Zustandsübergangsdiagramme

Udo Kelter

03.10.2003

Zusammenfassung dieses Lehrmoduls

Zustandsübergangsdiagramme (ZÜD) modellieren die Zustände eines Systems und die Übergänge zwischen diesen Zuständen aufgrund äußerer Ereignisse. Zustandsübergangsdiagramme werden in unterschiedlichen Phasen und Kontexten eingesetzt. Dieses Lehrmodul stellt zunächst die grundlegenden Varianten von Zustandsübergangsdiagrammen vor, die Mealy- bzw. Moore-Automaten entsprechen. Aufbauend hierauf werden Hierarchien von Zuständen bzw. ZÜD definiert. Schließlich skizzieren wir mehrere Erweiterungen, die in der UML definiert werden.

Vorausgesetzte Lehrmodule:

empfohlen: – Systemanalyse und Systemmodellierung

Stoffumfang in Vorlesungsdoppelstunden: 0.9

Inhaltsverzeichnis

1 Motivation und Einordnung

Zustandsübergangsdiagramme (ZÜD) modellieren die Zustände eines Systems und die Übergänge zwischen diesen Zuständen aufgrund äußerer Ereignisse. Der Begriff System ist hier sehr weit gefaßt; Beispiele für Systeme und ihre Zustände sind:

- Ein Informationssystem kann zwei Zustände *Tagbetrieb* und *Nachtbetrieb* haben. Im Nachtbetrieb sind nur einfache Auskunftsfunktionen verfügbar. Um 8 bzw. 18 Uhr wird jeweils zwischen den Zuständen umgeschaltet.
- Ein Drucker kann beispielsweise folgende Zustände haben:
  - initialisiert sich gerade (läuft warm)
  - wartet auf Druckauftrag
  - druckt gerade
  - kein Papier mehr vorhanden
  - gestört
- Ein Objekt in einem Programm, das ein graphisches Auswahlmenü ("radio buttons") realisiert, befindet sich im Zustand *nichts gewählt* oder im Zustand *Menüeintrag x gewählt*, wobei x einer der Menüeinträge ist. Beim Anklicken einer der Schaltflächen oder Eingabe eines entsprechenden Buchstabens über die Tastatur geht das Objekt in den gewählten Zustand über, beim Drücken der Escape-Taste in den Zustand *nichts gewählt*.

Diese Beispiele zeigen, daß die Größe der modellierten Systeme um Größenordnungen schwanken kann. Das letzte Beispiel ist ferner ein Objekt, das für die Systemanalyse nicht mehr relevant sein wird, d.h. die Untersuchung von Systemzuständen kann auch in späteren Phasen, also beim Entwurf oder beim Programmieren im Kleinen, sinnvoll sein.

Die grundlegenden Formen von ZÜD kann sind graphische Darstellungen von endlichen Automaten (s. Abschnitt 2.3). ZÜD bzw. endliche Automaten werden vielfach zur Modellierung und/oder exakten Spezifikation von Systemen eingesetzt.

ZÜD eignen sich nicht zur Modellierung verteilter Systeme, die aus Komponenten bestehen, deren Zustand sich unabhängig weiterentwickelt. Hierfür sind Petri-Netze, die in einem eigenen Lehrmodul vorgestellt werden, besser geeignet.

Neben einfachen Varianten von ZÜD, die nur die grundlegenden Konzepte beinhalten, gibt es diverse Erweiterungen, z.B. die *statechart diagrams* der UML [UML99]. Ein *statechart diagram* ist die graphische Darstellung einer *state machine*. Die *state machines* der UML vereinigen Konzepte, die aus ZÜD, Petri-Netzen und Programmablaufplänen stammen.

Zustandsmodelle werden fast immer zusammen mit Modellen anderen Typs benutzt (s. Abschnitt 2 in [SASM]). Sie treten daher als eine unter mehreren Modellarten in vielen komplexen Analysemethoden auf, insb. in der UML und in der Modernen Strukturierten Analyse (s. [MSA]). Wie üblich werden dabei Konsistenzkriterien zwischen den ZÜD und anderen Modelltypen, namentlich Funktionsmodellen, definiert.

2 Einfache Zustandsübergangsdiagramme

2.1 Zustände

ZÜD unterstellen, daß sich das *gesamte* betrachtete System immer in einem von mehreren möglichen Zuständen befindet. Sei **Z** die Menge möglicher Zustände. Z muß endlich sein. Beispiele für Zustandsmengen wurden schon in Abschnitt 1 angegeben.

Aus der Menge Z wird genau ein Zustand als **Anfangszustand** ausgewählt. Dieser wird nach der Entstehung des Systems automatisch als erster eingenommen.

