Simulation kontinuierlicher Prozesse in hierarchischen Produktionsplanungssystemen

Entwicklung von Werkzeugen und Experimentelle Analyse

Dissertation zur Erlangung des Grades
eines Doktors der Wirtschafts- und Sozialwissenschaften
Doctor rerum politicarum

vorgelegt von
Dipl. Wirtsch. Ing. Christian Seipl

Eingereicht am 27.10.2009

Universität der Freien und Hansestadt Hamburg
Fakultät Wirtschafts- und Sozialwissenschaften
Fachbereich Wirtschaftswissenschaften

Das wissenschaftliche Gespräch fand am 27.01.2010 statt.
Mitglieder des Prüfungsausschusses
Vorsitzender: Prof. Dr. Stefan Voß
1. Gutachter: Prof. Dr. Hartmut Stadtler
2. Gutachter: Prof. Dr. Bernd Page

Dipl. Wirtsch. Ing. Christian Seipl

Simulation kontinuierlicher Prozesse in hierarchischen Produktionsplanungssystemen

Entwicklung von Werkzeugen und Experimentelle Analyse

Für diese Arbeit wurde - mit Ausnahme von Xpress-MP - ausschließlich freie Software verwendet. Hervorzuheben sind folgende Distributionen, Pakete und Programme: openSuSE, KDE, Java, Eclipse, MySQL, R, LaTeX, Asymptote, Gnuplot, JabRef, kile sowie die Allzweckwerkzeuge Perl, sed und awk.

Diese Arbeit wurde unterstützt durch dashOptimization (jetzt FICO - Fair Isaac Corporation), welche im Rahmen des Academic-Partner-Programs die Software Xpress-MP zur Verfügung gestellt hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

APS	Advanced Planning System
bzw.	beziehungsweise
CLSP	Capacitated Lot Sizing Problem
CLSPL	Capacitated Lot Sizing Problem with Linked lot sizes
CSLP	Continuous Setup and Lotsizing Problem
d.h.	das heißt
DLSP	Discrete Lotsizing and Scheduling Problem
EOQ	Economic Order Quantity
ERP	Enterprise Resource Planning
ff.	folgende (Seiten)
GLSP	General Lotsizing and Scheduling Problem
GPL	General Public License, Lizenz zur Lizensierung freier Software, Herausgeber: Free Software Foundation
JDK	Java Development Kit, Entwicklungsumgebung für Java Programme
JVM	Java Virtual Machine, Teil der Java Laufzeitumgebung, der den Bytecode ausführt
LGPL	GNU Lesser General Public License Lizenz zur Lizensierung freier Software, speziell für Programmbibliotheken, Herausgeber: Free Software Foundation
MIP	Mixed Integer Programming

MRG32k3a	Multiple Recursive Random Number Generator - mehrfacher gekoppelter linearer Kongruenz Automat - 32 bedeutet hier, daß die Länge der Koeffizienten 32 bit beträgt, k3a ist die Klassifizierung von L'Ecuyer (1999, S. 161)
MRP	Material Requirements Planning
MRP-II	Manufacturing Resources Planning
MT19937	Mersenne Twister, Pseudozufallszahlengenerator - 19937 bezieht sich auf die Anzahl der Koeffizienten im Zustandsvektor
ODBC	Open Database Connectivity, standardisierte Datenbankschnittstelle
PLSP	Proportional Lotsizing and Scheduling Problem
SB	Sicherheitsbestand
TBO	Time Between Orders
u.	und, bei Autorenangaben
u.a.	unter anderem - bei Autorenangaben: und andere
UML	Unified Modeling Language
V	Variationskoeffizient
vgl.	vergleich(e)
VRSPTW	Vehicle Routing and Scheduling Problem with Time-Windows
z.B.	zum Beispiel

