieder „set“ drücke, wird ein Zustandsübergang ausgelöst und zwar nach „Stunden einstellen“.

Das war ein einfaches Beispiel für einen Zustandsautomaten.

## Zustand

- („echter“) Zustand (state)  
System kann sich dauerhaft im Zustand befinden
  - Zustand im eigentlichen Sinn  

  - Endzustand  

- Pseudozustand  
transient (System kann nicht dauerhaft in einem Pseudozust. sein)
  - Startzustand  

  - Flacher/Tiefer History-Zustand
  - Parallelisierungsknoten & Synchronisierungsknoten
  - Terminierungsknoten
  - Entscheidungsknoten



© BIG / TU Wien



7

Es gibt also **echte Zustände**. Ein System kann sich dauerhaft in einem echten Zustand befinden. Das sind die **Zustände im eigentlichen Sinne**. Diese werden dargestellt mit einem Viereck mit abgerundeten Kanten.

Weiters gibt es den **Endzustand**. Er ist auch ein echter Zustand in dem ich verweilen kann. Z.B. könnte ich im vorherigen Beispiel der Digitaluhr ein Event „Mit Hammer auf Oberfläche schlagen“ einführen und dann würde ich wahrscheinlich egal in welchem Zustand ich mich befinde in den Endzustand übergehen. Im Endzustand kann man verweilen, es handelt sich also nicht um einen Pseudozustand. Das ist im Gegensatz zum Aktivitätsdiagramm, wo der Endzustand einen Pseudozustand darstellt.

Dann gibt es eben noch **Pseudozustände**. Dazu gehört der Startzustand, der in jedem Zustandsdiagramm vorkommt. Damit wird explizit festgelegt, mit welchem Zustand man beginnt. Der Startzustand kann in Sonderfällen auch fehlen. In den Übungen müssen Sie den Anfangszustand aber immer einzeichnen. Weiters gibt es noch den flachen und den tiefen History-Zustand, die wir am Ende der Vorlesung kennenlernen werden. Wir haben Parallelisierungsknoten und Synchronisierungsknoten, die im Zustandsdiagramm eher unüblich sind, die wir aber auch in der Vorlesung durchnehmen werden. Außerdem gibt es den Terminierungsknoten der im Zustandsdiagramm auch nicht von großer Bedeutung ist und den Entscheidungsknoten, der verwendet werden darf – wovon ich allerdings abrate, warum das so ist, dazu kommen wir auch noch.



## Aktivitäten innerhalb eines Zustands

- **entry / aktivität**
  - Wird beim Eingang in den Zustand ausgeführt
- **exit / aktivität**
  - Wird beim Verlassen des Zustands ausgeführt
- **do / aktivität**
  - Wird ausgeführt, Parameter sind erlaubt
- **event / aktivität**

Aktivität behandelt Ereignis innerhalb des Zustands

  - Wird ausgeführt, wenn sich das System in dem Zustand befindet und das Ereignis eintritt



Während wir uns in einem Zustand befinden können wir **Aktivitäten** durchführen. Dabei gibt man das Event an, bei dessen Auftreten eine Aktivität durchgeführt wird. Es gibt drei vordefinierte Events, nämlich:

„**entry**“: Sobald ich in den Zustand eintrete wird die Aktivität ausgeführt, wie das Piepsen bei unserer Uhr.

Dann gibt es „**do**“: Die angegebene Aktivität wird kontinuierlich während des Zustands ausgeführt, könnte aber auch von der Dauer beschränkt sein. Z.B. könnten Sie sich an einem Kaffee-Automaten befinden und der Automat kennt den Zustand „Kaffeebecher befüllen“. In diesem Zustand führt er die Aktivität des Kaffeebecher-Befüllens kontinuierlich als „do“-Aktivität aus, aber nur solange, bis der Becher voll ist.

Als Gegenstück zu „entry“ gibt es auch das Event „**exit**“: So könnte man ja auch spezifizieren, dass die Uhr nicht dann piepst wenn man einen Zustand betritt, sondern dann, wenn man ihn verlässt. D.h. tritt ein Event ein, das zu einem Zustandsübergang führt, so wird vor Verlassen des Zustands diese Aktivität ausgeführt.

Weiters kann man eigene Events angeben, die eintreten sollen, damit eine gewisse Aktivität ausgeführt wird. Dazu wird nicht „**event**“ angegeben, sondern der Name des Events, wie z.B. „Knopfdruck“ oder „mouseOver“. D.h. „entry“, „do“ und „exit“ werden auch tatsächlich mit diesen Strings angegeben, bei „event“ muss ich das tatsächliche Event angeben. Bei „event“ behandelt die Aktivität das Ereignis innerhalb des Zustands und wird ausgeführt, wenn sich das System in dem Zustand befindet und das Ereignis eintritt. Der Zustand wird dann aber nicht verlassen, sondern die Aktivität wird nur ausgeführt. „event“ bezeichnet also innere Transitionen, die wir auch noch später in einer weiteren Folie behandeln werden.

## Zustandsübergang (1/2)

- Ein Zustandsübergang (Transition) erfolgt, wenn
  - das **Ereignis** eintritt
    - eine evt. noch andauernde **Aktivität** im Vorzustand wird **unterbrochen!**
  - und die **Bedingung** (guard) **erfüllt** ist
    - bei Nicht-Erfüllung geht das nicht »konsumierte« Ereignis verloren  
=> wenn die Bedingung erst zu einem späteren Zeitpunkt erfüllt wird, kann die Transition ohne neuerliches Ereignis nicht durchgeführt werden
- Durch entsprechende Bedingungen können **Entscheidungsbäume** modelliert werden



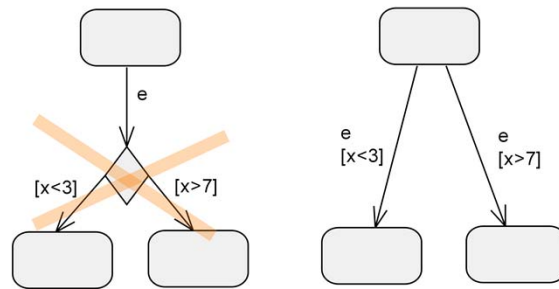
Wir kommen zu den **Zustandsübergängen** oder auch Transitionen genannt. Ein Zustandsübergang bzw. eine Transition erfolgt, wenn das Ereignis eintritt, das diesen Zustandsübergang definiert. Eine eventuell noch andauernde Aktivität im Vorzustand wird unterbrochen. Das haben wir schon bei unserem Anfangsbeispiel gesehen. Es war dunkel und Sie haben gepfiffen. Ich habe den Zustandsübergang nach hell bzw. „ein“ ausgelöst und Sie haben aufgehört zu pfeifen. D.h. also eine eventuell noch andauernde Aktivität im Vorzustand wird einfach abgebrochen.

Was wir bisher noch nicht hatten sind **Guards** oder **Bedingungen**, die zusätzlich noch überprüft werden müssen. D.h. man kann angeben, dass wenn in einem Zustand ein gewisses Event eintritt, der Zustandsübergang nur erfolgt, wenn beim Eintreten dieses Events auch eine bestimmte Bedingung erfüllt ist. Wenn man sich z.B. einen Bankomat vorstellt, so gibt man einen Betrag ein, den man beheben möchte. Durch Bestätigen dieses Betrags, sollte der Bankomat in den nächsten Zustand übergehen, nämlich in „Betrag auszahlen“, jedoch nur dann, wenn das Konto nicht überzogen ist. D.h. man kann definieren, dass man vom Zustand „Betrag eingeben“ in den Zustand „Betrag auszahlen“ zwar einen Zustandsübergang mit dem „Bestätigen“-Knopf auflöst, jedoch zusätzlich noch die Bedingung erfüllt sein muss, dass das Konto nicht überzogen ist.

Wenn es zu einem Event nur eine Bedingung gibt, so bedeutet das, dass wenn diese Bedingung nicht erfüllt ist, man in dem vorhergehenden Zustand bleibt. Man kann mit mehreren Bedingungen für dasselbe Event natürlich auch steuernd eingreifen wie in dem Beispiel das hier dargestellt ist. Es tritt bei einem Zustand ein Event ein und es gibt zwei mögliche Folgezustände, z.B. beim Bankomat „Auszahlung starten“ und „Kunde informieren“, also sich wechselseitig ausschließende Bedingungen für einen Zustandsübergang. Wie in unserem Beispiel links wo ich sehe, wir sind in einem Zustand, das Event „e“ tritt ein, ich komme zu einem Entscheidungsknoten und ich habe hier wechselseitig ausschließende Bedingungen [B] und [NOT B]. Damit ist gesichert, dass ich entweder nach rechts oder nach links gehe. Somit kann man die Zustandsfolge über diese Bedingungen steuern.

