

Entsprechend den verschiedenen Charakteristiken eines zu simulierenden Systems unterscheidet man statische und dynamische, kontinuierliche und diskrete sowie deterministische und stochastische Simulationstechniken. Wir wollen uns hier mit *dynamischen, diskreten und stochastischen Simulationen* beschäftigen, die softwaretechnische Lösungen für die folgenden Aufgabenstellungen bereitstellen:

- Nachbildung des Systemverhaltens im Ablauf der Zeit,
- Berechnung der Zustandsänderungen in diskreten Zeitpunkten und
- Modellierung von zufälligen Phänomenen.

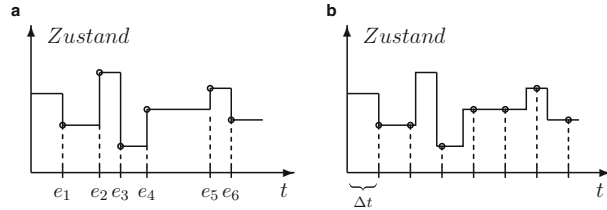
Bei komplexen Problemstellungen, die einer geschlossenen mathematischen Behandlung nicht zugänglich sind, ist es oftmals möglich, den Zustand des Systems durch einfache Berechnungen Schritt für Schritt fortzuschreiben und dadurch das zeitliche Systemverhalten nachzuspielen. Mit Hilfe von Computern kann auf diese Weise in angemessener Zeit ein genügend langer Zeitraum simuliert werden, der es erlaubt, Gesetzmäßigkeiten des Systems abzuleiten.

Die zufälligen (stochastischen) Größen eines Systems kommen meist mehrmals in unterschiedlicher Ausprägung vor, deshalb können sie als *Zufallsvariablen* aufgefasst werden (vgl. Abschn. 8.2). Ist die Wahrscheinlichkeitsverteilung einer Zufallsvariablen bekannt, so kann sie mit einem Zufallszahlengenerator im Rechner simuliert werden. Mit einer genügend großen Anzahl von Messwerten, die aus einem sehr langen Simulationslauf oder aus vielen einzelnen Simulationsläufen gewonnen werden, kann ein Modell mit stochastischen Größen experimentell erschlossen und statistisch beurteilt werden.

Die systematische Durchführung von Simulationsexperimenten zur Gewinnung von neuen, statistisch gesicherten Erkenntnissen über ein stochastisches System unter Anwendung von Zufallszahlengeneratoren wird auch als *Monte-Carlo-Simulation* bezeichnet, da einzelne Ergebnisse genauso wie beim Roulette nicht vorhergesehen werden können.

Legt man einen kontinuierlichen Zeitparameter zugrunde, so müssen zur Durchführung einer diskreten Simulation geeignete Zeitpunkte gewählt werden, in denen der Zustand des Systems neu berechnet wird. Es gibt dazu zwei Möglichkeiten: Entweder man wählt die Eintrittszeitpunkte der Ereignisse oder man verwendet ein

Abb. 2.1 a Ereignisorientierte, b periodenorientierte Zustandsberechnung



Zeitraster mit gleichlangen Zeitabständen. Das erste Prinzip wird als *ereignisorientiert* oder *ereignisgesteuert* bezeichnet, das zweite als *periodenorientiert* oder *zeitgesteuert* (Abb. 2.1).

Das Δt im Zeitraster beim periodenorientierten Vorgehen wurde in der Skizze zu groß gewählt, so dass ein Zustandswert zwischen dem zweiten und dritten Messpunkt nicht erfasst wird. Verwendet man ein zu kleines Δt , dann wird zu oft der Zustand neu berechnet, auch wenn er mehrere Perioden lang unverändert bleibt. Dies macht das Problem der sinnvollen Wahl des Rasterabstands deutlich. Bei der ereignisorientierten Vorgehensweise wird jede Zustandsänderung erfasst und es wird kein unnötiger Rechenaufwand betrieben.

Im Folgenden werden drei prinzipielle Techniken für die Simulation diskreter Prozesse vorgestellt, die ereignisorientierte, die prozessorientierte und die periodenorientierte Simulation. Wir beschränken uns auf diese drei Prinzipien, da sie stellvertretend die wichtigsten Aspekte der Simulation diskreter Prozesse abdecken. Für andere Verfahren verweisen wir auf die Literatur wie z. B. Banks et al. (2009) oder Page und Kreutzer (2005).

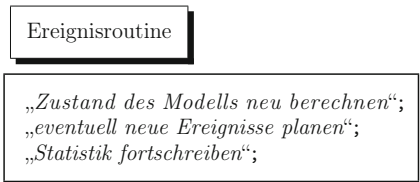
2.1 Ereignisorientierte Simulation

Die ereignisorientierte Sicht auf diskrete Prozesse bildet die Grundlage vieler Modellierungsfomalismen und Simulationsprogramme, die heute im praktischen Einsatz sind. Der große Vorteil liegt darin, dass das dynamische Verhalten eines Systems mit hoher Genauigkeit durch eine Software nachgespielt wird. Die Konzentration auf die Eintrittszeitpunkte von Ereignissen ermöglicht eine modulare Gestaltung der Simulationssoftware. Insbesondere bildet die ereignisorientierte Modellierung auch die Grundlage der im nächsten Abschnitt vorgestellten prozessorientierten Simulation.

2.1.1 Das Prinzip der ereignisorientierten Simulation

Bei der *ereignisorientierten Simulation* (engl. *discrete-event system simulation*, abgek. *DES*) werden alle Ereignisse, die beim Ablauf eines diskreten Prozesses eintreten, nachgespielt. Dabei wird für jedes Ereignis (engl. *event*) eine vom Ereignistyp abhängende *Ereignisroutine* ausgeführt. In Abb. 2.2 ist die prinzipielle Struktur einer Ereignisroutine dargestellt.

Abb. 2.2 Struktur einer Ereignisroutine



Eine Ereignisroutine ist ein Programmcode als Teil der Simulationssoftware, der im Allgemeinen drei Aufgaben erledigt: die Berechnung des neuen Zustands, der sich aus mehreren Komponenten zusammensetzen kann, die Planung neuer, in der Zukunft stattfindender Ereignisse, falls dies möglich ist, sowie die Durchführung statistischer Auswertungen.