Weiter kann es eine beliebige Menge von **Endzuständen** geben, in denen kein Übergang auf einen anderen Zustand mehr möglich ist. Endzustände können so interpretiert werden, daß das System seine Aufgabe erfüllt hat und aufgelöst werden kann.

2.2 Ereignisse und Zustandsübergänge

Wenn ein System immer in seinem Anfangszustand bliebe, wäre das langweilig. Es müssen also Zustandsübergänge möglich sein. Zustandsübergänge werden ausgelöst durch **Ereignisse**. Beispiele für Ereignisse sind das Drücken irgendwelcher Tasten, Ankunft einer Nachricht über eine Schnittstelle usw. Ereignisse sind insofern extern, als sie nicht durch das betrachtete System selbst ausgelöst werden, sondern eben durch eine äußere Instanz.

Die Menge möglicher Ereignisse muß bei der Bildung des ZÜD vollständig bekannt sein; wir bezeichnen diese Menge mit **E**. E muß natürlich auch endlich sein.

Der Effekt eines Ereignisses hängt vom Zustand ab, in dem sich das System befindet. Betrachten wir als Beispiel eine Lampe, die die Zustände *ein*, *aus* und *blinkend* haben kann und die zwei Taster 'EIN/AUS' und 'Blinken' hat; die möglichen Ereignisse sind das Drücken je einer der Tasten. In der folgenden Tabelle ist angegeben, welcher neuer Zustand abhängig vom vorhandenen Zustand und vom eingetretenen Ereignis eingenommen wird:

bisheriger Zustand	Ereignis	neuer Zustand
aus	EIN/AUS	ein
ein	EIN/AUS	aus
ein	Blinken	blinkend
blinkend	EIN/AUS	aus

Im Zustand *aus* oder *blinkend* hat das Drücken der Taste 'Blinken' keinen Effekt, deshalb fehlen entsprechende Zeilen in der Tabelle.

Jede Zeile in der Tabelle stellt eine **Transition** dar. Wenn das angegebene Ereignis eintritt, sagt man, daß die Transition "feuert".

Die vorstehende **Zustandsübergangstabelle** stellt, wenn man es etwas abstrakter betrachtet, eine partielle Funktion

$$z\ddot{u}f : Z \times E \to Z$$

dar. Diese **Zustandsübergangsfunktion** ordnet einem Zustand

$z \in Z$ und einem Ereignis $e \in E$ entweder keinen Wert zu - dann hat e keinen Effekt - oder genau einen Folgezustand.

Übergänge zwischen den Zuständen sind zeitlos bzw. atomar. Von eventuellen in der Realität auftretenden Übergängen wird in den endlichen Automaten bzw. ZÜD abstrahiert.

2.3 Endliche Automaten

Endliche Automaten sind ein grundlegendes Konzept der (theoretischen) Informatik. In der einfachsten Form ist ein **endlicher Automat**[1] definiert durch die Menge der Zustände und die o.g. Zustandsübergangsfunktion. Endliche Automaten sind somit sehr abstrakte, mathematische Modelle konkreter Systeme, die Darstellungsform (graphisch oder tabellarisch) bleibt hier außer Betracht.

Eine klassische und sehr häufige Anwendung endlicher Automaten liegt darin, bei Texten zu überprüfen, ob sie syntaktisch korrekt sind, und ggf. den Text in einen anderen zu übersetzen. Die Syntax der Texte muß hier relativ einfach strukturiert sein, sie muß in Form von regulären Ausdrücken angebbar sein. Die von regulären Ausdrücken definierten Sprachen nennt man auch regulär. Zu jeder regulären Sprache kann ein endlicher Automat gefunden werden, der Worte aus dieser Sprache erkennt, und umgekehrt[2].

Ein konkreter Automat entspricht einer konkreten Syntax bzw. Sprache und behandelt beliebige Texte aus dieser Sprache. Endliche Automaten haben also eine *operationale Semantik* und daher die Qualität eines ausführbaren Programms, sie sind nicht nur ein vereinfachendes oder unvollständiges Modell eines anderen Systems (wie z.B. Klassen- oder Sequenzdiagramme), sie sind selbst das System. Sie können durch einen "Compiler" in ausführbaren Code transfor-

[1] Genauer gesagt ein deterministischer endlicher Automat ohne Ausgabe; es gibt auch nichtdeterministische endliche Automaten, die wir hier aber nicht betrachten. Ausgaben entsprechen den anschließend diskutierten Aktionen. Endliche Automaten werden auch als Zustandsautomaten, *finite automaton* und *sequential machine* bezeichnet.

[2] Diese Thematik wird im Rahmen von Vorlesungen über Compilerbau oder Theoretische Informatik detailliert behandelt.

miert werden. Die komplexen Varianten von ZÜD (z.B. solche mit bedingten Übergängen) stellen keine endlichen Automaten mehr dar und haben demzufolge auch keine eindeutige operationale Semantik.

