Hierarchical State Machines*

Alexej Schatz

20. Dezember 2004

Inhaltsverzeichnis

1 Motivation

Die meisten Systeme, die auf bzw. mit dem PC implementiert werden, sind reaktiv. Ihr Verhalten lässt sich somit durch ihre Antworten auf die externe Ereignisse darstellten. Beispiele für solche Systeme sind: Betriebssysteme, Prozeßlenkungssysteme für diskrete Prozesse, u.s.w. Die Komplexität dieser Systeme besteht nicht so sehr in der möglichen großen Anzahl der Ereignisse, auf die das System reagieren muss, und der entsprechenden Antworten auf diese, sondern in der Anzahl der möglichen Nebenbedingungen unter welchen eine bestimmte Antwort ausgewählt wird. Eine direkte Umsetzung mit Hilfe der *if-else*-Konstruktionen würde eine entsprechend große Anzahl und tiefe Verschachtelung dieser erfordern. Dies könnte dazu führen, dass man nicht mehr nachvollziehen

*Ausarbeitung im Rahmen des Seminar "Humanoide Roboter" von Robert Haschke an der AG Neuroinformatik der Universität Bielefeld

kann, wieso das System zu einer bestimmten Antwort gekommen ist, oder auch die Wartung bzw. Erweiterung des Systems erschweren. Eine der möglichen Abhilfen sind die Zustandsautomaten mit UML als Modellierungswerkzeug.

2 Moore- und Harel-Automaten

Da die normalen Zustandsautomaten der sogenannten **Moore**-Automaten, und die hierarchische der **Harel**-Automaten entsprechen, möchte ich hier diese formal definieren.

Def. 1 Ein **Moore-Automat** ist ein 6-Tupel M=$(Q, \Sigma, T, \delta, \lambda, q_0)$ und besteht aus:

1. einem endlichen Alphabet Q von Zuständen,
2. einer endl. Menge Σ von Ereignissen ($Q \cap \Sigma = \emptyset$),
3. einer Übergangsfunktion $\delta : Q \times \Sigma \rightarrow Q$,
4. einem Startzustand $q_0 \in Q$,
5. einem endlichen Ausgabealphabet T,
6. einer Ausgabefunktion $\lambda : Q \rightarrow T$.

Def. 2 Ein **Harel-Automat** ist ein 7-Tupel H=$(Q, \Sigma, T, \Pi, \delta, \lambda, q_0)$ und besteht aus:

1. einer Zustandsmenge Q, die hierarchisch organisiert werden kann,
2. einer endl. Menge Σ von Ereignissen ($Q \cap \Sigma = \emptyset$),
3. eine Menge von Prädikaten Π für Wächter,
4. einem endl. Ausgabealphabet T,
5. einem Startzustand $q_0 \in Q$,
6. einer Übergangsfunktion $\delta : Q \times \Sigma \times \Pi \rightarrow T \times Q$,
7. einer Ausgabefunktion $\lambda : Q \rightarrow T$.

Aus diesen Definitionen ist leicht ersichtlich (durch Umformen der Zustandsmenge Q, und Übergangsfunktion δ des Harel-Automaten), dass der Harel-Automat eine Erweiterung des Moore-Automaten ist, der damit auch die gleiche Mächtigkeit wie der Moore-Automat besitzt.

3 Zustandsautomaten in UML

Im Gegensatz zum vorigen Abschnitt, wo die die Zustandsautomaten formal definiert wurden, werden ich hier die einzelne Bestandteile der Zustandsautomaten und ihre Repräsentation in UML vorstellen.

3.1 Elementare Bestandteile des Zustandsautomaten

Ein **Zustand** ist eine Situation im Leben des System, in der es eine Aktivität ausführt oder auf ein Ereignis wartet. Zustände werden in UML als Rechtecke mit abgerundeten Ecken dargestellt (siehe dazu Abb. 1 links). Ein Zustand besitzt mehrere Bestandteile: Name, Eintritts- bzw. Austrittsaktionen, Aktivitäten und verzögerte Ereignisse. Außerdem gibt es zwei Pseudozustände: der **Anfangszustand**, der den normalen Startzustand des Automaten oder Teilzustandes angibt, und der **Endzustand**, der den Endzustand des Automaten oder Teilzustandes angibt. Diese Zustände können keine der oben aufgezählten Bestandteile besitzen, außer einem Namen und einem Zustandsübergang für den Endzustand.