Es gibt Situationen, in denen kein neues Ereignis geplant werden kann. In schwierigen Fällen müssen schon geplante Ereignisse verworfen und eventuell neu eingeplant werden. Ereignisroutinen werden nur für unabhängige Ereignisse erstellt. Sie enthalten alle Berechnungen, die für die abhängigen Ereignisse erforderlich sind.

Die für die Zukunft geplanten unabhängigen Ereignisse werden in der *Ereignisliste* (engl. *future event list*) verwaltet (Abb. 2.3). Jedes geplante Ereignis wird darin durch eine Datenstruktur repräsentiert, die alle für das Ereignis charakteristischen Informationen enthält, insbesondere den *Ereignistyp* und den *Eintrittszeitpunkt*. Die Ereignisliste liegt immer nach den Eintrittszeitpunkten aufsteigend sortiert vor.

Üblicherweise ist von jedem Ereignistyp nur ein Eintrag in der Ereignisliste vorhanden, denn erst bei der Bearbeitung dieses Ereignisses wird das nächste Ereignis desselben Typs geplant, in manchen Fällen sogar noch später.

Der Steuerungsalgorithmus einer ereignisorientierten Simulation ist in Abb. 2.4 dargestellt. Er ist völlig unabhängig von einer Anwendung.

Der Algorithmus besteht aus einer Schleife, die immer das erste Element `nextEvent` aus der Ereignisliste entfernt, die Simulationszeit `simZeit` aktualisiert und die zum jeweiligen Ereignistyp `typAktuell` gehörende Ereignisroutine ausführt. Das Programm endet, falls die vorgegebene Simulationsdauer `simEnde` erreicht bzw. überschritten wird.

Vor dem allgemeinen Ablauf werden in einem Initialisierungsteil die Simulationszeit auf 0 gesetzt, der Zeitpunkt für das Simulationsende festgelegt und die statistischen Werte initialisiert. Zudem müssen Anfangsereignisse geplant werden, damit der Steuerungsalgorithmus überhaupt in Gang kommen kann.

Abb. 2.3 Ereignisliste

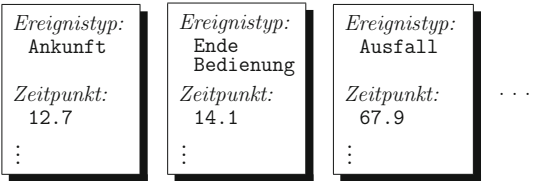


Abb. 2.4 Steuerungs-
algorithmus einer
ereignisorientierten Simu-
lation

```
while (simZeit < simEnde)
{
    „Hole das nächste Ereignis nextEvent aus der
    Ereignisliste und lösche diesen Eintrag“;
    simZeit = „Eintrittszeitpunkt von nextEvent“;
    typAktuell = „Typ von nextEvent“;
    „Führe die Ereignisroutine zu typAktuell aus“;
}
```

Abb. 2.5 Symbolische Dar-
stellung eines einfachen
Warteschlangensystems



Da bei der Durchführung der Ereignisroutinen normalerweise neue Ereignisse geplant werden, wird die Ereignisliste nicht leer und der Zyklus läuft so lange weiter, bis das Ende der Simulation erreicht wird.

Hängt der Eintrittszeitpunkt eines Ereignisses vom Zufall ab, so wird er mit Hilfe eines Zufallszahlengenerators bestimmt. Dies hat zur Folge, dass die Zielgrößen des Systems nur mit statistischen Methoden beschrieben werden können.

Die Ereignisliste kann als doppelt verkettete Liste mit geeigneten Einfüge- und Löschfunktionen verwaltet werden (siehe z. B. Banks et al. 2009). Noch effizienter sind *priority queues*, die aufgrund ihrer Baumstruktur eine besonders schnelle Verwaltung der Ereigniseinträge bieten (Sedgewick 2002).

2.1.2 Ereignisorientierte Simulation einer Warteschlange

Um einen ersten Eindruck zu vermitteln, wie eine ereignisorientierte Simulation programmiert wird, wollen wir ein einfaches Warteschlangensystem simulieren (Abb. 2.5). Das System besteht aus einer Kasse in einem Supermarkt, an der Kunden abgefertigt werden, sowie einer Warteschlange, die sich aufbaut, falls die Kunden nicht schnell genug abgefertigt werden.

Die Warteschlange wird immer nach der Reihenfolge „First In First Out (FIFO)“ abgearbeitet. Alle Kunden verbleiben in der Warteschlange, bis sie bedient werden. Die Kasse soll durch zwei Zustände charakterisiert sein: *aktiv*, falls gerade ein Kunde bedient wird, und *frei*, falls kein Kunde vorhanden ist. Mögliche Ereignisse sind die Ankunft eines Kunden und das Bedienungsende an der Kasse.

Zufällige Größen sind die Bedienzeiten und die Zeiten, die zwischen zwei Ankünften verstreichen, die Zwischenankunftszeiten. Die zufälligen Größen werden bei Bedarf durch einen Zufallszahlengenerator bereitgestellt.

In Abb. 2.6 ist der Ablauf der beiden Ereignisroutinen für die Ankunft- bzw. Bedienungsende-Ereignisse dargestellt. Da wir den Schwerpunkt auf die Ablauflogik setzen, lassen wir die Fortschreibung der Statistik jeweils weg.

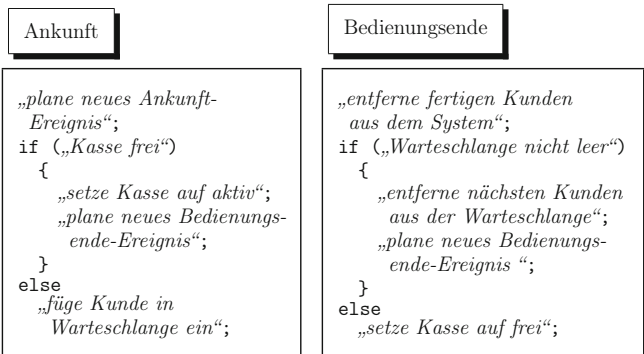


Abb. 2.6 Ereignisroutinen für Ankunft und Bedienungsende

Bei einem Ankunft-Ereignis planen wir sofort das nächste Ankunft-Ereignis, ohne dass wir die Quelle des Ankunftsstroms getrennt modellieren. Bei der Beendigung einer Bedienung wird das Kundenobjekt ohne weitere Berücksichtigung entfernt.