Festzuhalten bleibt, daß ZÜD nicht nur in der Systemanalyse, sondern auch beim Entwurf und beim Programmieren im Kleinen einsetzbar sind.

2.4 Aktionen

Daß sich ein modelliertes System in einem bestimmten inneren Zustand befindet und diesen ab und zu wechselt, mag ja schön und gut sein; die Frage ist, was hat der Rest der Welt davon? Um etwas außerhalb des modellierten Systems bewirken zu können, wird das Konzept der Aktion benötigt.

Eine **Aktion** ist eine Operation oder ein Dienst, den die Umgebung des modellierten Systems zur Verfügung stellt. Analog zu Ereignissen unterstellen wir eine endliche Menge **A** von Aktionen.

Für den Aufruf von Operationen sind zwei Ansätze denkbar; die Wahl hängt davon ab, wie man den Sachverhalt, daß sich ein System in einem Zustand z befindet, interpretiert:

1. Das System tut nichts und wartet auf das nächste Ereignis. Diese Auffassung liegt den sog. **Mealy-Automaten** zugrunde.
2. Das System ist in einem Zustand mit der Bearbeitung einer zugehörigen Aufgabe beschäftigt. Auf dieser Auffassung basieren die sog. **Moore-Automaten**.

Beide Automatentypen sind äquivalent, wir werden hier, sofern nichts anderes erwähnt ist, von Mealy-Automaten ausgehen.

2.5 Mealy-Automaten

Bei Mealy-Automaten werden Aktionen nur "zwischen den Wartezuständen", also im Rahmen der Verarbeitung eines Ereignisses aufgerufen, sonst nicht. Da wir die Abarbeitung von Ereignissen als atomar ansehen, ist konsequenterweise auch die Ausführung von Aktionen zeitlos.

Welche Aktion aufgerufen wird, hängt analog zum Übergang in den Folgezustand vom bisherigen Zustand *und* dem eingetretenen Ereignis ab. Wir können die gewählte Aktion als weitere Spalte in der Zustandsübergangstabelle notieren, etwa wie folgt:

bisheriger Zustand	Ereignis	neuer Zustand	Aktion
aus	EIN/AUS	ein	Strom einschalten
ein	EIN/AUS	aus	Strom ausschalten
ein	Blinken	blinkend	Unterbrecher einschalten
blinkend	EIN/AUS	aus	Strom ausschalten

Auch hier liegt analog zu *züf* eine partielle Funktion

$$af : Z \times E \to A$$

vor. Diese **Aktionsfunktion** ordnet einem Zustand und einem eingetretenen Ereignis, sofern sie definiert ist, genau eine auszuführende Aktion zu.

2.6 Graphische Darstellung

In den vorstehenden Abschnitten haben wir eine textuelle Darstellung von endlichen Automaten benutzt. Zustandsübergangsdiagramme erlauben es, endliche Automaten graphisch darzustellen. Wir lehnen uns hier an die Notationsformen der UML an. Bild 1 zeigt das vorstehende Beispiel in graphischer Notation.

Die UML benutzt folgende Notationsformen:

- Jeder *Zustand* wird durch einen Kasten mit gerundeten Ecken repräsentiert. Innen steht der Name des Zustands. Der Name kann auch fehlen, wenn er unwichtig ist, dann wird der Zustand **anonym** genannt. Jedes Zustandssymbol ohne Namen repräsentiert einen anderen (anonymen) Zustand.

  Ein einziger Zustand kann auch durch mehrere Zustandssymbole, die alle den gleichen Namen enthalten, repräsentiert werden.

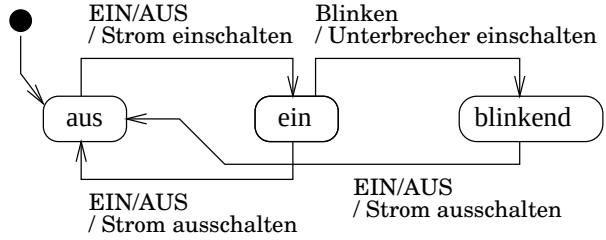

Abbildung 1: Beispiel für ein Zustandsübergangsdiagramm

Dies kann ausgenutzt werden, um die Diagramme übersichtlicher zu machen.