Symbolverzeichnis

$\eta_{j,k,t}$	Untere Grenze für die Anzahl an Rüstvorgängen für Produkt j im Intervall $[k,t]$
$\kappa_{m,j}$	Fertigungsgeschwindigkeit von Produkt j auf Maschine m
$\rho_{j,k,t}$	Untergrenze der Nachfrage, die im Zeitraum $[k,t]$ nicht durch die verfügbare Kapazität abgedeckt werden kann
$BL_{j,t}$	Bestand an Nachlieferungen (Backlog) für Produkt j am Ende von Periode t
C_j^{max}	Maximal verfügbare Kapazität für Produkt j während des gesamten Planungshorizonts, aggregiert über alle Maschinen abzüglich der Rüstzeiten
c_j^{BL}	Strafkosten je Periode für eine Mengeneinheit Backlog von Produkt j
c_j^{IT}	Strafkosten je Periode für eine Mengeneinheit, um die der Ziellagerbestand von Produkt j unterschritten wird
c_j^{SS}	Strafkosten je Periode für eine Mengeneinheit, um die der Sicherheitsbestand von Produkt j unterschritten wird
$c_{m,t}$	Verfügbare Kapazität von Maschine m in Periode t
$d_{j,k,t}$	Kumulierte Nachfrage für Produkt j von Periode k bis einschließlich Periode t

$d_{j,t}$	Nachfrage nach Produkt j in Periode t
h_j	Lagerkostensatz je Periode und Mengeneinheit von Produkt j
i_j^T	Ziel=Lagerbestand von Produkt j am Ende des Planungshorizonts
I_j^-	Höhe der Unterschreitung des Zielbestands von Produkt j am Ende des Planungshorizonts
$I_{j,0}$	Lageranfangsbestand von Produkt j
$I_{j,t}$	Lagerbestand für Produkt j am Ende von Periode t
J	Anzahl an Produkten
M	Anzahl an Maschinen
$sc_{m,j}$	Kosten für einen Rüstvorgang von Produkt j auf Maschine m
$ss_{j,t}$	Sicherheitsbestand von Produkt j in Periode t
$SS_{j,t}^-$	Höhe der Unterschreitung des Sicherheitsbestands von Produkt j in Periode t
$st_{m,j}$	Rüstzeit von Maschine m für Produkt j
T	Anzahl an Perioden
w_j^M	Menge der Maschinen, auf denen Produkt j produziert werden kann
w_m^J	Menge der Produkte, die auf Maschine m produziert werden kann
$X_{m,j,t}$	Produktionsmenge von Produkt j auf Maschine m in Periode t
$Y_{m,j,t}$	Indikator für einen Rüstvorgang auf Maschine m für Produkt j in Periode t
$Z_{j,t}$	Indikator für eine Rüstvorgang von Produkt j in Periode t

Kapitel 1

Einleitung

1.1 Fragestellung

Der Ursprung dieser Arbeit liegt in folgender Frage, die während der Zusammenarbeit mit einem Unternehmen entstanden ist: *Betrachtet wird eine Anlage in der Prozeßindustrie, auf der eine kontinuierliche Fertigung stattfindet. Wenn innerhalb eines Jahres während der Produktion Fehlmengen entstehen, welche Auswirkungen hat dies auf die Verfügbarkeit (den β-Servicegrad)?*

Die meisten Antworten auf diese Frage lassen sich durch eine einfache Faustregel zusammenfassen „Je höher die Fehlmenge, desto geringer der Servicegrad“. Während diese Aussage für die meisten Systeme uneingeschränkt gültig ist, wird sich im Rahmen dieser Untersuchung zeigen, daß die Auswirkungen der Fehlmengen nicht durch eine einfache Regel zu beschreiben sind. Bereits in einem einfachen mehrstufigen Planungssystem treten Effekte auf, die diese Faustregel in Frage stellen. Beispiele hierfür werden in Kapitel 8 gezeigt.

Vorangehende Arbeiten betrachten meist nur zwei Planungsebenen (Jahresplanung und Monatsplanung) und vernachlässigen die dritte Planungsebene (Feinplanung, vgl. Abschnitt 6.1 auf Seite 188). Sie gehen von der Annahme aus, daß die Fehlmengen nur im Bereich der mittelfristigen Planung entstehen und daß zu einer Fehlmenge immer

der gleiche β-Servicegrad gehört. In Kapitel 8 wird gezeigt, daß diese Annahme hier nicht zulässig ist.