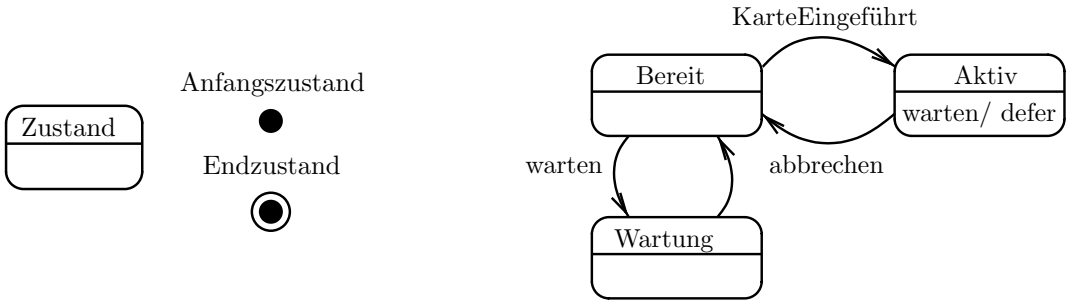

Abbildung 1: Links: Bsp. für ein Zustand und einen Anfangs- bzw. Endzustand. Rechts: Bsp. für einen Zustandsübergang und Ereignis.

Zustandsübergänge definieren eine Beziehung zwischen zwei Zuständen: falls der Automat sich in einem Zustand befindet und es tritt ein Ereignis ein, so gibt der Zustandsübergang den nachfolgenden Zustand an. Demzufolge besteht ein Zustandsübergang aus folgenden Bestandteilen: Ausgangszustand, auslösendes Ereignis, mögliche Wächterbedingung, mögliche Aktion und dem Folgezustand. Zustandsübergang wird in UML mit einem Pfeil dargestellt, der vom Ausgangszustand auf den Folgezustand zeigt. Als Beispiel ist in Abb. 1 rechts ein Zustandsautomat für ein Geldautomat angegeben. Dieser kann sich in drei Zuständen befinden: *Bereit* (auf Ereignis wartend), *Aktiv* (Kundentransaktionen verarbeiten) und *Wartung*. Wenn der Automat sich im Zustand *Bereit* befindet und es tritt der Ereignis *KarteEingeführt*, so wird ein Zustandsübergang nach *Aktiv* initiiert.

Ereignisse sind systemabhängige Stimuli, die zu einem Zustandsübergang führen können, als solche werden sie in UML durch ihren Namen über dem Zustandsüberganpfeil dargestellt (s. Abb. 1 rechts). Für den Fall, dass der eingetroffene Ereignis vom aktuellen Zustand nicht verarbeitet werden soll, aber auch nicht verlorengehen darf, gibt es die sog. **verzögerte Ereignisse**. Verzögerte Ereignisse werden gespeichert und später, wenn das System sich in einem für diesen Ereignis zuständigen Zustand befindet, bearbeitet, und zwar so als ob dieser gerade eingetroffen ist. Beispiel für einen verzögerten Ereignis ist der Ereignis *warten* im Zustand *Aktiv* in Abb. 1 rechts.

Aktionen werden mit einem Zustandsübergang assoziiert. Sie sind ausführbare atomare Einheiten, d.h. können nicht unterbrochen werden, und finden deshalb in einer

kurzen Zeitspanne statt. Da sie zu einem Zustandsübergang angehören, werden sie in UML hinter dem auslösendem Ereignis, durch einen „/“ getrennt, angegeben. In Abb. 2 links ist ein Automat für Heizung angegeben, wenn der Automat im Zustand *warten* ist und eine minimale Temperatur erreicht wird, so wechselt er in den Zustand *heizen* und führt die Aktion *HeizungEin* aus.

Für den Fall, dass bestimmte Aktionen immer beim verlassen bzw. betreten eines Zustandes ausgeführt werden müssen, gibt es die **Eintritts- bzw. Austrittsaktionen**. Diese gehören jetzt dem Zustand selbst an, und werden in UML mit Hilfe der Schlüsselwörter `entry` bzw. `exit` angegeben. Als Beispiel ist in Abb. 2 rechts ein Automat für Mikrowelle angegeben. Da die Mikrowelle immer ausgeschaltet werden muss, ob man die Tür öffnet oder die Zeit abläuft, kann diese Bedingung durch die Eintrittsaktion im Zustand *Aus* erreicht werden.