- *Zustandsübergänge* werden durch Pfeile vom bisherigen Zustand zum Folgezustand repräsentiert. Der Pfeil hat eine Beschriftung in der Form "e / a" worin e das Ereignis ist, das den Zustandsübergang auslöst, und a die aufgerufene Aktion.
- Der *Anfangszustand* wird durch einen "spontanen" Zustandsübergang markiert. Hierzu wird als graphisches Hilfssymbol ein dicker schwarzer Punkt benutzt, von dem aus genau ein Pfeil (also ein Zustandsübergang) ohne Beschriftung ausgeht und zu dem kein Pfeil hinführt. Man kann dies so interpretieren, daß der schwarze Punkt einen allerersten Urzustand darstellt, von dem aus sofort und ohne äußeren Anlaß und ohne Aufruf einer Aktion in den eigentlichen Anfangszustand übergegangen wird. Es sei aber betont, daß der schwarze Punkt keinen Zustand im Sinne von logischen Modelldaten repräsentiert[3]. In der UML-Diktion wird der schwarze Punkt als **Pseudozustand** bezeichnet.
- Ein Zustand, von dem keine Zustandsübergänge ausgehen, ist infolgedessen implizit ein Endzustand. Ein Zustand kann auch explizit als Endzustand gekennzeichnet werden, und zwar durch einen unbeschrifteten Zustandsübergang zu einem speziellen graphischen Symbol, einem eingekreisten schwarzen Punkt ("Bullauge"; s. Bild 4); dieses Symbol ist in der UML-Diktion ebenfalls ein Pseudozu-

[3] vgl. Abschnitt 5 in [DMER]

stand.

2.7 Moore-Automaten

Bei einem Moore-Automaten wird einem Zustand ein bestimmter Dienst zugeordnet, der in der Zeit ausgeführt wird, in der das System in diesem Zustand ist. Dies unterstellt, daß die Dienstausführung eine Zeitlang dauert - im Gegensatz zu Mealy-Automaten, wo Dienstausführungen zeitlos waren. Dienste werden daher hier als **Aktivitäten** bezeichnet.

Welche Aktivität ausgeführt wird, hängt hier *nur vom neuen Zustand* ab (und nicht vom alten Zustand oder dem Ereignis, das den Übergang verursachte). Textuell kann man dies durch eine Tabelle darstellen, die jedem Zustand eine Aktivität zuordnet. In der graphischen Darstellung kann man die Aktivität innerhalb oder unterhalb des Symbols für den Zustand notieren (s. Abschnitt 4.5).

Beim Übergang in einen bestimmten Zustand z wird die Ausführung eines Dienstes gestartet. Wenn die Ausführungszeit unbegrenzt ist ('blinken'), wird die Ausführung des Dienstes erst durch ein Ereignis, das einen Zustandswechsel verursacht, unterbrochen und beendet.

Ist die Ausführungszeit begrenzt, müßte konsequenterweise, wenn die Abarbeitung des Dienstes endet, der Zustand verlassen werden; um zu entscheiden, in welchen Folgezustand man übergeht, braucht man aber ein Ereignis. Hier sind zwei Lösungen denkbar:

- Es wird in dem Zustand verblieben und auf das nächste Ereignis gewartet.
- Man definiert eine Vorgabe, welcher Folgezustand beim Verlassen dieses Zustands einzunehmen ist. Dieser Zustandsübergang benötigt kein Ereignis – d.h. in der Diagrammnotation tritt ein Übergangspfeil auf, an dem kein Ereignis steht – oder anders gesehen liegt hier ein **implizites Ereignis** vor.

  Von einem Zustand aus darf maximal eine Transition mit einem impliziten Ereignis ausgehen.

Wenn ein Ereignis e eintritt, das von z aus wieder zu z führt, also

$z\ddot{u}f(z,e) = z$, wird der Zustand zwar nicht gewechselt, aber dennoch die zugehörige Aktivität erneut ausgeführt.

3 Zustandshierarchien

3.1 Motivation

In komplexen Systemen kann die Zahl der Zustände groß werden, wodurch die ZÜD unübersichtlich werden. Ferner kann es sinnvoll sein, mehrere Detaillierungsgrade zu unterscheiden. Betrachten wir hierzu noch einmal das obige Beispiel eines Druckers, dort den Zustand *gestört*. Eine Störung kann mehrere Ursachen haben:

- Es ist keine Tonerpatrone eingelegt.
- Papierstau
- Störung im Netzwerkanschluß bei der Übertragung von Daten

Der Zustand *gestört* kann somit in drei Unterzustände geteilt werden. Weiter kann ein Papierstau an unterschiedlichen Stellen des Papierlaufwegs eingetreten sein, daher muß im Falle eines Papierstaus zusätzlich unterschieden werden, an welcher Stelle der Stau aufgetreten ist. In einem Display erscheint dann ein Hinweis für den Benutzer, welche Inspektionsklappe er öffnen soll. Nach Öffnen einer Klappe erscheinen im Display weitere Hinweise. Der Unterzustand Papierstau kann also noch weiter aufgeteilt werden. Bild 2 zeigt den resultierenden Zustandsbaum.