Wir wollen die ersten 10 Minuten des Verlaufs einer Simulation in Form einer Tabelle sichtbar machen (Tab. 2.1).

Die zufälligen Größen für die Zwischenankunftszeiten (Zufallsvariable *A*) und für die Bedienzeiten (Zufallsvariable *B*) sind den Zufallszahlenfolgen der Tab. 2.2 zu entnehmen (von links nach rechts). Diese Werte sollen als Zeitdauern in Minuten interpretiert werden.

Tabelle 2.1 enthält pro Spalte in den oberen beiden Feldern die Zeitpunkte der Ereignisse, die sich jeweils nach Durchlaufen der Schleife der Steuerungsroutine in der Ereignisliste befinden. Die Reihenfolge der Ereignisse kann anhand der Zeitpunkte leicht abgeleitet werden. Darunter sind der Zustand der Kasse und die Anzahl der Kunden in der Warteschlange eingetragen.

Jede Spalte der Tabelle repräsentiert die Ereignisliste sowie den Systemzustand zu dem im untersten Feld eingetragenen Zeitpunkt.

Die Spalte für den Zeitpunkt 0,0 wird als Initialisierung des Systems vorgegeben (grau unterlegt). Zu Beginn soll sich ein Kunde an der Kasse befinden und die Warteschlange soll leer sein. Die Zeitpunkte für die nächste Ankunft und für das Ende der aktuellen Bedienung sind schon geplant.

Der Übergang von einer Spalte zur nächsten Spalte erfolgt nach dem Prinzip der ereignisorientierten Simulation: Das Ereignis mit dem frühesten Zeitpunkt wird aus der Ereignisliste gelöscht, indem es nicht in die nächste Spalte übernommen wird. Der zugehörige Zeitpunkt wird als Simulationszeit in die nächste Spalte eingetragen. Abhängig vom Typ des gelöschten Ereignisses werden in der nächsten Spalte eventuell neue Ereignisse geplant und Zustandswerte aktualisiert, so wie es die entsprechende Ereignisroutine vorgibt.

In der Ereignisliste dieses Beispiels befindet sich immer genau ein zukünftiges Ankunft-Ereignis. Vom Typ Bedienungsende ist entweder ein Eintrag vorhanden

Tab. 2.1 Tabellarische Darstellung der ereignisorientierten Simulation eines einfachen Warteschlangensystems

<i>Ankunft</i>	0,6	0,7	4,9	4,9	4,9	7,4	7,4	7,4	10,0	13,1
<i>Bedienungsende</i>	2,3	2,3	2,3	3,1	4,9	4,9	6,1	–	10,2	10,2
Kasse	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>f</i>	<i>a</i>	<i>a</i>
Warteschlange	0	1	2	1	0	1	0	0	0	1
<i>Simulationszeit</i>	0,0	0,6	0,7	2,3	3,1	4,9	4,9	6,1	7,4	10,0

a = aktiv, *f* = frei

Tab. 2.2 Zufallszahlenströme für Ankunftsabstände und Bedienzeiten

<i>A</i>	0,1	4,2	2,5	2,6	3,1	2,9	0,8	0,7	6,9	5,3	...
<i>B</i>	0,8	1,8	1,2	2,8	1,9	1,5	3,3	1,8	1,7	2,1	...

oder keiner, falls das Warteschlangensystem leer ist. In diesem Fall wird das nächste Bedienungsende-Ereignis erst bei der nächsten Ankunft geplant.

Bei dem dargestellten Ablauf finden im Zeitpunkt 4,9 sowohl ein Ankunfts- als auch ein Bedienungsende-Ereignis statt. Da sich die beiden Ereignisse nicht gegenseitig beeinflussen, ist es egal, welches Ereignis zuerst bearbeitet wird. Wir haben hier zuerst das Ankunfts-Ereignis gewählt, um diesen Spezialfall nicht mit dem Spezialfall eines leeren Systems zu vermischen. Die Bearbeitung eines leeren Systems wird im Zeitpunkt 6,1 deutlich.

2.2 Prozessorientierte Simulation

Die elementare ereignisorientierte Simulation kann für große Systeme durch das Auftreten einer Vielzahl von Ereignissen an unterschiedlichen Orten schnell unübersichtlich werden. Eine strukturierte Sicht der Abläufe in einem dynamischen System liefert die *prozessorientierte Simulation* (engl. *process-interaction approach*), bei der man Prozesse als dynamische Einheiten, die sich auf ein Objekt beziehen, in modularer Weise verwaltet.

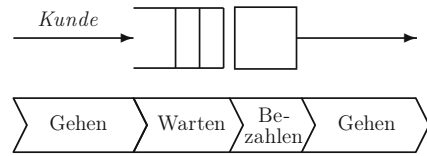
2.2.1 Prozessorientierte Simulation einer Warteschlange

Wir wollen das Prinzip der prozessorientierten Simulation anhand der Vorgänge an einer Supermarktkasse erklären (Abb. 2.7).

Die ankommenden Kunden betreten den Kassenbereich, der die räumliche Umgebung des zugrunde liegenden Systems darstellt. Sie gehen zunächst zur Warteschlange, warten im Normalfall, bezahlen und gehen dann zur Grenze des Kassenbereichs auf der anderen Seite, um das System zu verlassen.

Jedes ankommende Kundenobjekt definiert einen Prozess, der durch die Folge von Aktivitäten „Gehen, Warten, Bezahlen, Gehen“ gekennzeichnet ist. Bei leerem

Abb. 2.7 Prozess Kunde im Kassensbereich eines Supermarkts



System kann dabei das Warten wegfallen. Die zugehörige Folge von Ereignissen ist „Betreten des Systems = Gehen-Beginn, Gehen-Ende = Ankunft an der Kasse = Warten-Beginn, Warten-Ende = Bezahlen-Beginn, Bezahlen-Ende = Gehen-Beginn, Gehen-Ende = Verlassen des Systems“.