Die Notwendigkeit zur Simulation der Feinplanung ergibt sich auch aus der Abbildung der in der Praxis gemessenen Kennzahlen. Der in Praxis häufig eingesetzte β-Servicegrad wird auf der Ebene der Feinplanung berechnet und nicht stichtagsbezogen am Ende eines Monats. Anders formuliert: Die Kundennachfrage trifft nicht nur am Monatsende ein. Abbildungsfehler, die bei dieser vereinfachenden Betrachtung entstehen, werden in Abschnitt 6.1.6 auf Seite 202 vorgestellt.

Die Untersuchung setzt sich aus drei Bausteinen zusammen: eine Simulationsumgebung, eine Menge an Testdaten sowie Verfahren zur Auswertung und Visualisierung der Daten. Die Simulationsumgebung stellt den ersten Forschungsbereich dieser Arbeit dar. Es existieren verschiedene Pakete - freie und kommerzielle - auf dem Markt, die kontinuierliche Prozesse simulieren können (vgl. Kapitel 4 und 5). Alle betrachteten Pakete verwenden eine interne Darstellung der Prozesse, die einer effizienten Simulation langer Zeiträume entgegen stehen. Zur Lösung dieses Problems werden Algorithmen und eine Referenzimplementierung entwickelt, die kontinuierliche Prozesse exakt simulieren und gleichzeitig die Rechnerkapazitäten gering belasten. Die einzige Einschränkung ist der Einsatz auf den Bereich der Produktionsplanung, da Annahmen getroffen werden, die aus den hier eingesetzten Modellen übernommen wurden.

Der zweite Baustein, die Testdaten, stellt einen weiteren Forschungsbereich dieser Arbeit dar. Eine standardisierte Methodik, um Testdaten zu erzeugen, sowie ein Satz an Methoden ist zum aktuellen Zeitpunkt nicht verfügbar gewesen. Aufgrund der Erfahrungen aus dieser und anderen Arbeiten, wurden verschiedene Standardmethoden entwickelt. Diese werden in einem allgemeinen Teil in Kapitel 3 vorgestellt, sowie in der Anwendung für Losgrößenmodelle in Kapitel 7 gezeigt.

Der letzte Baustein - die Auswertung und Visualisierung - stellt im Bezug auf die Software keine Probleme dar, da eine ausreichende Anzahl an kommerziellen und nicht-kommerziellen Statistikpaketen existiert. Basierend auf einer neuen Art der Darstellung wird der Begriff vom „Fingerabdruck eines Planungssystems“ entwickelt. Diese

Darstellung ermöglicht weitere Einblicke und Ansätze zur Erklärung der beobachteten Daten sowie für zukünftig zu entwickelnde Erklärungsmodelle.

1.2 Aufbau

Die vorliegende Arbeit besteht aus den im vorangehenden Abschnitt vorgestellten Bausteinen:

- Theoretische Grundlagen zur Erzeugung von Testdaten
- Entwicklung von Algorithmen zur Simulation kontinuierlicher Materialflüsse
- Anwendung auf die hierarchische Planung

Der erste Teil umfaßt die Kapitel 2 und 3. Kapitel 2 gibt einen kurzen Überblick über die verschiedenen Algorithmen zur Erzeugung von Zufallszahlen sowie ihre Implementierungen. Es stellt eine Grundlage sowohl für die Simulation als auch für die Erzeugung von Testdaten dar, da in beiden Fällen ein verläßlicher Zufallszahlengenerator die Voraussetzung für eine erfolgreiche Durchführung und statistische Auswertung ist. Der Schwerpunkt in diesem Kapitel liegt auf der Vorstellung eines Zufallszahlengenerators mit Teilströmen. Dieses Konzept wird nicht nur in der Simulation eingesetzt, sondern auch in der Erzeugung von Testdaten in den darauf folgenden Kapiteln.