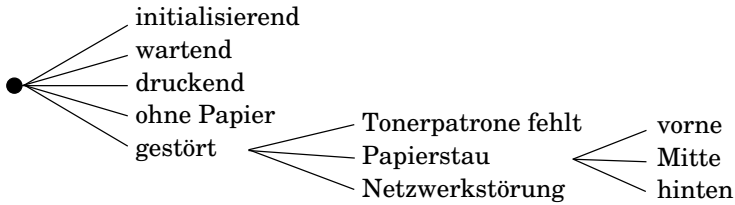

Abbildung 2: Baum der Zustände eines Druckers

Allgemein kann man eine beliebige baumartige Hierarchie von Zuständen bilden. Die Blätter dieses Baums sind die "konkreten" Zustände, die Nicht-Blätter (also Zustände mit Unterzuständen) sind "abstrakt"[4]. Ein abstrakter Zustand entspricht der Menge der konkreten Zustände, die direkt oder indirekt Unterzustand von ihm sind. Ein System befindet sich dann immer in genau einem konkreten Zustand und damit implizit auch in allen zugehörigen abstrakten Zuständen.

3.2 Die Verfeinerung eines Zustands

Analog zur prozeduralen Abstraktion (bzw. zum *information hiding*) durch Funktionen oder Prozeduren sollte es für ein ZÜD *zd* keine Rolle spielen, ob ein darin auftretender Zustand *z* Unterzustände hat oder nicht; dementsprechend sollte eine Änderung in der Struktur der Unterzustände von *z* keine Auswirkungen auf *zd* haben. Dies setzt voraus, daß die Unterzustände von *z* sich nur in Details unterscheiden, von denen auf der Abstraktionsebene von *zd* abstrahiert wird.

Als **Verfeinerung** eines abstrakten Zustands bezeichnen wir die Menge seiner direkten Unterzustände zusammen mit den Übergängen zwischen diesen Zuständen. Eine Verfeinerung bildet einen "flachen" endlichen Automaten, der sozusagen das interne Verhalten innerhalb des übergeordneten abstrakten Zustands darstellt.

Graphisch dargestellt werden kann die Verfeinerung eines Zustands *z* auf zwei Arten:

- Das Zustandssymbol von z wird groß ausgelegt und dadurch zu einer Zeichenfläche, auf der die Verfeinerung von z eingetragen werden kann. Der Name z selbst wird in die obere linke Ecke der Zeichenfläche geschrieben. Bild 3 zeigt als Beispiel eine modifizierte Version

[4] In der UML-Diktion werden abstrakte Zustände als **zusammengesetzt** (*composite*) bezeichnet und die hier beschriebenen Unterzustände als *mutually exclusive disjoint substates*. Die Bezeichnung zusammengesetzt ist hier insofern etwas irreführend, als sich diese Zustände gerade nicht aus lokalen Zuständen von Komponenten des Systems zusammensetzen. Die UML definiert noch andere Arten der Zustandszerlegung, auf die die Bezeichnung zusammengesetzt besser zutrifft, die wir aber in diesem Lehrmodul nicht behandeln.

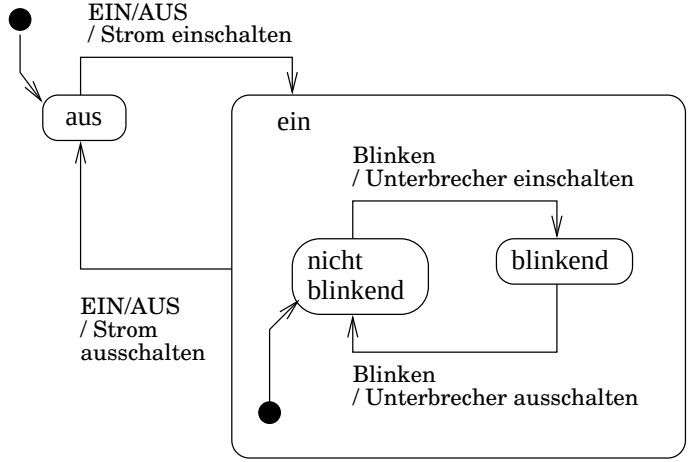

Abbildung 3: Zustandsübergangsdiagramm mit Verfeinerung

unserer Blinklampe. Im Zustand *ein* kann man nun durch Drücken der Taste Blinken zwischen *blinkend* und *nicht blinkend* wechseln.

– Durch ein separates Diagramm.