Es gibt also im Zustandsdiagramm die **Entscheidungsknoten** mit denen man Entscheidungsbaume modellieren kann. Ich selbst bin kein Anhänger davon. Ich selbst bevorzuge die Vorgehensweise auf der rechten Seite. Hier wird kein Entscheidungsknoten eingesetzt. Vom Quellzustand werden zwei ausgehende Zustandsübergänge durch dasselbe Event ausgelöst, aber mit wechselseitig ausschließenden Bedingungen. Die Darstellung rechts ist äquivalent zur linken Darstellung. Die Probleme bei der linken Darstellung werden wir auf der nächsten Folie kennenlernen.

## Zustandsübergang (2/2)

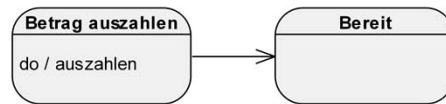


Es kann nämlich bei den Entscheidungsknoten ein Problem auftreten, wenn die Guard-Bedingungen zwar gegenseitig ausschließend sind, aber nicht den gesamten Lösungsraum abdecken. Wenn ich einen schlechten Ablauf modelliere starte ich z.B. wie hier dargestellt beim oberen Zustand, wenn das Event eintritt gehe ich aus dem Zustand hinaus in den Entscheidungsknoten. Wenn dann die Guard-Bedingungen  $[x < 3]$  und  $[x > 7]$  sind, dann gibt es für den Bereich von 3-7 keinen Zustandsübergang und damit bleibt der Zustandsautomat beim Entscheidungsknoten hängen, wenn einer der Werte zwischen 3 und 7 für  $x$  auftritt. Die rechte Darstellung ist äquivalent zu der Darstellung mit Entscheidungsknoten. D.h. ich kann auch aus einem Zustand mehrere ausgehende Zustandsübergänge haben, die alle dasselbe Event spezifizieren, jedoch müssen diese wechselseitig ausschließende Bedingungen anführen. Und dementsprechend habe ich rechts das gleiche abgebildet wie links. Es kann mir aber nie passieren, dass ich in einem Entscheidungsknoten hängen bleibe. Auch nicht, wenn ich zwar wechselseitig ausschließende Bedingungen habe, diese aber nicht den ganzen Lösungsraum abdeckenden. Denn wenn  $x$  kleiner 3 ist, gehe ich auf die linke Seite, wenn  $x$  größer 7 ist, gehe ich auf die rechte Seite und bei einem Wert zwischen 3 und 7 bleibe ich im Ausgangszustand. Und nachdem man sich ein Modellierungselement spart und das Modell dadurch besser zu lesen ist, verwende ich persönlich niemals Entscheidungsknoten, weil ich mit der anderen Darstellungsform dasselbe modellieren kann und zusätzlich noch Bedingungen behandeln kann, die zu keinem Zustandsübergang führen sollen.

Wichtig ist auch, dass bei **Nicht-Erfüllung der Bedingung** das nicht konsumierte Ereignis verloren geht. In unserem Beispiel haben Sie am Bankomat kein Geld bekommen weil ihr Konto überzogen war. Wenn jetzt zu einem späteren Zeitpunkt das Konto wieder positiv wird weil man ja etwas eingezahlt hat, wird aber beim Bankomat trotzdem nicht einfach Geld ausgegeben. Wenn also ein Event auftritt, ein Zustandsübergang aber nicht möglich ist weil z.B. die Bedingung  $[x > 7]$  nicht erfüllt ist, dann andere Operationen durchgeführt werden und dadurch  $x$  größer als 7 wird, ist der Zustandsübergang dennoch nicht möglich, denn bei Nicht-Erfüllung einer Bedingung zum Zeitpunkt des Events, geht dieses Event verloren. Ich muss also warten, bis das Event wieder eintritt und die Bedingungen dann auch wieder überprüfen.

## Default Werte für Zustandsübergänge

- **Default-Werte**
  - Fehlendes Ereignis entspricht dem Ereignis »Aktivität ist abgeschlossen«
  - Fehlende Bedingung entspricht der Bedingung [true]
- **Beispiel: Bankomat**



Es gibt für fehlende Ereignisse/Events bei Zustandsübergängen und bei Bedingungen **Default-Werte**. Meistens verzichten wir in dieser Lehrveranstaltung auf die Nutzung der Default-Werte.

Ein **fehlendes Ereignis** entspricht dem Ereignis „**Aktivität ist abgeschlossen**“. Das macht nur dann Sinn, wenn auf einem Zustandsübergang kein Ereignis angegeben ist. Es sind also zwei Zustände mittels eines Zustandsübergangs miteinander verbunden. Wir haben vorher kennengelernt, dass ein Zustandsübergang durch ein Event ausgelöst wird. D.h. prinzipiell, dass für jeden Zustandsübergang ein Event angegeben werden muss. Ich bitte Sie in der Übung immer ein Event anzugeben. Der Default-Wert lautet hier: Ein fehlendes Ereignis entspricht dem Ereignis „Aktivität ist abgeschlossen“. Wenn wir wieder an den Bankomat denken, so könnte er sich im Zustand „Betrag auszahlen“ befinden. Nachdem er das Geld ausgegeben hat ist dieser Zustand eigentlich abgeschlossen und der Bankomat wechselt in den Zustand „Bereit“. Und in diesem Fall gebe ich vielleicht kein Event an, sondern der Bankomat hat im Zustand „Betrag auszahlen“ eine Aktivität durchzuführen und ich nehme an, dass wenn diese Aktivität abgeschlossen ist, der Zustand verlassen wird und der Bankomat in den nächsten Zustand „Bereit“ übergeht.

Dann gibt es noch den Default-Wert für Bedingungen. D.h. es wird ein Event angegeben, jedoch keine Bedingung, so löst das Event auf jeden Fall den Zustandsübergang aus. D.h. die Bedingung wird nicht überprüft, denn der Default-Wert für die Bedingung lautet einfach „true“.

## Syntax von Zustandsübergängen

- Ereignisse, Bedingungen und Aktivitäten auf Zustandsübergängen möglich
- Notation: Ereignis (Argumente) [Bedingung] / Aktivität
  - Die Aktivität kann aus mehreren Aktionen bestehen
  - Spezielle Aktivität: Nachricht an anderes Objekt senden  
send empfänger.nachricht()
- Beispiel:

*Ereignis* *Bedingung*

```
right-mouse-button-down (loc) [ loc in window ]  
/ obj:= pick-obj (loc); send obj.highlight()
```

*Aktion 1* *Aktion 2*



© BIG / TU Wien



12

Hier haben wir die **Syntax** in der wir **Zustandsübergänge** angeben dargestellt. Die ist immer dieselbe, bitte prägen Sie sich diese ein. Als erstes erfolgt die Angabe des Ereignisses. Wenn dieses Ereignis Parameter besitzt, wie etwa ein Operationsaufruf, dann werden diese Parameter in Klammern () angegeben. Nach dem Ereignis gebe ich innerhalb der eckigen Klammern [] eine Überwachungsbedingung an, die erfüllt sein muss, wenn das Event eintritt, um zu einem Zustandsübergang zu führen. Und dann gibt man noch nach dem Slash / die Aktivität bzw. die Aktivitäten an, die ausgeführt werden sollen wenn das Ereignis eintritt und die Bedingung erfüllt ist. D.h. die Syntax ist immer dieselbe. Es gibt jedoch keine spezifische Syntax für die Angabe der Aktivitäten.

Wenn wir uns das Beispiel hier anschauen: Wir haben „right-mouse-button-down“ als Ereignis, Parameter ist die Location in der die rechte Maustaste gedrückt wird. Die Bedingung in den eckigen Klammern [] lautet, dass sich diese Location im Window befinden muss. Und dann nach dem Slash / wird angegeben, was ausgeführt werden soll, wenn das Event eintritt und die Bedingung erfüllt ist. Hier haben wir zwei Befehle angegeben, die nacheinander ausgeführt werden sollen und deshalb durch einen Strichpunkt ; voneinander getrennt sind. Die Syntax wie mehrere Aktivitäten angegeben werden, ist hier durch die Programmiersprache dominiert, dafür gibt es keine bestimmte Syntax von UML.