Die Aktivität „Bezahlen“ durch den Kunden verläuft völlig gleichzeitig mit der Aktivität „Kassieren“ an der Kasse. Die Kasse definiert einen Prozess, der aus einer Folge von Kassieren-Aktivitäten, eventuell unterbrochen durch Warten-Aktivitäten bei fehlenden Kunden, besteht.

Das Ereignis „Bezahlen-Beginn“ aus der Sicht eines wartenden Kunden ist abhängig vom Ereignis „Kassieren-Ende“, das die Kasse für den vorhergehenden Kunden ausführt. Auch das Bezahlen-Ende-Ereignis wird von der Kasse verursacht, d. h. sie übernimmt die aktive Rolle während des Kassiervorgangs.

Für jeden Kunden im System läuft ein eigener Prozess ab, gleichzeitig läuft der Prozess der Kasse ab. Im realen System finden alle diese Prozesse parallel statt. Wenn man in einem Rechner nicht für jeden Prozess einen eigenen Prozessor zur Verfügung hat, so ist man auf die Sequenzialisierung der Prozesse angewiesen, man spricht dann nur noch von *quasi-parallelen* Prozessen.

Üblicherweise geschieht die Sequenzialisierung paralleler Prozesse dadurch, dass man die Prozesse nur stückweise ausführt, wobei jeder Prozess nach vorgegebenen Vorrangsregeln abwechselnd an die Reihe kommt. Wir wollen hier einen Implementierungsansatz vorstellen, der in Page et al. (2000) und Page und Kreutzer (2005) beschrieben ist. Für einen Prozess unterscheidet man die folgenden drei Zustände:

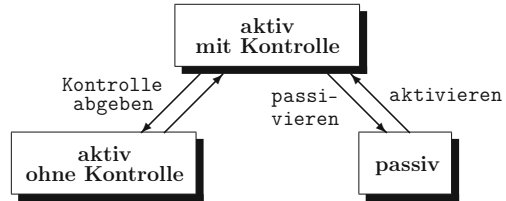
Aktiv mit Kontrolle (rechnend, engl. running): Befindet sich ein Prozess in einer aktiven Phase und ist sein nächstes Ereignis das global gesehen nächste Ereignis, so bekommt er die Kontrolle der Simulation übergeben. Die Simulationszeit wird auf den Eintrittszeitpunkt seines nächsten Ereignisses hochgesetzt, es wird der Zustand aller Objekte berechnet, die sich ändern, und der Prozess plant im Normalfall neue Ereignisse. Dann wird die Kontrolle an denjenigen Prozess übergeben, für den das neuerdings nächste Ereignis geplant ist.

Aktiv ohne Kontrolle (bereit, engl. ready): Wurde das nächste Ereignis für den Prozess schon geplant und müssen vorher noch Ereignisse anderer Prozesse abgearbeitet werden, so ist der Prozess zwar aktiv, wird aber erst später bearbeitet.

Passiv (wartend, engl. waiting): Während einer passiven Phase eines Prozesses liegt die Kontrolle der Simulation durchgehend bei anderen Prozessen. Er bleibt passiv, bis ein von einem anderen Prozess herbeigeführtes Ereignis eintritt, das ihn wieder aktiv werden lässt und er selbst sein nächstes Ereignis planen muss.

Abbildung 2.8 (nach Page und Kreutzer 2005) zeigt die möglichen Zustandsübergänge eines Prozesses zusammen mit entsprechenden Befehlen.

Abb. 2.8 Mögliche Zustände eines Prozesses



Die Befehle *Kontrolle abgeben*, *passivieren* und *aktivieren* werden durch Programmierbefehle im prozessspezifischen Programmcode formuliert. Die Übernahme der Kontrolle erfolgt durch die übergeordnete Simulationssteuerung (weshalb in der Abbildung dafür kein Befehl angegeben ist). Da immer nur ein Prozess die Kontrolle hat, wird verhindert, dass zwei parallel ablaufende Prozesse dieselbe Zustandsvariable verändern.

Die aktiven Prozesse werden in einer Liste verwaltet, in der für jeden aktiven Prozess das nächste geplante Ereignis vorgemerkt ist. Die Liste ist nach den Eintrittszeitpunkten der Ereignisse sortiert und wird wie die Ereignisliste bei der ereignisorientierten Simulation abgearbeitet, indem das erste Element entfernt wird und ein zugehöriger Codeteil ausgeführt wird. Die passiven Prozesse werden in Warteschlangen oder Zwischenpuffern abgelegt, bis sie aktiviert werden und in die Liste der aktiven Ereignisse aufgenommen werden.

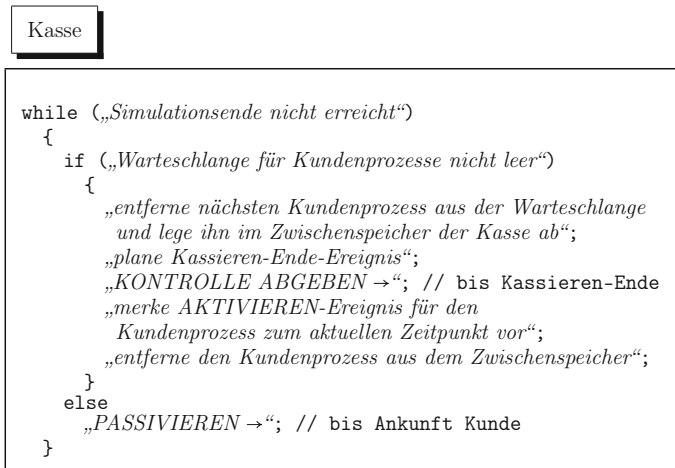
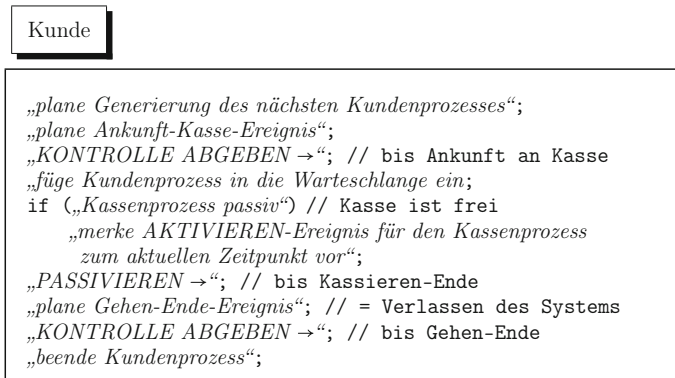
Im Supermarktbeispiel ist der Kundenprozess während des Wartens und während des Bezahlens passiv, dagegen ist der Kassenprozess in dieser Zeit aktiv. Erst durch das Ereignis „Kassieren-Ende“ wird der Kundenprozess wieder aktiviert und es wird das neue Ereignis *Gehen-Ende* für den Kunden geplant. Beim Verlassen des Systems bekommt der Kundenprozess noch einmal die Kontrolle, um das Objekt zu entfernen und um statistische Auswertungen durchzuführen.