3.3 Zustandsübergänge zwischen abstrakten Zuständen

Wenn ein Zustandsübergang in einen abstrakten Zustand z erfolgt, muß ein konkreter(er) Zustand aus der Verfeinerung von z angenommen werden. Sei v das ZÜD, das die Verfeinerung von z darstellt. v hat, wie jedes ZÜD, genau einen Anfangszustand. Dieser bestimmt, welcher konkrete(re)[5] Zustand initial eingenommen wird. Bei unserer Blinklampe wird also beim Übergang in den abstrakten Zustand *ein* zunächst der konkrete Zustand *nicht blinkend* eingenommen.

Verlassen werden kann ein abstrakter Zustand durch ein Ereignis auf der Abstraktionsebene dieses Zustands oder einer noch höheren Ebene[6]: unsere Blinklampe wird im Zustand *ein* durch Drücken

[5] Ein Zustand innerhalb von v kann seinerseits eine Verfeinerung haben, also abstrakt sein. Immerhin ist er "konkreter" als z.

[6] Hierzu muß man die von außen eintretenden Ereignisse eindeutig einer Ebe-

der Taste EIN/AUS unabhängig davon, ob sie blinkt oder nicht, in den Zustand *aus* überführt. Bei vielen Systemen kann in beliebigen Zuständen durch Drücken einer reset-Taste ein Übergang in den Anfangszustand erzwungen werden.

Wenn das die Verfeinerung darstellende ZÜD v Endzustände hat und ein solcher Endzustand erreicht wird, liegt im Prinzip das gleiche Problem vor, das wir schon in Abschnitt 2.7 im Kontext vom Moore-Automaten diskutiert haben:

- Wenn man das Erreichen eines Endzustands als Beendigung einer lokalen Verarbeitung innerhalb des abstrakten Zustands ansieht, muß man von dem abstrakten Zustand aus einen Zustandsübergang mit einem impliziten Ereignis vorsehen.

  Bild 4 zeigt ausgehend vom Zustand *gestört* einen Übergang in den Zustand *bereit*, bei dem kein Ereignis explizit angegeben ist: das implizite Ereignis ist sozusagen das Erreichen des Endzustands innerhalb des abstrakten Zustands *gestört*.

  Von einem Zustand darf nur dann eine Transition mit einem impliziten Ereignis ausgehen, wenn der Zustand eine Verfeinerung hat[7].
- Andernfalls passiert "intern" in dem abstrakten Zustand nichts mehr, d.h. man wartet auf ein Ereignis auf der Abstraktionsebene des abstrakten Zustands.

3.4 Hierarchische ZÜD

Da Zustände in mehreren Ebenen verfeinert werden können, entsteht i.a eine Verfeinerungshierarchie von ZÜD, deren Baumstruktur gerade durch die Struktur der abstrakten Zustände gegeben ist. Die Menge der ZÜD mit ihrer Struktur bezeichnen wir als **hierarchisches ZÜD**.

ne zuordnen können. Die Ereignisbezeichnungen auf den verschiedenen Ebenen müssen daher verschieden sind.

[7]Daß die Existenz der Verfeinerung bekannt ist, widerspricht eigentlich dem Prinzip des *information hiding*.

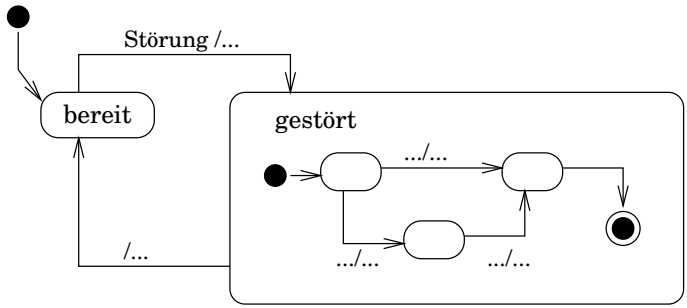

Abbildung 4: Zustandsübergang mit implizitem Ereignis

Selbsttestaufgabe: Zeigen Sie, daß mit den vorstehenden Definitionen ein hierarchisches ZÜD in ein äquivalentes "flaches" ZÜD umgeformt werden kann, dessen Zustandsmenge aus den konkreten Zustände des hierarchischen ZÜD besteht.

4 Konzeptionelle Erweiterungen

Die bisher vorgestellten ZÜD beruhen auf sehr klaren und einfachen mathematischen Grundlagen, nämlich endlichen Automaten. Sie können allerdings nicht alle in der Praxis auftretenden Aufgaben lösen, daher existieren diverse Erweiterungen, die nicht nur neue Notationen verursachen, sondern auch neue Modellierungskonzepte erfordern. Wir stellen hier nur eine Auswahl aus den sehr vielfältigen Erweiterungen vor, die in der UML definiert werden.