Beherrsigen Sie bitte immer die Syntax zur Angabe von Zustandsübergängen. Es passiert bei den Übungen durchaus, dass die Studenten die Teile vermischen. Die Reihenfolge lautet also Ereignis-Bedingung-Aktivität.

## Zustandsübergang: Ereignistypen (1/2)

- **CallEvent**  
Empfang einer Nachricht (Operationsaufruf)
  - Bsp.: stornieren(), kollidiertMit(Termin)
- **SignalEvent**  
Empfang eines Signals
  - Bsp.: right-mouse-button-down, ok-Taste-gedrueckt
- **ChangeEvent**  
Eine Bedingung wird wahr
  - Bsp.: when(x<y), when(a=1), when(terminBestaetigt)
- **TimeEvent**  
Zeitablauf oder Zeitpunkt
  - Bsp.: after(5 sec.), when(date=31.01.2008)



Es gibt verschiedene Arten von Zustandsübergängen. Prägen Sie sich auch diese ein.

Der erste Typ ist das **CallEvent**. Hier wird von der Klasse eine Operation aufgerufen. Das hatten wir in unserem Uhrenbeispiel. Hier hatten wir eine Klasse „Digitaluhr“ und diese hatte zwei Methoden: „set()“ und „increment()“. Genau diese beiden Methoden wurden im Zustandsdiagramm aufgerufen. Das wären CallEvents, die an diese Klasse gerichtet werden. Wir hatten im ersten Jahr der OOM-Vorlesung ein zusammenfassendes Beispiel in dem ein Klassendiagramm gegeben war und die Aufgabenstellung lautete, man soll ein Zustandsdiagramm mit CallEvents modellieren. Das bedeutet natürlich, dass für die Zustandsübergänge nur die Operationen der Klasse aus dem Klassendiagramm verwendet werden sollten. 80% der Studierenden hatten diese Bedingung nicht berücksichtigt. D.h. wenn in einer Angabe gefordert ist, dass CallEvents verwendet werden müssen, dann sollten die Operationen, die die Ereignisse darstellen, auch als Operationen im Klassendiagramm vorhanden sein.

Das **SignalEvent** zeigt den Empfang eines Signals an. Das hatten wir im Beispiel mit dem Lichtschalter. Hier wird das Signal abgefangen, dass der Knopf gedrückt wurde und es wird entsprechend hell oder dunkel bzw. „ein“ oder „aus“.

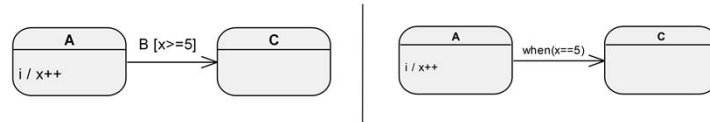
Dann haben wir das **ChangeEvent**. Das behandeln wir auf der nächsten Folie noch intensiver. Ein ChangeEvent horcht sozusagen permanent ob eine spezifizierte Bedingung eintritt und wenn dies der Fall ist, dann wird ein Zustandsübergang ausgelöst bzw. das Event feuert.

Als letztes gibt es noch das **TimeEvent**. Hier können Zeiten angegeben werden. D.h. nach einer gewissen Periode oder zu einem gewissen Zeitpunkt, d.h. after(periode) oder when(zeitpunkt), wird ein Zustandsübergang ausgelöst, bzw. das Event feuert.



## Zustandsübergang: Ereignistypen (2/2)

- Unterschied ChangeEvent und Bedingung
- **ChangeEvent:**
  - Bedingung wird permanent geprüft
  - wenn Bedingung wahr ist, kann zugehöriger Zustandsübergang ausgelöst werden (falls nicht durch zugehörige Überwachungsbedingung blockiert)
- **Bedingung:**
  - wird nur geprüft, wenn zugeordnetes Ereignis eintritt
  - kann selbst keinen Zustandsübergang auslösen



© BIG / TU Wien



Oft haben die Leute Schwierigkeiten beim Unterscheiden zwischen ChangeEvent und Überwachungsbedingung. Der Unterschied zwischen einem ChangeEvent und einer Überwachungsbedingung ist der folgende. Eine **Überwachungsbedingung** braucht immer ein Event das eintritt und wenn dieses Event eintritt, dann wird die Bedingung dazu geprüft. Ist diese Bedingung erfüllt erfolgt der Zustandsübergang, ist diese Bedingung nicht erfüllt so bleibt man im Zustand. Umgekehrt braucht das **ChangeEvent** kein anderes Event, denn es ist selbst ein Event das eintritt damit ein Übergang erfolgt.

Ich möchte das an einem Beispiel veranschaulichen.

Ich habe hier zwei Zustände „A“ und „C“ und einen Zustandsübergang von „A“ nach „C“ mit dem Event „B“ und der Bedingung  $[x \geq 5]$ . Haben wir im Moment  $x=4$  und das Event „B“ tritt ein, so ist die Bedingung nicht erfüllt, da  $x$  nicht größer als 5 ist und daher bleibe ich im Zustand „A“. Dann tritt wieder „B“ auf,  $x$  ist noch immer noch nicht größer als 5, damit passiert nichts. Im Zustand „A“ wird  $x$  um 1 erhöht wenn das Event „i“ eintritt. Dann tritt „i“ ein und jetzt ist  $x=5$ . Aber es passiert solange nichts, bis nicht „B“ eintritt. Erst wenn „B“ wieder eintritt wird die Bedingung  $[x \geq 5]$  wieder überprüft und der Zustandsübergang wird ausgelöst.

Im Unterschied dazu, betrachten wir auf der rechten Seite einen Zustandsübergang mit dem ChangeEvent  $\text{when}(x==5)$ . Ist  $x$  ursprünglich 4, und erfolgt dann das Event „i“, so wird  $x$  auf 5 erhöht und jetzt horcht das ChangeEvent permanent und stellt fest, dass  $x=5$  ist und damit wird der Zustandsübergang unmittelbar im Anschluss an die Durchführung des Events „i“ ausgelöst. Es handelt sich hier also um ein ChangeEvent und das horcht permanent ob eine Änderung auftritt.

## Start- u. Endzustand, Terminierungsknoten

### Startzustand

- "Beginn" des Zustandsdiagramms
- keine eingehenden Transitionen
- genau eine ausgehende Transition
  - wird sofort ausgelöst, wenn sich das System im Startzustand befindet
  - keine Bedingungen und Ereignisse (Ausnahme: Ereignis zur Erzeugung des betrachteten Objekts)
  - Angabe von Aktivitäten ist erlaubt



### Endzustand

- keine ausgehenden Transitionen
- kein Pseudozustand!



### Terminierungsknoten

- Objekt, dessen Verhalten modelliert wird, hört auf zu existieren



© BIG / TU Wien

15



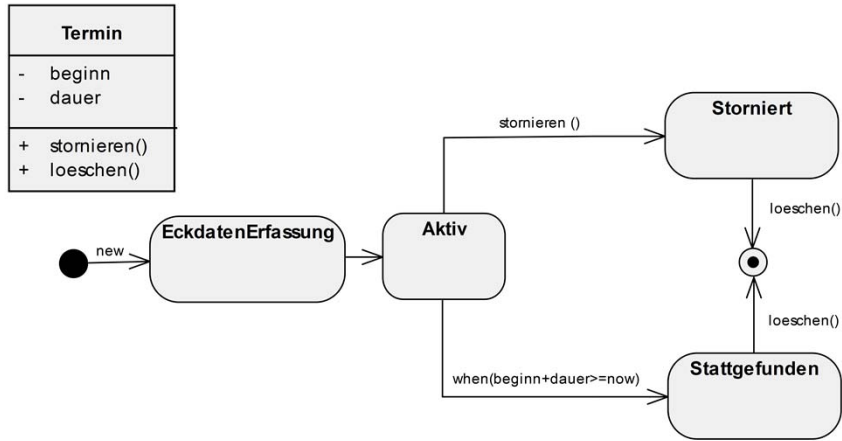
Wir kommen zum **Startzustand**. Er steht am Beginn des Zustandsdiagramms und hat somit keine eingehenden Kanten. Er hat im Zustandsdiagramm genau eine ausgehende Transition. Sie wird sofort ausgelöst, wenn sich das System im Startzustand befindet, d.h. ich kann nicht im Startzustand verweilen, deshalb ist der Startzustand ein Pseudozustand. Diese Transition besitzt auch keine Bedingungen und Ereignisse mit Ausnahme des Ereignisses zur Erzeugung des betrachteten Objekts, also sozusagen ein „new“. Die Angabe von Aktivitäten ist erlaubt. Das hatten wir auch in unserem Digitaluhr-Beispiel. Am Start erzeugten wir ein Objekt mit „new“ und in den Aktivitäten wurden die Stunden und Minuten auf 0 gesetzt.

