

8 Lösungen für fortgeschrittene Probleme

8.1 Zustandsraumbasiertes Testen

Ein beliebter Tummelplatz für Fehler sind Zustandsübergänge. Ein Zustand ist eine fachlich motivierte Abstraktion einer Menge möglicher Werte eines Modellelements wie z. B. einer Klasse, wobei deren Werte in den Attributen abgelegt werden. Ein Zustandsübergang, also eine Transition von einem Zustand in einen anderen, wird durch ein Ereignis ausgelöst.

Zustandsraumbasiertes Testen können wir unabhängig von prozeduraler oder objektorientierter Programmierung immer dann einsetzen, wenn wir ein zustandsbasiertes Regelwerk implementieren. Ein solches System bezeichnen wir auch als *Zustandsautomaten*. Er besteht aus einer endlichen Anzahl interner Zustände [3]. Wir werden im Zusammenhang mit objektorientierten Testmustern in Abschnitt 9.3.1 ab Seite 112 noch einmal auf dieses Thema zurückkommen.

Zustände können mit der UML modelliert werden. Die entsprechenden Elemente sind Abb. 8.1 zu entnehmen. Im Wesentlichen basiert die Zustandsmodellierung immer noch auf einem Artikel von David Harrel von 1987 [39].

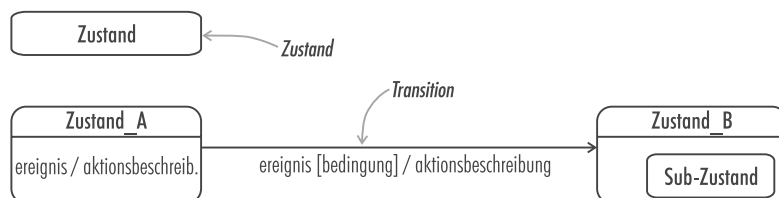


Abbildung 8.1: Die zentralen Elemente der Zustandsmodellierung mit der UML.

Ein Zustand kann Unterzustände haben und Aktionen auslösen, die an bestimmte Ereignisse geknüpft sind. Ein Zustandsübergang (Transition) ist ebenfalls mit einem Ereignis verbunden, welches zusätzlich durch eine Be-

dingung geschützt sein kann. Transitionen können auch Aktionen auslösen. Aktionen sind z. B. das Senden eines Signals oder der Aufruf von Methoden.

Ein Zustand kann also für definierte Ereignisse bestimmte Aktionen auslösen. Drei solcher zustandsinternen Ereignisse sind bereits vordefiniert:

do Während der Zustand aktiv ist, wird die beschriebene Aktion ausgeführt.

entry Beim Eintreten in diesen Zustand wird die beschriebene Aktion ausgeführt.

exit Beim Verlassen des Zustands wird die beschriebene Aktion ausgeführt.

Die Transitionen werden von Ereignissen ausgelöst und können an Bedingungen geknüpft sein. Diese Bedingung wird auch als *Guard* bezeichnet. Eine Transition kann eine Aktion auslösen, z. B. das Senden eines Signals.

Die möglichen Werte eines Modellelements wie z. B. Klasse spannen einen mehrdimensionalen Raum auf, den Zustandsraum (Abb. 8.2). Eine Klasse legt ihre Zustände in ihren Attributen ab, wobei sich ein bestimmter Zustand aus einer Kombination verschiedener Attribute zusammensetzen kann. Im Zustandsraum gibt es erlaubte Übergänge von einem Zustand in den anderen und verbotene Wege, die mögliche Fehler im späteren Programm bilden können, die Schleichpfade. Im Zustandsraumbasierten Test prüfen wir die erlaubten und verbotenen Übergänge. Gerade die möglichen Schleichpfade werden nur zu gerne beim Test vergessen.

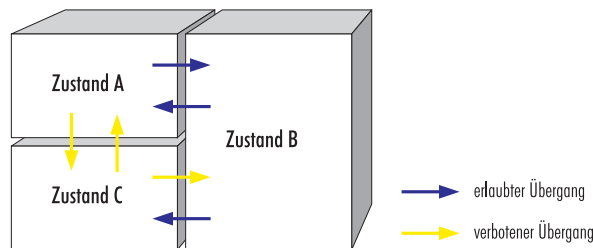


Abbildung 8.2: Schematische Darstellung eines Zustandsraums mit erlaubten und verbotenen Zustandsübergängen. Zwischen Zustand A und B kann hin und her gewechselt werden, aber nur aus B heraus können wir in Zustand C gelangen, der den Endzustand darstellt.