4.1 Zeitangaben

Betrachten wir als Beispiel ein Telefon. Nach Abheben des Hörers sind wir in einem Zustand *wählend*, in dem wir eine Telefonnummer eingeben können (und sollen). Wenn nach 30 Sekunden noch keine Ziffer eingegeben worden ist, wird der Wahlversuch abgebrochen und der Zustand *Wahl unterbrochen* angenommen, in dem ein Besetzt-Signal

ertönt; danach kann nur noch der Hörer aufgelegt werden. Bild 5 zeigt einen Ausschnitt aus einem ZÜD für das Telefon. Zwischen den Zuständen *wählend* und *Wahl unterbrochen* befindet sich ein Übergang für das Ereignis `after (30 sec)`. Während bisher Ereignisse nur von äußeren Instanzen verursacht wurden, wird dieses Ereignis sozusagen von der virtuellen Maschine, die das ZÜD interpretiert, selbst generiert.

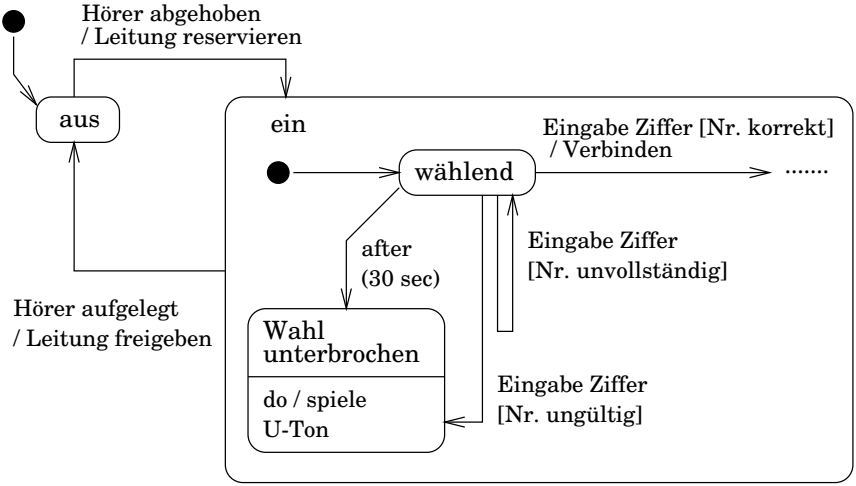

Abbildung 5: Zeitangaben und Bedingungen

4.2 Bedingungen

Statt einer Wartezeit kann auch ein konkreter Zeitpunkt angegeben werden, z.B. in der Form `when (date = 1.1.2000)`[8].

Genereller können in einer `when`-Klausel Bedingungen angegeben werden, die nach dem Eintritt in den Ausgangszustand immer wieder überprüft werden, wenn sich eine der Eingangsgrößen ändert. Sobald die Bedingung wahr wird, erzeugt dies ein Ereignis, durch das die

[8] Eine präzise Syntax ist in der UML-Definition [UML99], 3.77.2 nicht angegeben.

Transition feuert.

4.3 Bedingte Transitionen

Beispiele für bedingte Transitionen (*guarded transition*) finden sich in Bild 5. Dort wird u.a. im Zustand *wählend* nach Drücken einer Ziffer unter der Bedingung, daß nunmehr eine ungültige Nummer eingegeben worden ist, der Zustand *Wahl unterbrochen* eingenommen. Die Bedingung (*guard condition*) steht in eckigen Klammern hinter dem Ereignis.

Im Gegensatz zu den `when`-Bedingungen ist eine *guard condition* kein Ersatz für ein Ereignis; vielmehr liegt schon ein Ereignis vor, bei dem aber *zusätzlich* der Übergang in einen Folgezustand davon abhängig gemacht wird, daß die *guard condition* erfüllt ist.

4.4 Operationen als Ereignisse bzw. Aktionen

In der UML werden ZÜD vor allem dazu verwendet, die Zustände von Objekten anzugeben. Auf alle Objekte einer Klasse trifft das gleiche ZÜD zu.

Objekte können ihren Zustand natürlich als Folge einer Operationsausführung ändern. In diesem Fall ist der Aufruf einer Operation des Objekts (in der UML-Diktion: Ankunft einer Nachricht) das Ereignis, das den Zustandsübergang auslöst. Konsequenterweise sind die Namen der Operationen auch zulässige Ereignisnamen.

Naheliegenderweise sind die Operationen auch als Aktionen zulässig. In der Regel wird man bei einem Aufruf einer Operation diese auch ausführen, d.h. man hätte dann einen Zustandsübergang mit einer Beschriftung der Form "o / o", wobei o der Name der Operation ist. Als Konvention kann man diese Beschriftung dahingehend vereinfachen, daß kein Ereignis angegeben wird.