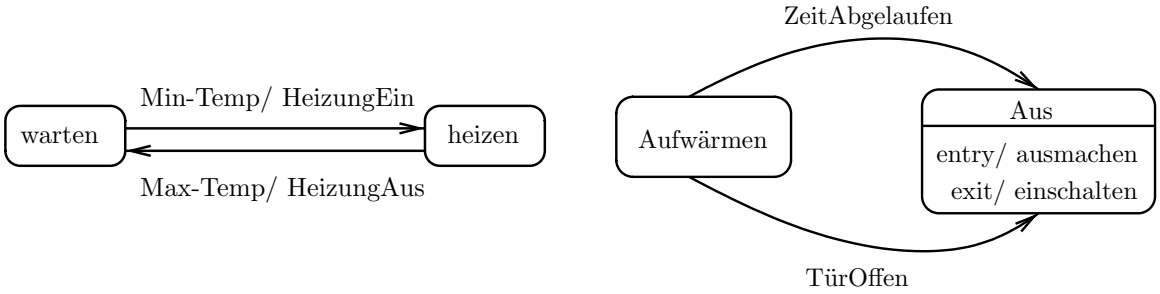

Abbildung 2: Links: Bsp. für Aktionen. Rechts: Bsp. für Eintritts- und Austrittsaktionen.

Aktivitäten sind Arbeiten, die das System während eines Zustandes ausführt. Sie haben im Vergleich zu Aktionen lange Dauer, und können jederzeit unterbrochen werden, wenn ein Zustandsübergang eingeleitet wird. In UML werden Aktivitäten mit dem Schlüsselwort `do` angegeben. Als Beispiel ist in Abb. 3 links Automat für einen Wecker: beim Erreichen des Zustandes *Wecken*, wird die Aktivität *Signal* ausgeführt.

Manchmal möchte man, dass ein Zustandsübergang nur dann initiiert wird, wenn Parameter, die der auslösende Ereignis bringt, bestimmte Werte aufweisen, in solchen Fällen verwendet man die *Wächterbedingungen*. Damit ist Wächterbedingung ein boolscher Ausdruck, der den Zustandsübergang nur im Falle „wahr“ erlaubt. Es kann auch mehrere Wächterbedingungen für ein Ereignis geben, aber nur wenn sie nicht gleichzeitig „wahr“ sein können, und damit auch mehrere Folgezustände für denselben Ereignis. Wächterbedingung steht in eckigen Klammern hinter dem zugehörigen Ereignis. In Abb. 3 rechts ist ein Automat für eine Tastatur angegeben, die sich in den Zuständen *shifted* und *default* befinden kann (groß- oder klein Schreiben der Buchstaben), in beiden Zuständen wird beim Drücken einer Taste überprüft, ob es die *Caps-Lock* ist, und nur in diesem Fall ein Zustandsübergang initiiert.

3.2 Teilzustände

Die hierarchische Struktur der Zustände wird mit Hilfe der sog. **Teil-** und aus ihnen **zusammengesetzten Zuständen** organisiert, wobei es drei Typen von Teilzuständen

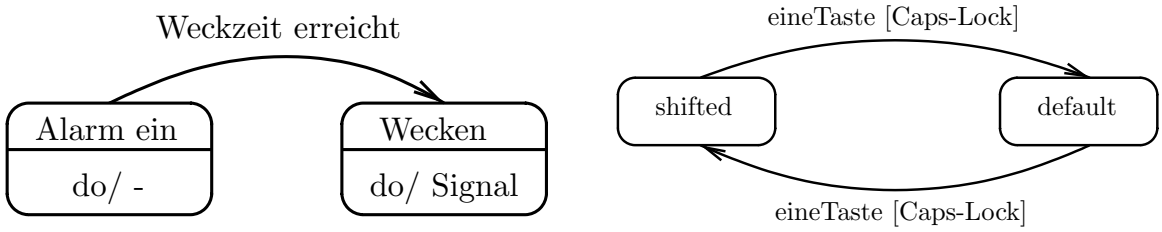

Abbildung 3: Links: Bsp. für ein Zustand. Rechts: Bsp. für einen Zustandsübergang und Ereignis.

gibt: **sequentielle Teilzustände**, **Erinnerungszustände** und **nebenläufige Teilzustände**.