Ein einfaches Beispiel für ein Zustandsmodell finden wir in Abb. 8.3. Dort ist das Modell für einen Kassettenrecorder dargestellt. O.k., Kassettenrecorder sind im Zeitalter der CD-Brenner etwas aus der Mode gekommen, aber das Beispiel ist aussagekräftig und dennoch einfach zu durch-

dringen. Außerdem handelt es sich um einen modernen Kassettenrecorder mit Sensortasten ähnlich einem CD-Player.

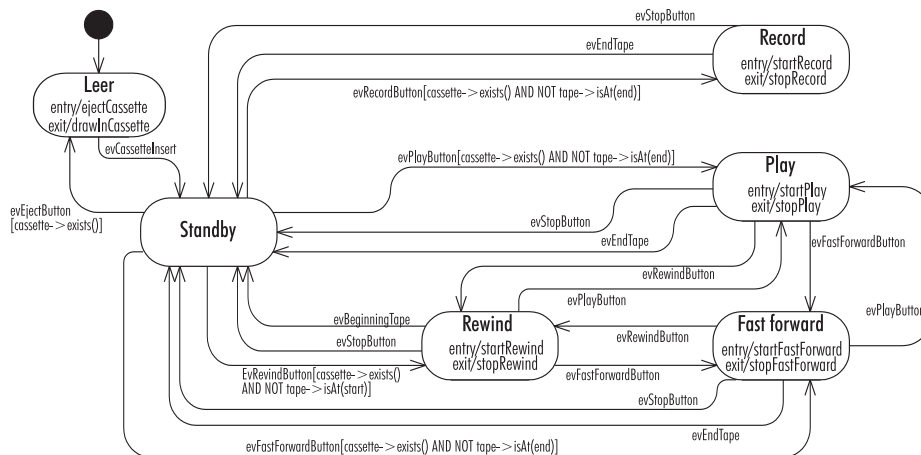


Abbildung 8.3: Zustandsmodell für einen Kassettenrecorder

Nach einem Zustandsmodell können wir nur schwer Tests definieren. Es bietet dafür keine gute Basis. Besser ist es, das Modell in einen Zustandsbaum zu überführen und eine Zustandsübergangstabelle abzuleiten.

Eine Zustandsübergangstabelle erzeugen wir, indem wir alle möglichen Zustände nebeneinander schreiben und an den Kopf jeder darunter liegenden Zeile die möglichen Ereignisse. In die Tabellenelemente können wir jetzt die resultierenden Zustände eintragen. Ein Beispiel für eine vereinfachte Form einer Zustandsübergangstabelle ist Tab. 9.1 auf Seite 115 zu entnehmen. Wie das für unser Kassettenrecorder-Beispiel aussehen kann, ist in Tab. 8.1 zu sehen.

Um sie später besser referenzieren zu können, sind in der Tabelle die Zustandsübergänge durchnummeriert. Übergänge, die an spezielle Bedingungen geknüpft sind, also durch Bedingungsprüfungen geschützt sind, haben wir dabei mit einem *g* für *Guard* markiert. Für die gültigen Übergänge sind die auslösenden Ereignisse angegeben, wobei die Tabelle von der Kopfzeile aus zu den einzelnen auslösenden Ereignissen in der ersten Spalte zu lesen ist. Der resultierende Zustand steht dann in dem Tabellenschnittpunkt. Die Nummern identifizieren die Übergänge eindeutig für den folgenden Zustandsübergangsbaum (Abb. 8.4).

Auf Basis einer Zustandsübergangstabelle kann versucht werden, diese als Tabelle zu kodieren und im Code bei der Behandlung eintreffender Ereignisse gegen diese Tabelle zu prüfen. Die korrekte Implementierung und Anpassungen im weiteren Projektverlauf sollten natürlich getestet werden.

	Leer	Standby	Rewind	Play	Fast Forward	Record
evEject-Button	×	2 (g) Leer	×	×	×	×
evRewind-Button	×	3 (g) Rewind	×	11 Rewind	15 Rewind	×
evPlayButton	×	4 (g) Play	7 Play	×	16 Play	×
evFastForward-Button	×	5 (g) FFwd	8 FFwd	12 FFwd	×	×
evRecord-Button	×	6 (g) Record	×	×	×	×
evStopButton	×	×	9 Standby	13 Standby	17 Standby	19 Standby
evEndTape	×	×	×	14 Standby	18 Standby	20 Standby
evBeginning-Tape	×	×	10 Standby	×	×	×
evInsert-Cassette	1 Standby	×	×	×	×	×

Tabelle 8.1: Die Zustandsübergänge aus dem Zustandsmodell eines Kassettenrecorders aus Abb. 8.3 als Wahrheitstabelle. Durch Bedingungen geschützte Übergänge sind durch (g) gekennzeichnet, ungültige durch ×.