4.5 Interne Aktionen

Wir hatten bisher Mealy-Automaten unterstellt, bei denen Aktionen nur beim Feuern von Transitionen aufgerufen werden. Oft möchte

man aber auch während der Zeit, in der man sich in einem Zustand befindet, Aktionen aufrufen. Die UML nennt dies **interne Aktionen**. Bild 5 enthält als Beispiel, daß, solange das Telefon im Zustand *Wahl unterbrochen* ist, ein Unterbrechungston abgespielt wird.

Angegeben werden interne Aktionen in einem eigenen Abschnitt innerhalb des Symbols für einen Zustand unterhalb des Namens des Zustands. Dort können mehrere Einträge der Form

ereignis / aktion

stehen. Als Angabe, wann und wie oft die angegebenen Aktionen ausgeführt werden sollen, sind folgende vordefinierte Schlüsselwörter für *ereignis* zulässig:

`entry`	einmal beim Eintritt in den Zustand
`exit`	einmal beim Verlassen des Zustands
`do`	ständig während des Aufenthalts in diesem Zustand

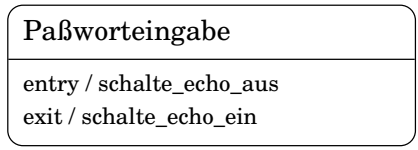

Abbildung 6: Zustand mit internen Aktionen

Bild 6 zeigt einen Zustand, in dem ein Paßwort abgefragt wird. Hierzu wird zunächst, um das Paßwort blind eingeben zu können, das Echo der Zeichen abgeschaltet, am Ende wird es wieder eingeschaltet.

Als *ereignis* können in der UML noch angegeben werden: Bedingungen wie bei bedingten Transitionen, Zeitangaben wie oben beschrieben, explizite Signale (auf die wir hier nicht eingehen), sowie Operationsaufrufe.

Literatur

[Ha87] Harel, D.: Statecharts: a visual formalism for complex systems; Science of Computer Programming 8, ISSN 0167-6423, Elsevier Science, p.231-274; 1987

[UML99] OMG Unified Modeling Language Specification (draft, Version 1.3 alpha R5, March 1999); OMG; 1999

[DMER] Kelter, U.: Lehrmodul "Datenmodellierung mit ER-Modellen"; 2003

[MSA] Kelter, U.: Lehrmodul "Moderne Strukturierte Analyse"; 2000/04

[SASM] Kelter, U.: Lehrmodul "Systemanalyse und Systemmodellierung"; 2002/01

Glossar

Aktion (*action*): Dienst, den die Umgebung des modellierten Systems zur Verfügung stellt

Aktion, interne: Konzept in der UML, durch das dargestellt werden kann, daß eine Aktionen ausgeführt wird, während sich das System in einem bestimmten Zustand befindet (vgl. Moore-Automat)

Anfangszustand: Zustand, in dem sich ein System initial befindet; in Zustandsübergangsdiagrammen durch einen dicken schwarzen Punkt und einen Pfeil von diesem Punkt auf den Anfangszustand gekennzeichnet

Endzustand (*final state*): Zustand, in dem kein weiterer Übergang auf einen anderen Zustand mehr möglich ist; in Zustandsübergangsdiagrammen optional durch einen umkreisten dicken schwarzen Punkt und einen Pfeil vom Endzustand zu diesem Punkt gekennzeichnet

Ereignis (*event*): Ereignisse werden von Instanzen außerhalb des modellierten Systems verursacht und lösen in dem System Zustandsübergänge aus

Ereignis, implizites: tritt ein, wenn im Falle eines Moore-Automaten sich die im aktuellen Zustand ausgeführte Operation beendet und im Falle eines Mealy-Automat und Zustandsverfeinerung im verfeinerten Zustand ein Endzustand erreicht wird; sofern in einem Zustand ein im-

plizites Ereignis eintreten kann, darf bei *einem* der von dort ausgehenden Zustandsübergänge das Ereignis fehlen, beim Eintreten eines impliziten Ereignisses wird diesem Übergang gefolgt

Mealy-Automat: endlicher Automat, bei dem das System zwischen Zustandsübergängen ruht und Aktionen atomar im Rahmen von Zustandsübergängen ausgeführt werden

Moore-Automat: endlicher Automat, bei dem das System in einem Zustand (also zwischen den Zustandsübergängen) eine bestimmte Aktion ausführt

Zustandsübergang, bedingter (*guarded transition*): Zustandsübergang, bei dem neben dem Ereignis eine zusätzliche Bedingung angegeben ist, die sich u.a. auf Datenwerte in Objekten beziehen kann (UML)

Zustandsübergangsdiagramm (*statechart diagram*): Diagramm, das die Zustände, die ein modelliertes System annehmen kann, die Zustandsübergänge, die diese auslösenden Ereignisse und die ausgelösten Aktionen darstellt; stellt i.w. einen endlichen Automaten dar

Index