Dann kommen wir zum **Endzustand**. Dieser besitzt keine ausgehenden Transitionen, da wir ja am Ende unseres Zustandsdiagramms angelangt sind. Er ist kein Pseudozustand, weil ich im Endzustand verweilen kann. In einem Zustandsdiagramm kann es durchaus mehrere Endzustände geben.

Dann gibt es noch den **Terminierungsknoten**, der selten eingesetzt wird. Er bedeutet, dass das Objekt, dessen Verhalten modelliert wird, aufhört zu existieren. D.h. das Objekt ist nicht nur am Ende der Laufzeit angekommen, sondern es wird, in Programmier-Analogie ausgedrückt, auch der Destruktor aufgerufen.

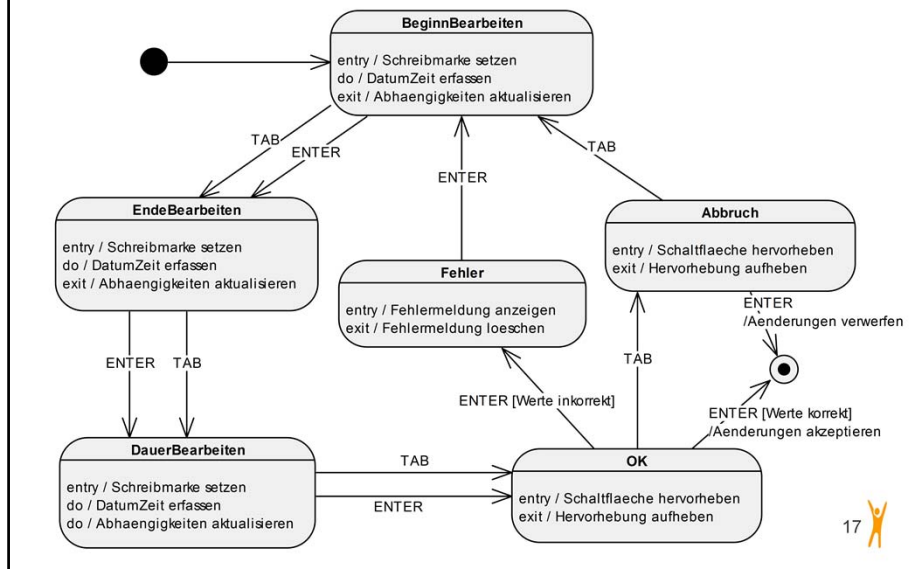


### Bsp.: Lebenszyklus eines Termins im CALENDARIUM



Wir haben hier das Objekt eines Termins. Der Termin wird neu angelegt, dann befinden wir uns im Zustand „EckdatenErfassung“. Nachdem die Aktivität in diesem Zustand abgeschlossen wurde, befindet sich der Termin im Zustand „Aktiv“. D.h. wenn die Eckdaten erfasst sind, d.h. die enthaltene Aktivität abgeschlossen ist, dann geht der Zustand des Termins über in „Aktiv“. Wird dann „stornieren()“ aufgerufen, geht der Zustand in „Storniert“ über. Wenn jedoch im Zustand „Aktiv“ der Umstand eintritt, dass der Beginn plus die Dauer größer als der momentane Zeitpunkt ist (`when(beginn+dauer>=now)`), ist der Termin vorbei und er geht in den Zustand „Stattgefunden“ über. Dann kann man den Termin auch nicht mehr stornieren, denn das Aufrufen von „stornieren()“ löst in „Stattgefunden“ keinen Zustandsübergang aus. Egal ob ich mich in „Storniert“ oder „Stattgefunden“ befinde, sobald ich „löschen()“ aufrufe, gehe ich hier in den Endzustand über.

### Bsp.: Lebenszyklus einer Eingabemaske im CALENDARIUM



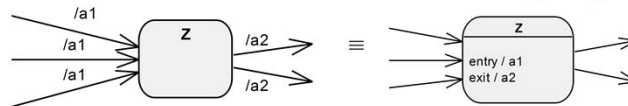
In diesem Beispiel sehen Sie den Lebenszyklus einer Eingabemaske im CALENDARIUM. Das Beispiel können Sie selbst durcharbeiten. Es beschreibt, wie sich die Eingabemaske verhält, wenn Sie „TAB“ oder „ENTER“ drücken.

## Zustandsübergang: Innere Transitionen

- Werden wie »äußere« Transitionen von Ereignissen ausgelöst, **verlassen** aber den **aktuellen Zustand nicht**
- Äquivalent zu Selbsttransition, sofern keine entry / exit-Aktivitäten vorhanden



- Gleiche Aktivitäten können in den Zustand hineingezogen werden:



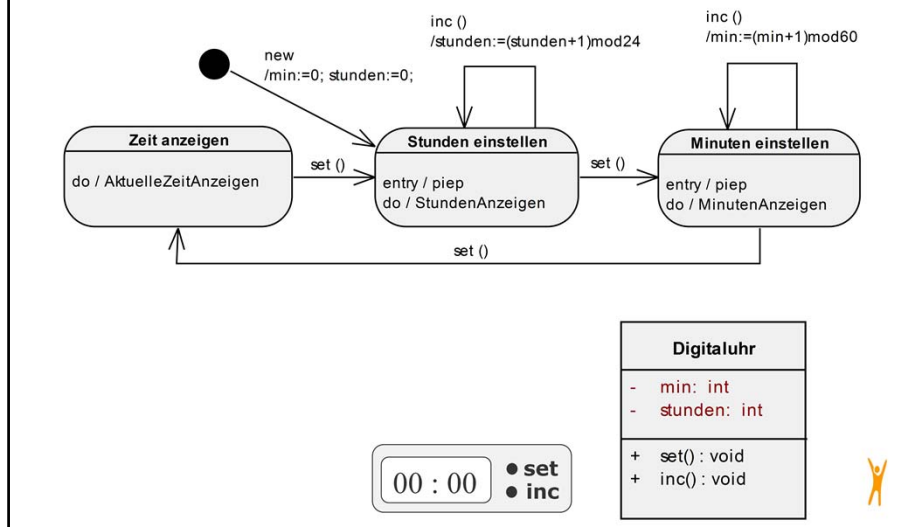
**Innere Transitionen** habe ich auf vorhergehenden Folien schon angesprochen. Sie werden wie äußere Transitionen von Ereignissen ausgelöst, sie verlassen aber den aktuellen Zustand nicht.

Links ist eine normale, äußere Transition dargestellt. Wenn im Zustand „Z“ das Event „e“ eintritt, wird „Z“ verlassen, „a“ wird ausgeführt und dann betritt man wieder „Z“.

Dasselbe könnte man auch wie rechts dargestellt modellieren. Es gibt also den Zustand „Z“. Wenn innerhalb von „Z“ das Event „e“ eintritt, wird die Aktivität „a“ ausgeführt.

Diese beiden Darstellungsformen sind äquivalent, sofern keine entry- oder exit-Aktivität im Zustand „Z“ vorhanden ist.

## Einführung – Beispiel Digitaluhr (2/2)



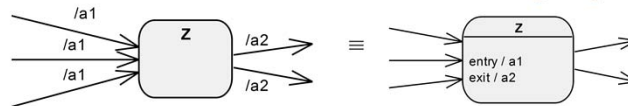
Gehen wir noch einmal zurück zur Folie mit dem Beispiel unserer Digitaluhr. Wenn wir uns den Zustand „Stunden einstellen“ anschauen, dann hatten wir hier das Drücken des Knopfs „increment“ als äußere Transition modelliert. Das hätten wir natürlich auch modellieren können, indem wir innerhalb von „Stunden einstellen“ das Event „increment“ mit der Aktivität zum Erhöhen der Stundenzahl angeben. Worin liegt der Unterschied ob ich das mit einer äußeren oder mit einer inneren Transition löse? Die Antwort ist, das Piepsen. Denn so wie ich das jetzt mit einer äußeren Transition modelliert habe, wird jedes Mal wenn ich „increment“ drücke der Zustand verlassen, die Aktivität ausgeführt und wir kehren wieder in den Zustand zurück und führen natürlich die entry-Aktivitäten „piep“ aus. D.h. nach jedem „increment“ wird gepiepst. Wenn ich jedoch das Ganze als innere Transition angebe, dann verlasse ich den Zustand beim Auftreten des Events „increment“ nicht, damit komme ich auch nicht in den Zustand wieder zurück und daher wird das entry auch nicht wieder ausgeführt. D.h. bei einer inneren Transition von „increment“ würde meine Uhr nur beim „set“ piepsen aber nicht beim „increment“. Wenn ich „increment“ hingegeben als äußere Transition modelliere, so piepst die Uhr auch beim „increment“.