Wir können jetzt beginnend mit dem initialen Zustand *Leer* jeden erlaubten Übergang als Transition zeichnen. So entsteht ein *Zustandsbaum*. Wir beenden eine Transitionsfolge, also einen Ast unseres Baums, wenn wir auf einen Zustand stoßen, den wir vorher bereits schon einmal hatten. Die einzelnen Transitionen werden mit den Nummern aus der Tab. 8.1 benannt. Für unser Kassettenrecorder-Beispiel ist der Zustandsbaum in Abb. 8.4 dargestellt. Ein Zustandsbaum ist kein eigenes UML-Diagramm. Er kann aber mit den Hilfsmitteln der UML beschrieben werden, wie in Abb. 8.4 geschehen. Auf Grundlage der Tabelle und des Baums können wir jetzt das Zustandsmodell vollständig testen. Wir gehen dabei in zwei Schritten vor.

1. Wir testen alle erlaubten Übergänge, indem wir mit unseren Tests jeden Ast unseres Zustandsbaums durchlaufen.
2. Aus der Tabelle entnehmen wir alle verbotenen Übergänge und prüfen, dass diese auch wirklich nicht möglich sind. Ansonsten haben wir einen verbotenen *Schleichpfad* gefunden.

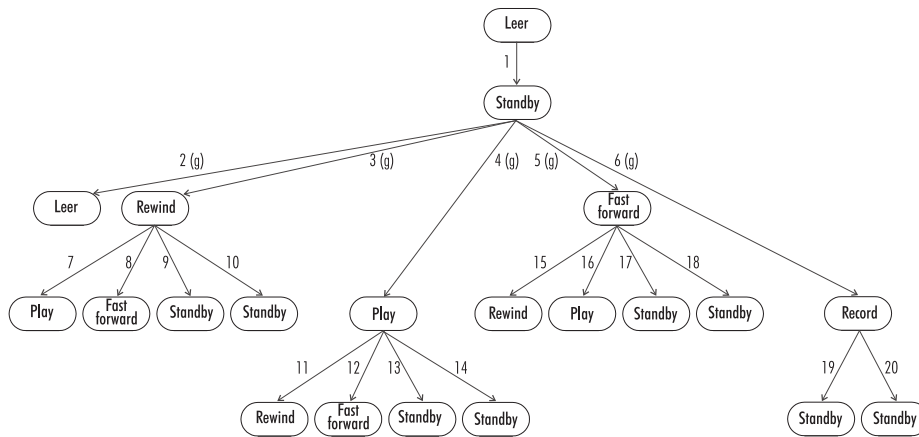


Abbildung 8.4: Aus dem Zustandsmodell aus Abb. 8.3 abgeleiteter Zustandsbaum für einen Kassettenrecorder.

Leider gibt es keine sinnvolle Vereinfachung für diese beiden Testschritte. Wir sollten sie stets vollständig ausführen. Anderenfalls kann uns hier ein schwerer Fehler durch unsere Tests rutschen!

8.2 Rekursion und Nebenläufigkeit

Besondere Probleme für den Test können rekursive Algorithmen oder Nebenläufigkeiten aufwerfen. Bei der Nebenläufigkeit ist das offensichtlich: Parallele Prozesse oder Threads sind komplex und damit per se fehleranfällig. Rekursive Algorithmen sind dagegen mathematisch klar, elegant und eindeutig und von daher eigentlich recht robust. Aber auch hier kann der Teufel im Detail stecken.

8.2.1 Rekursive und iterative Algorithmen

Eine Prozedur, die sich selbst aufruft, heißt *rekursiv*. Der Aufruf erfolgt dabei entweder direkt oder über andere Prozeduren bzw. Methoden indirekt (Abb. 8.5).

Ein rekursiver Algorithmus kann auch nicht-rekursiv programmiert werden. Generell sollten wir uns fragen, ob wir überhaupt eine rekursive Implementierung haben wollen oder nicht lieber den Algorithmus in eine Iteration umformulieren. Als Beispiel, wie die beiden unterschiedlichen Implementierungen aussehen, habe ich einen Klassiker ausgewählt, den Quicksort-Algorithmus nach C. A. R. Hoares [55, 91].