3.2.1 Sequentielle Teilzustände

Wenn wir den alten Beispiel für einen Geldautomaten in Abb. 1 rechts nochmal betrachten, so könnte man den Zustand *Aktiv* in weitere Zustände aufspalten, um die Bearbeitung der einzelnen Kundentransaktionen übersichtlicher zu gestalten. Dann hätte man z.B. die Zustände: *Prüfen*, in dem die Karte überprüft wird, *Auswählen*, in dem der Kunde eine Transaktion auswählt, *Verarbeiten*, hier wird Transaktion vom System verarbeitet, und *Drucken*, in dem der Kontostand angezeigt wird. Diese Aufspaltung bringt aber ein Problem mit sich, denn der Kunde sollte die Möglichkeit haben, jeder Zeit abzubrechen, was den Automaten fehleranfälliger (beim Modellieren durch den Menschen) und unübersichtlicher (für den Menschen) macht, da viel mehr Zustandsübergänge benötigt werden. Aus diesem Grund werden diese Zustände zu einem weiteren Automaten zusammengefasst s. Abb. 4, der sich im Zustand *Aktiv* befindet, deswegen ist der neue Automat ein sequentieller Teilzustand. Die allgemeinen Eigenschaften gehören dann dem zusammengesetzten Zustand (hier *Aktiv*) an, und werden an die Teilzustände weitervererbt. D.h. befindet sich der Automat im Zustand *Auswählen* und es tritt der Ereignis *abbrechen* ein, so wird dieser an den zusammengesetzten Zustand weitergegeben und sofort verarbeitet.

3.2.2 Erinnerungszustände

Mit Hilfe eines Erinnerungzustandes kann angegeben werden, dass der zusammengesetzte Zustand in dem Teilzustand die Verarbeitung fortsetzt, in dem er als letztes war, außer der zusammengesetzte Zustand wurde bis zum Ende durchgelaufen, dann wird im definierten Startzustand fortgesetzt. Es gibt zwei Arten von Erinnerungzuständen, die nach Erinnerungstiefe zu unterscheiden sind. Der erste wird durch H in einem Kreis repräsentiert und erinnert sich nur an den Teilzustand des zusammengesetzten Zustands zu dem er gehört, der zweite durch H* in einem Kreis und erinnert sich auch in welchem Teilzustand der Teilzustand des zusammengesetzten Zustands war, und das so tief wie der Automat aufgebaut ist.

Betrachten wir als Beispiel einen Softwareagenten für Datensicherung in Abb. 5 links. Als Hauptzustände hat er: *Command*, Verarbeitung der Benutzeranfragen, und *Backin-*

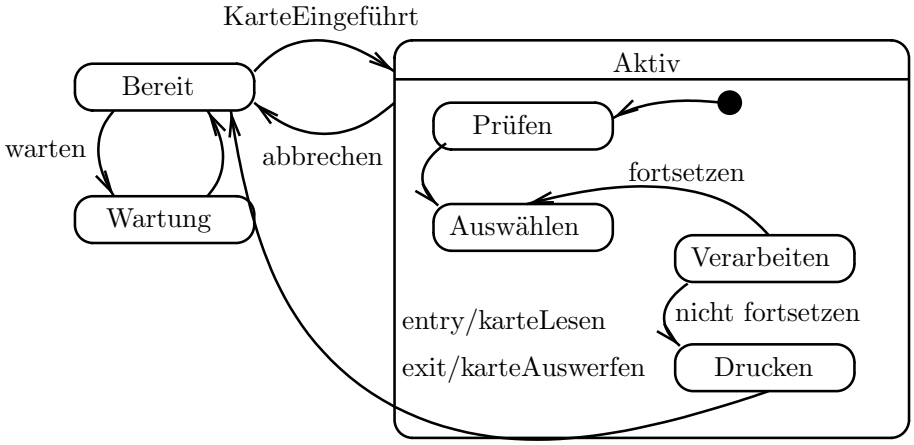

Abbildung 4: Bsp. für einen sequentiellen Teilzustand.

gUp. Der Sicherungsvorgang wird durch einen sequenziellen Zustand aus *Collecting*, *Copying*, *CleaningUp* und dem Erinnerungszustand, organisiert. Wenn der Agent sich im Zustand *Copying* befindet, und eine Anfrage vom Benutzer kommt, wechselt der Agent in Zustand *Command* und nach Bearbeiten der Anfrage wieder in Zustand *Copying*, wo er die Sicherung der Daten fortsetzt.

3.2.3 Nebenläufige Teilzustände

Nebenläufige Teilzustände bestehen aus mehreren Teilautomaten, die parallel abgelaufen werden. Als Beispiel betrachten wir den Geldautomaten in Abb. 5 rechts. Wenn der Automat sich im Zustand *Wartung* befindet, so werden die sequentielle Teilzustände *Testen* un *Steuern* simultan abgelaufen, bis beide ihren Endzustand erreicht haben, und geht anschließend in den Zustand *Bereit* über. Auch die nebenläufige Teilzustände erben die Eigenschaften der aus ihnen zusammengesetzten Zustandes.