## Zustandsübergang: Innere Transitionen

- Werden wie »äußere« Transitionen von Ereignissen ausgelöst, **verlassen** aber den **aktuellen Zustand nicht**
- Äquivalent zu Selbsttransition, sofern keine entry / exit-Aktivitäten vorhanden

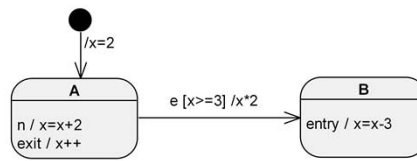


- Gleiche Aktivitäten können in den Zustand hineingezogen werden:



Wir kehren zurück zu unserer Folie zu inneren Transitionen. Gleiche Aktivitäten können auch in den Zustand hineingezogen werden. Anstatt dass ich immer auf allen eingehenden Transitionen „a1“ als Aktivität angebe, kann man „a1“ auch als entry-Aktivität des Zielzustandes angeben. Und wenn bei allen ausgehenden Transitionen die Aktivität „a2“ ausgeführt wird, kann ich „a2“ auch als exit-Aktivität des Quellzustands definieren.

## Ausführungsreihenfolge von Aktivitäten – Bsp.



Event	Zustand	Variable
„Start“	A	x=2
e	A	x=2
n	A	x=4
e	B	x=7



© BIG / TU Wien



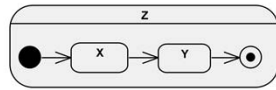
19

Hier noch ein Beispiel um die **Ausführungsreihenfolge** der Aktivitäten verständlich zu machen.

Wir beginnen mit unserem Zustandsautomaten beim Startzustand,  $x$  wird auf 2 gesetzt und wir kommen in den Zustand „A“. Nun tritt das Event „e“ auf. Würde der Zustandsübergang stattfinden, so würde die exit-Aktivität von „A“ ausgeführt werden und damit  $x$  um 1 erhöht werden. Dann wäre  $x \geq 3$  und damit wäre die Bedingung des Zustandsübergangs erfüllt. Diese Interpretation ist jedoch nicht korrekt. Die Reihenfolge ist nämlich folgende: Ich befinde mich in einem Zustand, dann tritt ein Event auf. Ungeachtet von der exit-Aktivität im Zustand und der entry-Aktivität im Nachfolgezustand, wird beim Eintritt von „e“ die Überwachungsbedingung überprüft. D.h. wir befinden uns in „A“,  $x$  ist auf 2 gesetzt, jetzt tritt das Event „e“ auf, wir bleiben weiterhin in „A“ weil die Bedingung nicht erfüllt ist, weil  $x$  noch immer 2 ist. Es tritt nun das Event „n“ ein. Dadurch wird  $x$  um 2 erhöht. Ich bleibe im Zustand „A“ und  $x$  wird jetzt auf 4 gesetzt. Als nächstes tritt das Event „e“ ein. Jetzt ist  $x$  größer als 3, damit ist die Bedingung für den Zustandsübergang erfüllt. Und jetzt ist natürlich die Abarbeitung der folgenden Events entscheidend. Hier ist die Reihenfolge so, dass zuerst das exit ausgeführt wird, d.h.  $x$  wird von 4 auf 5 erhöht. Dann wird die Aktivität des Zustandsübergangs ausgeführt, d.h.  $x = 5 * 2 = 10$  und in der entry-Aktivität von „B“ wird nun 3 abgezogen. D.h. wir befinden uns jetzt im Zustand „B“ und  $x$  hat den Wert 7. D.h. zunächst tritt ein Ereignis ein, dann wird die Bedingung geprüft und wenn die Bedingung wahr ist wird zuerst das exit abgearbeitet, dann die Übergangs-Aktivität und dann das entry.

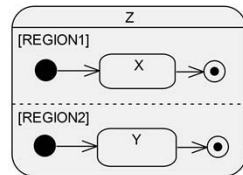
## Komplexe Zustände (1/2)

- = Zustände, die aus mehreren Subzuständen zusammengesetzt sind  
⇒ geschachteltes Zustandsdiagramm
- Die Subzustände sind disjunkt, d.h. genau ein Subzustand ist aktiv, wenn der komplexe Zustand aktiv ist



*Zu einem Zeitpunkt kann nur X ODER Y aktiv sein!*

- Teilung des Superzustandes in mehrere Regionen
  - → die Subzustände sind nebenläufig, gleichzeitig aktiv
  - Z = „orthogonaler Zustand“



*Zu einem Zeitpunkt sind X UND Y aktiv!*

20 

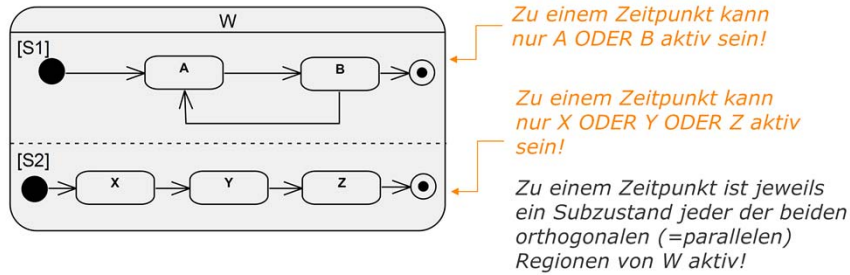
Wir kommen nun zu den **komplexen Zuständen**. Bis jetzt haben wir die Zustandsmaschine, ein Zustandsdiagramme kennengelernt, das nicht geschachtelt war. Jedoch hinter jedem Zustand kann wieder eine Zustandsmaschine liegen.

Hier ist der Zustand „Z“ dargestellt, der Teil von einem anderen Zustandsdiagramm sein kann und innerhalb von „Z“ ist wieder eine Zustandsmaschine spezifiziert. Diese besteht aus den Subzuständen „X“ und „Y“ und dem Endzustand. Hier haben wir eine sogenannte **ODER-Verfeinerung**. Unser System kann sich nämlich während es sich in „Z“ befindet entweder in „X“ oder in „Y“ befinden.

Man kann im Zustand „Z“ aber auch mehrere Regionen definieren und in jeder der Regionen befindet sich wieder eine eigene Zustandsmaschine. Alle diese Zustandsmaschinen werden parallel zueinander abgearbeitet werden. Das entspricht einer **UND-Verfeinerung**. In unserem zweiten Beispiel auf der Folie haben wir „Z“ mit zwei Subzustandsmaschinen. Die erste hat einen Subzustand „X“ und einen Endzustand und die zweite hat einen Zustand „Y“ und ebenfalls einen Endzustand. Dementsprechend kann ich mich sowohl in „X“, als auch in „Y“ gleichzeitig befinden. Die Subzustandsmaschinen werden parallel zueinander ausgeführt.