Der Prozess der Kasse wird gleich zu Beginn der Simulation erzeugt und endet erst beim Simulationsende.

Für die Implementierung einer prozessorientierten Simulation werden die Prozesse durch *Prozessroutinen* beschrieben, in denen alles geregelt wird, was den jeweiligen Prozess betrifft. Für das Supermarktbeispiel werden zwei Prozessroutinen, die den Kassenprozess (Abb. 2.9) und den Kundenprozess (Abb. 2.10) modellieren, festgelegt.

Die in den Prozessroutinen verwendete Formulierung *Planen eines Ereignisses* für einen Prozess beinhaltet die Bestimmung des Eintrittszeitpunkts mit Hilfe einer geeigneten Zufallszahl sowie das Vormerken in der Ereignisliste.

Die Formulierung „*KONTROLLE ABGEBEN* → “ bedeutet, dass der Prozess unterbrochen wird und die Kontrolle einem anderen Prozess übergeben wird. Vorher wurde sein nächstes Ereignis geplant, so dass der Prozess aktiv bleibt. Bekommt der unterbrochene Prozess durch ein vorgemerkt Ereignis wieder die Kontrolle übergeben, so läuft seine Prozessroutine an der Stelle weiter, an der sie unterbrochen wurde.

**Abb. 2.9** Prozessroutine für den Prozess der Kasse**Abb. 2.10** Prozessroutine für den Prozess eines Kunden

Beim „*PASSIVIEREN* →“ wird ebenfalls die Kontrolle abgegeben und der Prozess wird unterbrochen, ohne dass sein nächstes Ereignis geplant ist. Er ist dann passiv und muss solange warten, bis er wieder aktiviert wird.

AKTIVIEREN-Ereignisse sind von rein technischer Natur und dienen der expliziten Aktivierung eines passiven Prozesses. Dies benötigt man für solche Fälle, in denen das Ende einer Aktivität des Prozesses, der gerade die Kontrolle inne hat, mit dem Beginn einer Aktivität eines passiven Prozesses zusammenfällt. Im Beispiel trifft dies auf das Kassieren-Ende-Ereignis des Kassenprozesses zu, das gleichzeitig mit dem Gehen-Beginn-Ereignis des Kundenprozesses stattfindet, oder auf das Ankunft-Ereignis bei freier Kasse, denn dann muss sofort das Kassieren gestartet werden.

Ist der Kassenbereich leer, so hängt das nächste Kassieren-Beginn-Ereignis vom Ankunft-Ereignis des nächsten Kunden ab. In dieser Phase ist der Prozess der Kasse

passiv. Die Ankunft des nächsten Kunden beendet das Warten der Kasse, passiviert den Kundenprozess und aktiviert den Kassenprozess.

Die Prozessroutine eines Kunden ist in Abb. 2.10 dargestellt. Bei diesem Ansatz generiert die erste Anweisung jeweils den nächsten Kundenprozess. Oftmals wird das Generieren von neuen Prozessen durch eine separate Programmeinheit realisiert, die als Quelle bezeichnet wird. Eine Quelle ist in der Lage, in zufälligen, durch eine Verteilung charakterisierten Zeitabständen neue Prozesse zu generieren und sie mit Initialisierungswerten zu versehen.

2.2.2 Vergleich der ereignisorientierten und der prozessorientierten Simulation

Die prozessorientierte Vorgehensweise bietet erhebliche Vorteile bei der Modellbildung, denn alle Abläufe, die sich auf ein Objekt beziehen, werden in ihrem Zusammenhang dargestellt. Allerdings muss die Einfachheit auf der konzeptuellen Ebene mit einer entsprechend aufwändigen programmiertechnischen Verwaltung und Koordination der Prozesse, die oftmals parallel ablaufen, erkauft werden.

Die ereignisorientierte Simulationstechnik lässt dem Modellierer mehr Freiheit und gibt ihm mehr Möglichkeiten an die Hand, den Simulationsverlauf selbst zu kontrollieren. Ereignisroutinen sind meist sehr einfach, da sie nur auf den aktuellen Zeitpunkt ausgerichtet sind. Bei komplexen Prozessen wird die Vielzahl der zu berücksichtigenden Ereignisse schnell unübersichtlich und die Ereignisliste benötigt eine effiziente Verarbeitung.

2.3 Periodenorientierte Simulation

Ist in einem System nicht der Zustand nach jedem einzelnen Ereignis relevant, sondern immer erst nach Ablauf eines festgelegten Zeitintervalls, so kann mit einem periodischen Überprüfen der Systemgrößen das dynamische Verhalten des Systems untersucht werden.

2.3.1 Das Prinzip der periodenorientierten Simulation

Bei der *periodenorientierten Simulation* (auch *zeitorientierte* oder *zeitgesteuerte Simulation* genannt, engl. *time driven simulation* oder *time-slicing*) werden die Zustandsänderungen, die sich jeweils nach Ablauf von gleichgroßen Zeitintervallen Δt ergeben, ermittelt. Nach Ausführung aller Berechnungen zum Zeitpunkt t_i rückt die Simulationsuhr auf $t_{i+1} = t_i + \Delta t$ vor.

In Abb. 2.11 ist der Steuerungsalgorithmus einer periodenorientierten Simulation in Form einer einfachen Schleife dargestellt.