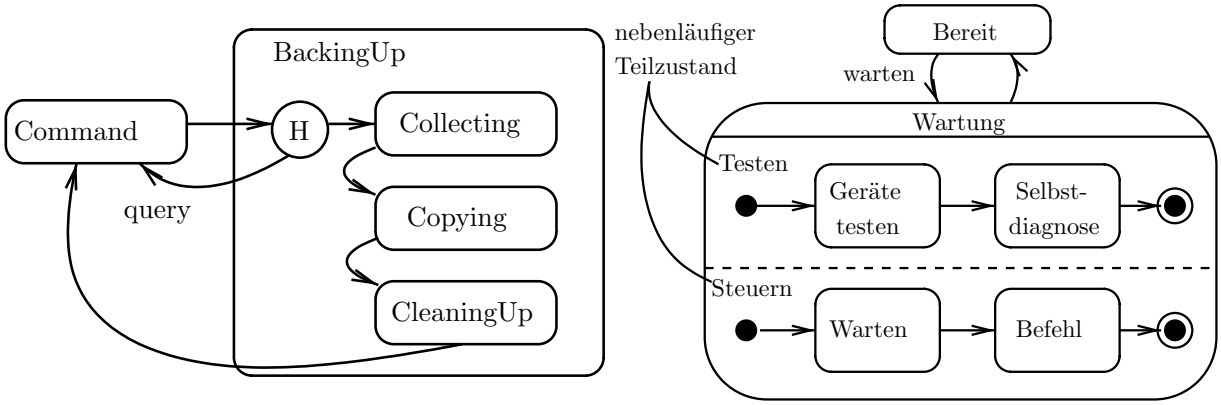

Abbildung 5: Links: Bsp. für einen Erinnerungzustand. Links: Bsp. für einen nebenläufigen Teilzustand.

4 Zustandsautomaten und OOP

Bis jetzt haben wir gesehen, was die Zustandsautomaten sind und wir sie mit UML modelliert werden können. Hier werde ich auf die Gemeinsamkeiten der Zustandsautomaten und der OOP eingehen, und damit auch eine Implementierungsmöglichkeit vorstellen.

Eine der wichtigsten Eigenschaften der objektorientierten Programmierung ist die Vererbung. Vererbung definiert den Verhältnis zwischen zwei oder auch mehr Klassen: eine neue Klasse wird auf Basis bereits vorhandener definiert, und erbt ihre Attribute und Methoden, die auch überlagert werden können.

Bei den hierarchischen Zustandsautomaten gibt es auch eine Vererbung zwischen den Zuständen, die Verhaltensvererbung ("behavioral inheritance"). Auf diese Art der Vererbung wurde bereits im Abschnitt 3.2 eingegangen und sie besagt, dass die Teilzustände sich wie die zusammengesetzte Zustände verhalten, und damit nur die Abweichungen bzw. Erweiterungen dieses Verhaltens im Teilzustand angegeben werden müssen. Damit entspricht die Verhaltesvererbung der "is-a" ("is-in") Beziehung und steht direkt im Bezug zur Vererbung zwischen den Klassen.

Eine andere Gemeinsamkeit ist die Ähnlichkeit zwischen Instantiierung bzw. Zerstörung einer Klasse, und den Eintritts- bzw. Austrittsaktionen. Bei Instantiieren einer Klasse muss der Konstruktor aufgerufen werden, was den Eintrittsaktionen entspricht, und beim Zerstören der Destruktor, entspricht den Austrittsaktionen.

5 Zusammenfassung

Die Zustandsautomaten eignen sich, wegen ihrer Nähe der menschlichen Denkweise, wegen der Hierarchisierung, als Werkzeug für eine einfache Modellierung und eventuell später notwendigen Wartung bzw. Erweiterung der reaktiven Systeme. Der Hierarchisierung ist es auch zu verdanken, dass man bei Implementierung die OOP mit ihren Vorzügen gegenüber der prozeduralen Programmierung benutzen kann.

Literatur

[1] Grady Booch, James Rumbaugh, Ivar Jacobson (2002). Das UML-Benutzerhandbuch.)

[2] Quantum Leaps: http://www.quantum-leaps.com/

[3] Vorlesungsskrip "Einführung in die Informatik II", Prof. Bernd Brügge, Ph.D.: http://atbruegge27.informatik.tu-muenchen.de/teaching/ss01/Info2/vorlesung/folien/09_Automaten_4.pdf

[4] Vorlesungsskript "Prozesslenkung", Prof. Dr. Wolfgang Fohl: http://users.etech.haw-hamburg.de/users/fohl/pl/wse6harel.pdf