## Komplexe Zustände (2/2)

### Beispiel



- Mögliche Kombinationen von gleichzeitig aktiven Zuständen:  
 A & X oder A & Y oder A & Z oder A & Endzustand von [S2]  
 B & X oder B & Y oder B & Z oder B & Endzustand von [S2]  
 Endzustand von [S1] & X oder Endzustand von [S1] & Y oder Endzustand von [S1] & Z oder Endzustand von [S1] & Endzustand von [S2]



© BIG / TU Wien

21

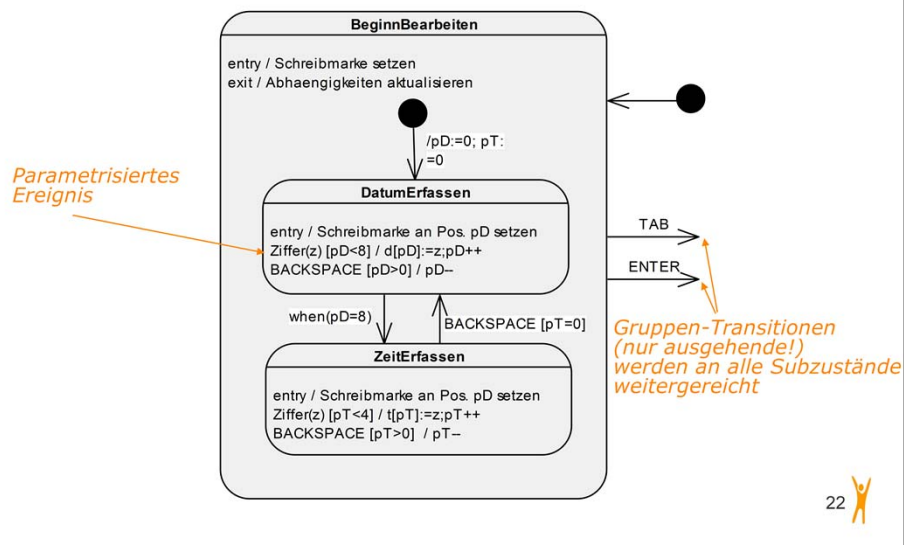


Hier in diesem Beispiel haben wir eine Kombination von UND- und ODER-Verfeinerung. Hier ist der Zustand „W“ dargestellt. Dieser ist unterteilt in zwei Regionen „S1“ und „S2“. Bei „S1“ befindet man sich entweder in „A“ oder in „B“ oder im Endzustand von „S1“. Bei „S2“ befindet man sich entweder in „X“ oder in „Y“ oder in „Z“ oder im Endzustand von „S2“. D.h. die Zustände in einer Region sind miteinander OR-verknüpft. Zwischen den Regionen sind sie mit UND verknüpft. D.h. man kann sich befinden in „A“ und „X“ oder in „A“ und „Y“ oder in „A“ und „Z“ oder in „A“ und Endzustand von „S2“ oder in „B“ und „X“ oder in „B“ und „Y“ oder in „B“ und „Z“ oder in „B“ und Endzustand von „S2“ oder in Endzustand von „S1“ und „X“ oder in Endzustand von „S1“ und „Y“ oder in Endzustand von „S1“ und „Z“ oder in Endzustand von „S1“ und Endzustand von „S2“. Befindet man sich in den beiden Endzuständen, so wird üblicherweise schon der Übergang zum Nachfolgerzustand von „W“ ausgelöst.



## Bsp.: Komplexer Zustand »BeginnBearbeiten«

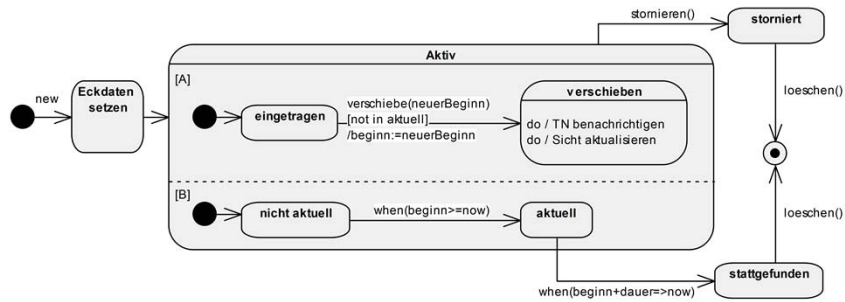
(CALENDARIUM-Bsp)



In diesem Beispiel sehen wir einen wesentlichen Punkt der komplexen Zustände. Transitionen des übergeordneten Zustands, wie hier durch das „TAB“ oder das „ENTER“ ausgelöst, werden als **Gruppen-Transitionen** bezeichnet. Hier ist es egal in welchem Subzustand von „BeginnBearbeiten“ man sich befindet, also egal ob man sich in „DatumErfassen“ oder in „ZeitErfassen“ befindet, wird „BeginnBearbeiten“ verlassen, wenn „TAB“ oder „ENTER“ gedrückt wird.

## Bsp.: Komplexer Zustand »Aktiv«

- ...vom Lebenslauf eines Termins

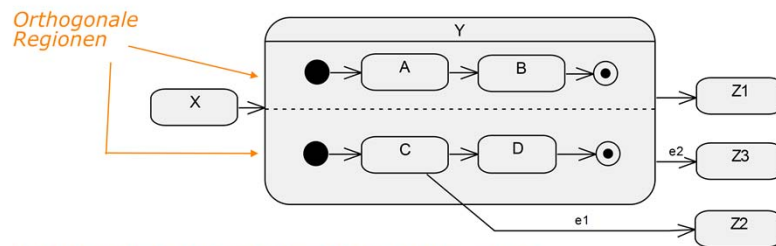


Im folgenden Beispiel sehen wir den Lebenslauf eines Termins. Das ist dasselbe Zustandsdiagramm, das wir vorher schon hatten, jedoch wurde der Zustand „Aktiv“ verfeinert. Innerhalb von „Aktiv“ befinde ich mich in „eingetragen“ oder in „verschieben“ und gleichzeitig in „nicht aktuell“ oder „aktuell“.

Sobald der Umstand „begin  $\geq$  now“ erfüllt ist, wird ein Termin „aktuell“. Aus dem Zustand „aktuell“ kann ich in den Zustand „stattgefunden“ wechseln, wenn  $\text{beginn} + \text{dauer} \geq \text{now}$  ist. D.h. hier erfolgt der Zustandsübergang von einem Ereignis der Verfeinerung. Das bedeutet, egal ob ich in Region „A“ in „eingetragen“ oder „verschieben“ bin, sobald in „aktuell“ diese Bedingung eintritt, verlasse ich den Gesamtzustand von „Aktiv“ und gehe in den Zustand „stattgefunden“ über. Damit wird auch die obere Zustandsmaschine terminiert.

Dann gibt es noch den Zustandsübergang „stornieren()“, der auftreten kann, egal in welchen Subzuständen von „Aktiv“ man sich befindet, denn der Übergang geht direkt vom übergeordneten Zustand „Aktiv“ weg. Wenn das Event „stornieren()“ eintritt, gehe ich in den Zustand „storniert“ über.

## Komplexer Zustand – Verlassen von komplexen Zuständen



- Der komplexe Zustand Y wird verlassen, wenn
  - B und D verlassen worden sind (Folgezustand Z1) [= die Subzustandsfolgen beendet sind] *oder*
  - im Zustand C Ereignis e1 eintritt (Folgezustand Z2) *oder*
  - in irgendeinem Subzustand Ereignis e2 eintritt (Folgezustand Z3)

Hier haben wir noch ein Beispiel zu den komplexen Zuständen.

Ich gehe von „X“ nach „Y“. „Y“ besitzt zwei parallele Regionen. Eine Region beinhaltet die Subzustände „A“ und „B“ und die andere „C“ und „D“.

Wann komme ich in den Zustand „Z1“? – Hier ist kein Event angegeben, d.h. dieser Zustandsübergang erfolgt dann, wenn alle Aktivitäten in den Vorgängerzuständen, also in „Y“, abgeschlossen sind. Das bedeutet bei parallelen Zustandsfolgen, dass alle Subzustandsmaschinen im Endzustand angekommen sind. D.h. wenn sowohl die obere, als auch die untere Zustandsmaschine am Endzustand angekommen sind, wird der nicht beschriftete Zustandsübergang ausgelöst und ich komme nach „Z1“.

Ein anderer Fall ist „Z3“: Egal wo ich mich im Zustand „Y“ befinde, z.B. in „A“ und „D“, ich komme auf jeden Fall zu „Z3“ wenn das Event „e2“ auftritt.

Umgekehrt wenn ich mich in „Y“ in „A“ und „D“ befinde und es tritt „e1“ auf, passiert gar nichts. Nämlich nur wenn ich mich in der unteren Zustandsmaschine in „C“ befinde und es tritt „e1“ auf, dann wird „Y“ komplett verlassen und ich gehe in den Zustand „Z2“.

## Historischer Zustand

- Historische Zustände können sich jenen internen **Zustand** in einem komplexen Zustand **merken**, von dem die **letzte Transition** (vor einer Unterbrechung) **ausgegangen** ist
- Zu einem späteren Zeitpunkt kann zu diesem Zustand über Transitionen **aus übergeordneten Zuständen zurückgekehrt** werden
  - alle Entry-Aktivitäten werden wiederum ausgeführt
- **Flacher History-Zustand** merkt sich **eine Ebene** 
- Über einen **tiefen History-Zustand** »H\*« werden alle Zustände über die **gesamte Schachteltiefe** hinweg gesichert 



© BIG / TU Wien



25

Jetzt kommen wir zu einem wirklich komplexen Konzept, das viele Fehler bei den Prüfungen verursacht, nämlich dem **History-Zustand**.