Die Neuberechnung des Zustands erfolgt nach vorgegebenen anwendungsspezifischen Berechnungsvorschriften, die sich an Ereignissen orientieren oder völlig

Abb. 2.11 Steuerungs-
algorithmus einer
periodenorientierten
Simulation

```
while (simulationsZeit < simulationsEnde)
{
    simulationsZeit += periodenDauer;
    „berechne den Systemzustand neu“;
}
```

unabhängig von Ereignissen sein können. Anstatt eine konkrete Zeit zu verwalten und nach jeder Periode ein Δt aufzuaddieren, wird häufig einfach ein Periodenzähler verwendet.

Da bei der periodenorientierten Simulation der Zustand eines Systems immer nur zu diskreten äquidistanten Zeitpunkten bestimmt wird, werden Zustandsänderungen zwischen den Messpunkten nicht erfasst. Um dadurch entstehende Ungenauigkeiten zu vermeiden, sollte das Zeitintervall zwischen je zwei Zustandsberechnungen möglichst klein sein. Je kürzer allerdings dieses Zeitintervall ist, desto größer wird der überflüssige Berechnungsaufwand in Phasen, in denen sich der Zustand nicht ändert.

2.3.2 Periodenorientierte Simulation einer Lagerhaltung

Eine typische Anwendung, bei der die periodenorientierte Vorgehensweise besonders sinnvoll ist, ist ein einfaches Lagerhaltungssystem. Da es sehr aufwändig ist, jede einzelne Lagerentnahme nachzubilden, begnügt man sich oftmals mit der Überprüfung des Lagerbestands nach festen Zeitperioden, z. B. immer am Abend oder nach Ablauf einer Woche. Es wird dann entschieden, ob das Lager aufgefüllt werden soll oder nicht.

Um eine solche Lagerhaltung zu simulieren, müssen durch statistische Erhebungen, etwa durch Berücksichtigung von Werten aus der Vergangenheit, Aussagen über die Veränderungen während eines zugrundeliegenden Zeitintervalls gewonnen werden.

Wir wollen als Beispiel ein Geschäft betrachten, das den aktuellen Bestseller eines bestimmten Autors verkauft. Im Lager seien zu Beginn 4 Bücher. Wenn nach einer Woche 2 oder weniger Bücher übrig geblieben sind, wird das Lager übers Wochenende aufgefüllt, so dass ab Montag wieder 4 Bücher zur Verfügung stehen. Der maximale Lagerbestand heißt *Richtbestand*, die Grenze, ab der aufgefüllt wird, wird *Meldebestand* genannt (vgl. Kap. 6.5). Werden bei leerem Lager Bücher nachgefragt, so gehen diese Kunden verloren.

In Abb. 2.12 ist ein Programm für eine periodenorientierte Simulation des Lager-Beispiels dargestellt. Es besteht aus einem Initialisierungsteil und einer Schleife, die solange periodisch die Zeit hochsetzt und den Zustand neu berechnet, bis die vorgegebene Anzahl von Perioden erreicht ist.

Für die Auswertung der Simulation müssten noch geeignete Variablen definiert und entsprechende Berechnungen eingefügt werden. Als Ergebnis der Simulation könnte man eine Schätzung für die folgenden Größen erhalten:

```
int anzahlPerioden = 0;
int simulationsDauer = 100;
int richtBestand = 4;
int meldeBestand = 2;
int lagerBestand = richtBestand;
int nachfrage = 0;

while (anzahlPerioden < simulationsDauer)
{
    anzahlPerioden++;
    „ermittle neuen Wert für nachfrage“;
    lagerBestand -= nachfrage;
    if (lagerBestand <= meldeBestand)
        lagerBestand = richtBestand;
}
```

Abb. 2.12 Periodenorientierte Simulation einer einfachen Lagerhaltung

Tab. 2.3 Verteilung der Nachfragemengen

Anzahl in Stück n	0	1	2	3	4
Wahrscheinlichkeit $p(n)$	0,10	0,20	0,30	0,30	0,10

Tab. 2.4 Zufallszahlenstrom für Nachfragemengen

<i>Nachfrage</i>	3	1	2	1	4	0	2	3	3	2	...
------------------	---	---	---	---	---	---	---	---	---	---	-----

Tab. 2.5 Tabellarische Darstellung der periodenorientierten Simulation einer einfachen Lagerhaltung

Lagerbestand zu Beginn der Woche	4	4	3	4	3	4	4	4	4	4
Nachfrage	3	1	2	1	4	0	2	3	3	2
Lagerbestand am Ende der Woche	1	3	1	3	−1	4	2	1	1	2
Auffüllmenge	3	0	3	0	4	0	2	3	3	2
<i>Woche</i>	1	2	3	4	5	6	7	8	9	10

- Anzahl, wie oft das Lager am Wochenende aufgefüllt werden muss,
- durchschnittliche Auffüllmenge,
- Anzahl der verloren gegangenen Kunden.

Der Ablauf der Simulation kann in Form einer Tabelle veranschaulicht werden, indem man pro Woche den Ausgangslagerbestand, die Nachfragemenge und den resultierenden Lagerbestand spaltenweise protokolliert.

Für die Nachfragemenge pro Woche soll die Verteilung der Tab. 2.3 zugrundegelegt werden.

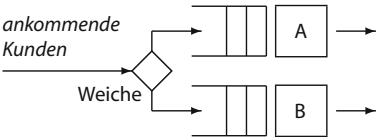
Im Programm wird die Ermittlung der Nachfragemengen als Pseudocode formuliert. Wie man zufällige Größen, die eine vorgegebene Wahrscheinlichkeitsverteilung aufweisen, mit Hilfe eines Zufallszahlengenerators erzeugt, wird später in Kap. 3 beschrieben. Deshalb wollen wir in Tab. 2.4 eine Folge von Nachfragemengen vorgeben, die die Verteilung für die Nachfragemengen widerspiegelt.

Tabelle 2.5 zeigt den Verlauf des Lagerbestands für 10 Wochen. Ein negativer Lagerbestand drückt aus, wie viele Kunden die nachgefragte Ware nicht bekommen und abgewiesen werden. Wir nehmen dabei vereinfachend an, dass jeder Kunde nur ein Buch kaufen möchte. Die Simulation liefert als weitere Information die Auffüllmenge jeweils am Ende einer Woche.