Der History-Zustand ist sozusagen ein Gedächtnis, das sich merkt in welchem Subzustand ich mich beim Verlassen eines übergeordneten Zustands befunden habe, wenn ich in diesen übergeordneten Zustand wieder zurückkehre.

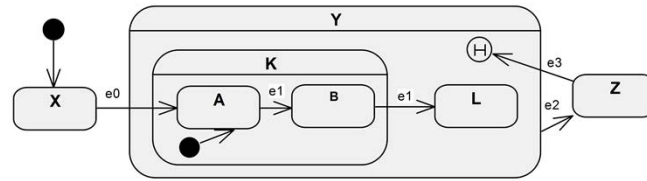
Ein Beispiel aus dem Bereich des Fernsehens: Wenn wir die Möglichkeit des Anzeigens von Teletext betrachten, so gibt es bei Fernsehern zwei Zustände, nämlich „Fernsehbild anzeigen“ und „Teletext anzeigen“. Der Zustand „Teletext anzeigen“ könnte geschachtelt sein in Subzustände für das Anzeigen der verschiedenen Teletextseiten. Wenn ich vom Teletext zum Fernsehbild wechsele und dann wieder in den Teletext wechsele, merken sich manche Fernseher wo ich das letzte Mal im Teletext war und manche beginnen wieder von vorne. D.h. manche Fernseher haben die Funktion eines History-Zustands und wechseln zur Letzt angezeigten Seite als ich mich noch im Teletext befunden habe und andere beginnen wieder von vorne auf der Seite 100.

Wir unterscheiden zwischen zwei Arten von History Zuständen.

Dabei gibt es den **flachen History-Zustand**. Er merkt sich den Subzustand nur über eine Ebene hinweg. Sie können Zustände aber natürlich auch mehrfach schachteln. D.h. z.B. können Sie einen Zustand „A“ haben, der eine Submaschine beinhaltet, die aus „B“ und „C“ besteht und der Zustand „B“ besitzt wieder die Subzustände „M“ und „N“. Beim flachen History-Zustand würde ich mir wenn ich „A“ verlasse nur merken, ob ich in „B“ oder „C“ war und in „B“ und „C“ beginne ich jeweils wieder mit dem Startzustand.

Ein **tiefer History-Zustand** H\* würde sich auch über alle Ebenen hinweg die Subzustände merken. Ich könnte statt dem \*, der für unendlich steht, auch eine Zahl hinschreiben um explizit die Ebenen zu definieren. Bei der Angabe von „H2“ würde sich dann der History-Zustand die Subzustände über 2 Ebenen hinweg merken.

### Bsp.: H vs. H\* (1/2)



Event	Zustand
„Start“	X
e0	Y/K/A
e2	Z
e3	(H→) Y/K/A
e1	Y/K/B
e1	Y/L
e2	Z
e3	(H→) Y/L



© BIG / TU Wien

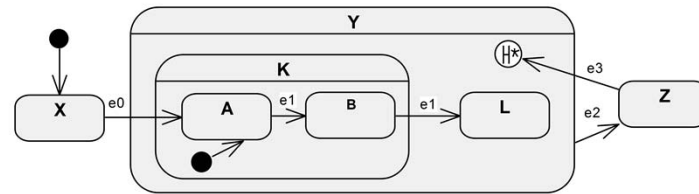
26



Den History-Zustände möchte ich an diesem Beispiel veranschaulichen.

Ich befinde mich zunächst in „X“. Es tritt nun das Event „e0“ auf. Wenn „e0“ auftritt, wechseln wir nach „Y“, innerhalb von „Y“ befinden wir uns dann in „K“ und innerhalb von „K“ in „A“. Jetzt tritt „e2“ auf, damit wird „Y“ verlassen und wir gehen durch das Auftreten von „e2“ in den Zustand „Z“. Es tritt „e3“ auf. Durch „e3“ gehe ich auf jeden Fall in den Zustand „Y“ und durch den History-Zustand ohne Stern (\*) und ohne Zahl merke ich mir nur die letzte Ebene. D.h. beim letzten Mal als ich in „Y“ war, war ich in „K“. D.h. hier gehe ich in „K“ hinein und innerhalb von „K“ starte ich am Startknoten und damit mit dem Zustand „A“. Jetzt tritt „e1“ ein, damit kommen wir in „Y“-„K“-„B“. Es tritt nun wieder „e1“ ein, damit befinden wir uns in „Y“-„L“. Es tritt „e2“ ein, damit befinden wir uns in „Z“. Es tritt „e3“ ein und ich gehe nach „Y“ und das letzte Mal, als ich in „Y“ war, habe ich mich in „L“ befunden und dementsprechend bin ich jetzt in „Y“-„L“.

## Bsp.: H vs. H\* (2/2)



Event	Zustand
„Start“	X
e0	Y/K/A
e1	Y/K/B
e2	Z
e3	(H*→) Y/K/B



© BIG / TU Wien



27

Jetzt verändere ich das Beispiel und zwar ändere ich den flachen History-Zustand in einen tiefen History-Zustand H\*. Wir starten wieder bei „X“. Mit „e0“ komme ich in „Y“-„K“-„A“. Jetzt tritt „e1“ ein. Ich komme in „Y“-„K“-„B“. Jetzt kommt der Aufruf „e2“. Damit komme ich nach „Z“. Und ich mache jetzt den Aufruf „e3“ und komme zum History-Zustand. Hier sehen wir den Unterschied zum flachen History-Zustand. Wenn es ein flacher History-Zustand wäre, würde ich nun nach „Y“-„K“ kommen und dort beim Startzustand und damit bei „A“ starten. Weil ich allerdings einen tiefen History-Zustand habe, merke ich mir über alle Ebenen hinweg, in welchem Subzustand ich zuletzt war und damit komme ich nach „Y“-„K“-„B“.

Das war also das Konzept des flachen bzw. des tiefen History-Zustands.

### Bsp.: Lebenslauf eines to-do-Eintrags

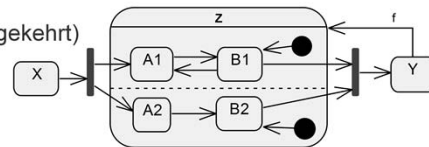
- Lebenslauf eines to-do-Eintrags kann auf folgende Weise vereinfacht dargestellt werden:
  - **Vereinfachung 1:** Markierung der Transition vom Superzustand zum äußeren Endzustand mit dem Ereignis »Löschung« - alle inneren Löschung-Transitionen und alle inneren Endzustände entfallen
  - **Vereinfachung 2:** Zusätzliches **Ausblenden** der Verfeinerung d.h. Verfeinerung wird an anderer Stelle dargestellt –  
Notation:



Wir kommen zu einem weiteren wesentlichen Punkt. Wir haben hier einen Zustand definiert, der rechts unten ein Symbol hat, das jedoch Enterprise Architect spezifisch ist und damit Tool-abhängig ist. Aber was durch dieses Symbol ausgedrückt wird ist generell gültig und nicht Enterprise Architect spezifisch. Es wird ausgedrückt, dass die Verfeinerung von diesem Zustand außerhalb und nicht innerhalb des Zustands erfolgt. Die **Verfeinerung** wird **ausgeblendet**. Wann ist das sinnvoll? – Wenn ich eine Verfeinerung direkt einzeichne, dann hängt sie in diesem Zustand drinnen. Wenn ich aber dieselbe Verfeinerung bei mehreren Zuständen verwenden möchte, dann brauche ich sie nur einmal modellieren und kann sie wiederverwenden. D.h. ich kann dieselbe Subzustandsmaschine zweimal hinterlegen. Den sinnvollen Einsatz dieses Konzepts demonstrieren wir an einem Beispiel in der Vorlesungs-Stunde.

## Komplexe Transition für Orthogonale Zustände

- Wird ein orthogonaler Zustand aktiviert, so werden **alle** seine **nebenläufigen Regionen aktiviert**
- Möchte man den Kontrollfluss jedoch **anders aufspalten** und nicht in allen Regionen die Startzustände aktivieren
  - Verwendung einer komplexen Transition in Form Parallelisierungsknoten bzw. Synchronisierungsknoten
  - Dieser kann eine Transitionsspezifikation tragen, die Pfeile der Zustandsübergänge selbst sind unmarkiert
  - Forderung bei Parallelisierungsknoten: Nachzustände müssen unterschiedlichen Regionen angehören und ihr Vorzustand muss außerhalb liegen  
(Bei Synchronisierungsknoten umgekehrt)



- Eine komplexe Transition ändert damit den »Grad an Nebenläufigkeit«




29

Zu guter Letzt behandeln wir noch die **parallelen Abläufe**.

Im dargestellten Beispiel würde ich im Normalfall von „X“ nach „Z“ gehen und dann habe ich hier zwei parallele Abläufe. Innerhalb von „Z“ sind die Anfangszustände „B1“ und „B2“. D.h. im Normalfall würde ich von „X“ nach „Z“ und damit nach „B1“ und „B2“ gehen. Weil ich aber parallele Abläufe habe ist es hingegen so, dass ich explizit festlege, dass „A1“ und „A2“ die Anfangszustände sind. Beim Verlassen werden diese nebenläufigen Kontrollflüsse wieder zusammengeführt und es wird in den Zustand „Y“ übergegangen. D.h. ich lege bei einem Übergang explizit fest, dass ich meine beiden Submaschinen nicht mit den normalen Anfangszuständen beginnen möchte. In der Praxis ist dieses Konzept nicht so relevant.



## Basiselemente (1/3)

Name	Syntax	Beschreibung
Zustand		Bei Erreichen des Zustands Z wird die Aktivität a1 ausgeführt, während Z der aktuelle Zustand ist, wird a2 ausgeführt und beim Verlassen von Z wird a3 ausgeführt.
Transition		Zustandsübergang
Startzustand		Beginn des Zustandsdiagramms





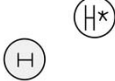
© BIG / TU Wien

30






Und damit sind wir am Ende unsers Kapitels zu den Zustandsdiagrammen. Auf den folgenden Folien finden Sie noch einmal die Notationselemente des Zustandsdiagramms zusammengefasst.

## Basiselemente (2/3)

Name	Syntax	Beschreibung
Endzustand		Ende
Terminierungs-knoten		Das modellierte Objekt hört auf zu existieren.
Flacher/tiefer History-Zustand		"Rücksprungadresse"



### Basiselemente (3/3)

Name	Syntax	Beschreibung
Entscheidungs-knoten		Knoten, von dem mehrere alternative Transitionen ausgehen können.
Parallelisierungs-knoten		Aufspaltung des Kontrollflusses in mehrere parallele Zustände
Synchronisierungs-knoten		Zusammenführung des Kontrollflusses von mehreren parallelen Zuständen



© BIG / TU Wien

32



## Zusammenfassung

---

- Sie haben diese Lektion verstanden, wenn Sie wissen..
- Was mit dem Zustandsdiagramm modelliert wird
- Was Ereignisse und Aktivitäten sind und wie sie eingesetzt werden
- Wozu Bedingungen benötigt werden und was der Unterschied zu Ereignissen ist
- Welche Aktivitäten es innerhalb eines Zustands gibt
- Wozu und wie ein Historischer Zustand eingesetzt wird
- Was komplexe und orthogonale Zustände sind

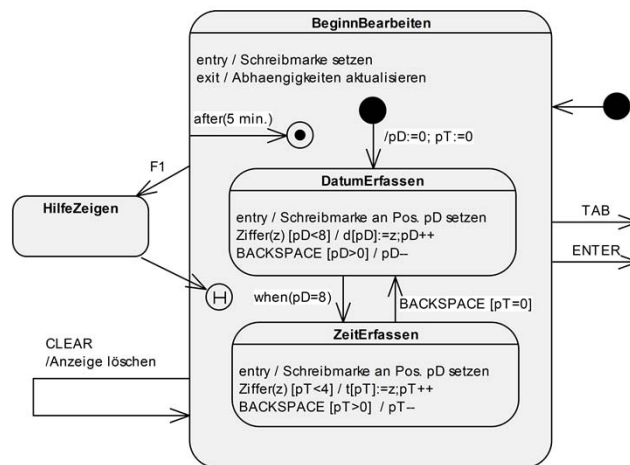


Anhang

## Bsp.: Lebenslauf einer Termin-Eingabemaske

- Variante des komplexen Zustands »BeginnBearbeiten«

(CALENDARIUM-Bsp)



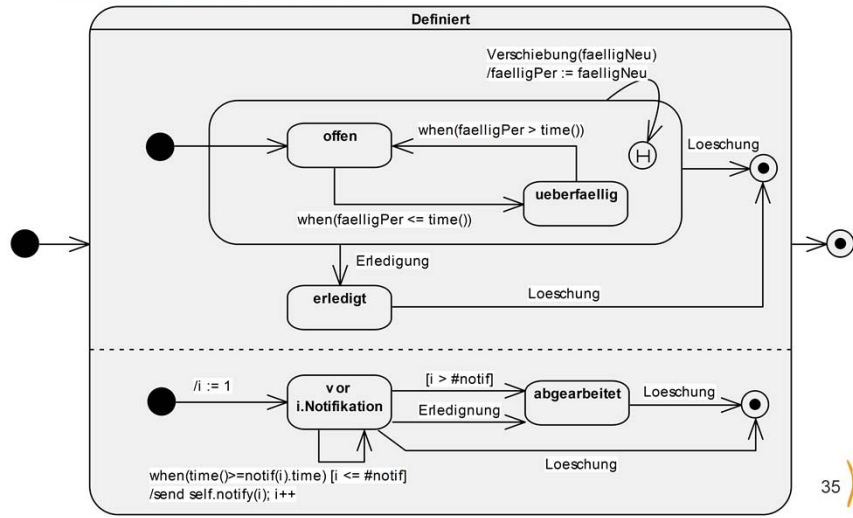
34



## Bsp.: Lebenslauf eines to-do-Eintrags

(CALENDARIUM-Bsp)

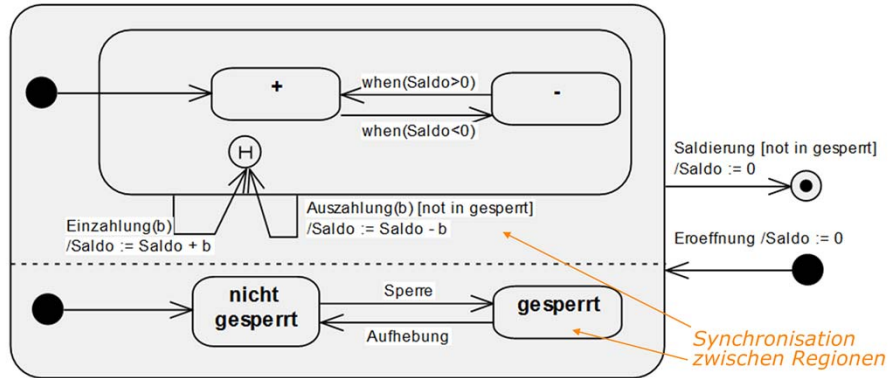
- Komplexer Zustand »Definiert«



35 

## Bsp.: Konto

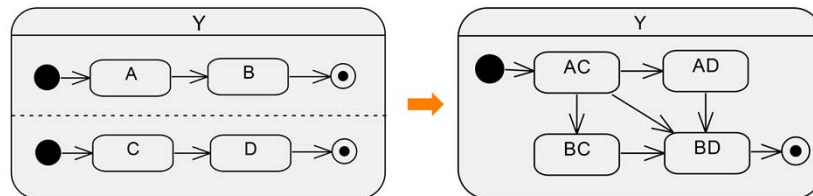
- Modellierung von unabhängigen Zustandsmengen eines Objektes ("mehrdimensionale Modellierung") durch orthogonale Regionen





## Orthogonale Zustände vs. sequentielle Form

- Orthogonale Zustände können durch die Erzeugung von Produktautomaten auf nicht orthogonale Zustände abgebildet werden
- Für jede mögliche Kombination von orthogonalen Zuständen wird ein eigener Zustand definiert – Transitionen werden entsprechend dupliziert
- Beispiel:



## Bsp.: orthogonale Zustände vs. sequent. Form

- Transformation eines orthogonalen Zustands in eine sequentielle Form ist **umständlich** und **semantisch nicht äquivalent**
  - Erzeugung von **Produktautomaten**
  - Beispiel: Konto
  - »G« und »NG« stehen für »gesperrt« bzw. »nicht gesperrt«