Bei diesem Verlauf musste das Lager 7 Mal am Wochenende aufgefüllt werden (die Woche 10 wird dabei mitgezählt), die durchschnittliche Auffüllmenge betrug $20/7 \approx 2,85$, und es musste in der Woche 5 ein Kunde abgewiesen werden, da nicht genügend Bücher im Lager waren.

2.4 Übungsaufgaben

1. **Ereignisorientierte Simulation eines Bankschalters**
- Zwei Bankschalter haben je eine eigene Warteschlange. Die ankommenden Kunden werden an einer Weiche immer zu demjenigen Schalter geleitet, an dem sich weniger Kunden befinden. Nach der Weiche soll das Einreihen in die Warteschlange bzw. eine eventuelle sofortige Bedienung ohne Zeitverzug erfolgen.
- Befinden sich an beiden Schaltern gleichviele Kunden, so werden sie immer zum Schalter A geschickt. Wenn sich an einem Schalter kein Kunde mehr befindet, und am anderen Schalter ein Kunde wartet, so wird dieser Kunde zum freien Schalter geholt und dort sofort bedient.



Beschreiben Sie mit Hilfe einer Tabelle eine ereignisorientierte Simulation des Systems, die den Ablauf 20 Minuten lang durchspielt. Vervollständigen Sie dazu die untenstehende Tabelle.

Die Initialisierung des Systems ist durch die erste Spalte vorgegeben.

Ankunft Kunde	2
Bedienungsende A	5
Bedienungsende B	–
Länge Warteschlange vor A	0
Zustand A	<i>a</i>
Länge Warteschlange vor B	0
Zustand B	<i>f</i>
SimZeit	0 2 ...

Die möglichen Zustände der Bedienstationen sind *a* (aktiv) und *f* (frei).

Die Zufallszahlen, die für die Simulation benötigt werden, entnehmen Sie bitte der Reihe nach der untenstehenden Tabelle. Dabei bedeuten (Zeitangaben jeweils in Minuten):

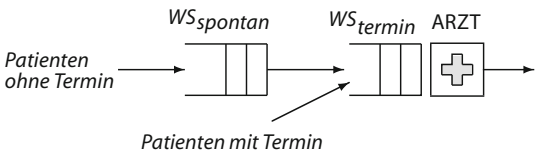
- K : Ankunftsabstand Kunden
- B_1 : Bedienzeit Schalter A
- B_2 : Bedienzeit Schalter B

K	1	5	2	2	1	7	3	5	4	9	1
B_1	8	2	3	2	7	1	4	7	1	8	5
B_2	2	3	9	5	1	3	6	8	7	5	2

Beide Warteschlangen sind unbeschränkt und werden nach der FIFO-Strategie abgearbeitet. Treten Ereignisse im gleichen Zeitpunkt ein, können Sie eine beliebige Reihenfolge wählen.

2. Ereignisorientierte Simulation einer Arztpraxis

Ein Arzt behandelt Patienten der Reihe nach. Es gibt eine Warteschlange WS_{termin} für Patienten mit einem vereinbarten Termin und eine Warteschlange WS_{spontan} für Patienten, die ohne Termin spontan gekommen sind. Patienten mit Termin werden bevorzugt behandelt.



Beide Warteschlangen sind unbeschränkt. Wenn WS_{termin} leer wird, so darf *ein* Patient ohne Termin aus WS_{spontan} in die Warteschlange WS_{termin} vorrücken, sofern WS_{spontan} nicht leer ist. Er kommt dann als nächster dran. Der Übergang der Patienten aus der Warteschlange WS_{spontan} nach WS_{termin} soll ohne Zeitverzug stattfinden.

Wenn im Spezialfall beide Warteschlangen leer sind, dann reiht sich ein ankommender spontaner Patient in die WS_{termin} ein.

Beschreiben Sie mit Hilfe einer Tabelle eine ereignisorientierte Simulation des Systems, die den Ablauf 7 Stunden lang durchspielt. Vervollständigen Sie dazu die untenstehende Tabelle. Die Initialisierung des Systems ist durch die erste Spalte vorgegeben.

Ankunft spontaner Patient	0,8		
Ankunft Patient mit Termin	0,5		
Behandlungsende	–		
Länge Warteschlange WS_{spontan}	0		
Länge Warteschlange WS_{termin}	0		
Zustand Arzt	f		
SimZeit	0,0	0,5	...

Die möglichen Zustände des Arztes sind a (aktiv) und f (frei).

Die Zufallszahlen, die für die Simulation benötigt werden, entnehmen Sie bitte der Reihe nach der untenstehenden Tabelle. Dabei bedeuten (Zeitangaben jeweils in Stunden):

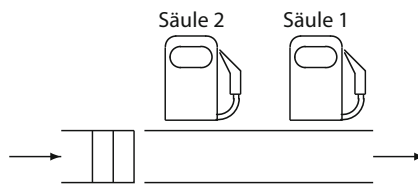
A_{spontan} : Ankunftsabstand spontane Patienten
 A_{termin} : Ankunftsabstand Patienten mit Termin
 B : Behandlungsdauer

A_{spontan}	2,3	0,8	0,2	0,1	0,5	1,4	1,5	2,3	1,8	0,7	1,0
A_{termin}	1,8	1,6	1,0	0,8	1,2	1,5	0,4	0,7	1,1	0,8	0,5
B	1,2	0,2	0,4	0,5	1,0	0,3	0,6	0,8	0,7	0,5	0,2

Treten Ereignisse im gleichen Zeitpunkt ein, können Sie eine beliebige Reihenfolge wählen.

3. Ereignisorientierte Simulation einer Tankstelle

Eine Tankstelle hat mehrere Fahrspuren mit jeweils zwei seriell angeordneten Zapfsäulen. Es soll eine solche Fahrspur mit den Zapfsäulen 1 und 2 simuliert werden.



Sind beide Zapfsäulen frei, so fährt ein ankommender Kunde zur Zapfsäule 1. Kommt ein weiteres Fahrzeug an der Fahrspur an, so tankt es an Säule 2. Sind beide Säulen belegt, so wartet der nächste Kunde in der Warteschlange. Enthält die Warteschlange ein Fahrzeug, dann fahren ankommende Kunden zu einer anderen Fahrspur.

Sind beide Säulen besetzt und der Kunde an Säule 2 wird zuerst fertig, so kann er den Tankbereich rückwärts verlassen, falls die Warteschlange leer ist. Befindet sich aber ein Fahrzeug in der Warteschlange, so muss er warten, bis auch der Tankvorgang an Säule 1 beendet ist (Säule 2 blockiert). Wenn Säule 1 frei wird, während Säule 2 noch besetzt ist, kann an Säule 1 nicht getankt werden, da es für andere Fahrzeuge keine Möglichkeit gibt, an diese Säule zu gelangen (Säule 1 blockiert).

Beschreiben Sie mit Hilfe einer Tabelle eine ereignisorientierte Simulation des Systems, die den Ablauf 10 Minuten lang durchspielt. Vervollständigen Sie dazu die untenstehende Tabelle. Die Initialisierung des Systems ist durch die erste Spalte vorgegeben.

Ankunft Fahrzeug	1,9
Bedienungsende Säule 1	0,5
Bedienungsende Säule 2	1,0
Zustand Säule 1	<i>a</i>
Zustand Säule 2	<i>a</i>
Anzahl Fahrzeuge in Warteschl.	1
SimZeit	0 0,5 ...

Die Zustände für die Säulen sind *a* (aktiv), *f* (frei) und *b* (blockiert).

Die Zufallszahlen, die für die Simulation benötigt werden, entnehmen Sie bitte der Reihe nach der untenstehenden Tabelle. Dabei bedeuten (Zeitangaben jeweils in Minuten):

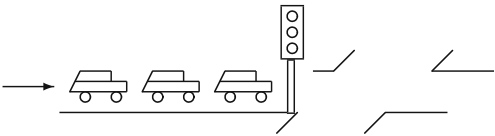
A: Zeitabstand zwischen Ankünften der Fahrzeuge an der Fahrspur
B: Dauer des Tankvorgangs einschließlich Bezahlen

A	0,7	0,4	0,6	1,0	1,2	3,6	0,6	1,3	1,9	1,0	0,5	2,1
B	4,0	2,1	4,3	3,4	1,6	3,7	2,6	3,2	6,0	4,1	2,5	3,6

Für die kurzen Fahrdauern auf der Fahrspur sollen keine Zeitdauern veranschlagt werden. Treten Ereignisse im gleichen Zeitpunkt ein, können Sie eine beliebige Reihenfolge wählen.

4. Ereignisorientierte Simulation der Warteschlange an einer Ampel

Die Warteschlange an einer Verkehrsampel soll ereignisorientiert in Form einer Tabelle simuliert werden. Fahrzeuge kommen als Ankunftsstrom an mit Zwischenankunftszeiten *A*, die aus der unten angegebenen Liste der Reihe nach entnommen werden sollen. Die Schaltreihenfolge der Ampel sei wie üblich Rot, Gelb, Grün, Gelb, Rot, usw. Es sollen drei Phasenlängen berücksichtigt werden: Rot 40 Sekunden, Gelb jeweils 5 Sekunden, Grün 30 Sekunden.



Schaltet die Ampel von Gelb auf Grün, setzen sich die wartenden Fahrzeuge in Bewegung. Die Zeitdauer, die die wartenden Fahrzeuge zum Reagieren benötigen, soll durch die Größe *R* modelliert werden (siehe Liste). Das erste Fahrzeug reagiert auf das Umschalten auf Grün, die weiteren Fahrzeuge reagieren auf das Starten des Vorderfahrzeugs.

Ab dem Zeitpunkt des Umschaltens von Grün auf Gelb startet kein weiteres Fahrzeug mehr, schon gestartete Fahrzeuge fahren aber immer weiter, und neu ankommende Fahrzeuge halten an. Ein ankommendes Fahrzeug reiht sich generell in die Warteschlange ein, es sei denn, sie ist leer und die Ampel hat Grün. Dann fährt das Fahrzeug ohne Verzögerung weiter.

Die folgende Tabelle enthält Zufallszahlen für die Ankunftsabstände A und für die Reaktionszeiten R (Zeitangaben jeweils in Sekunden).

A	9	8	10	7	8	6	12	8	9	10	6	11	10	9	8
R	3	2	4	3	2	1	2	3	2	4	2	3	2	4	3

Führen Sie eine Tabellensimulation der Warteschlange für den Verlauf von 100 Sekunden durch und bestimmen Sie für Ihren Simulationslauf, wieviele Fahrzeuge durchschnittlich warten.

Der Zustand des Systems ist durch die Anzahl wartender Fahrzeuge sowie die Farbe der Ampel charakterisiert. Treten Ereignisse im gleichen Zeitpunkt ein, können Sie eine beliebige Reihenfolge wählen.

Initialisierung: Zu Beginn sollen sich 4 Fahrzeuge in der Warteschlange befinden und die Ampel soll gerade auf Grün umgeschaltet haben. Planen Sie die nächste Ankunft eines Fahrzeugs und das Starten des ersten Fahrzeugs der Warteschlange mit Hilfe der ersten Zahlen der Zufallszahlenlisten, sowie das nächste Umschalten der Ampel.

Literatur

- J. Banks, J.S. Carson II, B.L. Nelson, D.M. Nicol, *Discrete-Event System Simulation*, 5. Aufl. (Pearson Education, Upper Saddle River, N. J., 2009)
- B. Page, W. Kreutzer, *The Java Simulation Handbook. Simulating Discrete Event Systems with UML and Java* (Shaker Verlag, Aachen, 2005)
- R. Sedgewick, *Algorithmen*, 2. Aufl. (Addison-Wesley, München, 2002)
- B. Page, T. Lechler, S. Claassen, *Objektorientierte Simulation in Java mit dem Framework DESMO-J* (Libri, Hamburg, 2000)



<http://www.springer.com/978-3-642-34870-9>

Simulation diskreter Prozesse

Methoden und Anwendungen

Hedtstück, U.

2013, XIII, 235 S. 232 Abb., Softcover

ISBN: 978-3-642-34870-9