Kapitel 3 gibt zu Beginn einen Literaturüberblick der bisher eingesetzten Verfahren zur Erzeugung von Testdaten. Nach einer Vorstellung von Anforderungen an Testdaten wird eine erste Klassifizierung durchgeführt. Anhand dieser Klassifizierung werden Merkmale der Testdaten und der erzeugenden Methoden abgeleitet, die Auswirkungen auf die Qualität der erzeugten Daten haben. Im weiteren Verlauf des Kapitels werden verschiedene Algorithmen vorgestellt, um Testdaten mit speziellen Eigenschaften zu erzeugen. Diese Algorithmen lassen sich nicht nur auf Probleme der Losgrößenplanung anwenden, sondern sind

allgemein für eine Vielzahl von Problemen einsetzbar. Abschließend wird die Implementierung von einigen Java-Klassen vorgestellt, welche die Erzeugung von Testdaten in Teilen standardisiert.

Der zweite Bestandteil dieser Arbeit ist die Entwicklung von Algorithmen zur effizienten Simulation kontinuierlicher Materialflüsse. Nach einer Vorstellung der auf dem Markt vorhandenen Softwarepakete in Kapitel 4 werden die Auswahlkriterien sowie die Entscheidung für das Paket DESMO-J vorgestellt. Anschließend erfolgt eine Beschreibung aller Modifikationen unter technischen Gesichtspunkten, sowie eine Vorstellung der Referenzimplementierung. Diese ist auch der Ausgangspunkt für die weiteren Entwicklungen im darauf folgenden Kapitel.

Im Zentrum des zweiten Teils steht Kapitel 5 mit der Entwicklung der Algorithmen zur Simulation kontinuierlicher Materialflüsse. Nach einer kritischen Betrachtung der klassischen Ansätze zur Simulation kontinuierlicher Prozesse werden die problemspezifischen Eigenschaften analysiert. Speziell für den Bereich der Losgrößenplanung läßt sich die Annahme der alleinigen Abhängigkeit eines Materialflusses von der Zeit ableiten. Ausgehend von dieser zentralen Annahme werden Definitionen und Algorithmen entwickelt, die eine im Vergleich zu den klassischen Ansätzen effiziente Simulation ermöglichen. Die Algorithmen wurden so allgemein gehalten, daß beliebige Erzeugnisstrukturen simuliert werden können. Auch die Funktionen zur Beschreibung der Materialflüsse können unter der Einschränkung, daß die Flußrichtung erhalten bleibt, beliebig gewählt werden. Im letzten Abschnitt des Kapitels wird die Implementierung anhand der Referenzimplementierung in Java vorgestellt.

Der letzte Teil dieser Arbeit enthält den Aufbau und die Durchführung einer Simulationsstudie im Bereich der hierarchischen Produktionsplanung. Untersuchungsgegenstand ist eine einstufige Erzeugnisstruktur, deren Produktionspläne von einem dreistufigen Planungssystem erzeugt werden. Die dabei eingesetzten Modelle und Planungsalgorithmen orientieren sich an den Verfahren, die in einem Advanced Planning System (APS) eingesetzt werden. Die Simulationsstudie bildet somit auch einen wesentlichen Bestandteil eines APS nach, wie er in der Prozeßindustrie einsetzbar ist.

Daran anschließend wird in Kapitel 7 ein für diese Problemstellung entwickelter Testdatengenerator beschrieben. Dabei werden die in Kapitel 3 entwickelten Konzepte und Methoden eingesetzt. Im letzten Kapitel erfolgt eine statistische Auswertung der Daten, sowie die Einführung einer neuen Darstellung der Daten: der „Fingerabdruck eines Planungssystems“. Dieser stellt den Zusammenhang zwischen Ein- und Ausgabegrößen des Planungssystems dar und ermöglicht eine graphische Erklärung der Auswirkungen von Änderungen an den Eingabegrößen. Die Arbeit schließt mit einem Rückblick auf die erreichten Ziele, sowie einem Ausblick auf zukünftige Entwicklungen.

Teil I

Grundlagen

Kapitel 2

Zufallszahlengenerator

Das folgende Kapitel soll einen kurzen Überblick über die Theorie und die Implementierung von Zufallszahlengeneratoren geben. Im ersten Abschnitt werden die Anforderungen vorgestellt, wobei sowohl die theoretischen Aspekte, als auch die praktischen Aspekte bei der Implementierung berücksichtigt werden. Im zweiten Abschnitt wird ein kurzer Überblick der akutell in der Forschung behandelten Typen von Zufallszahlengeneratoren gegeben. Der letzte Abschnitt des Kapitels stellt eine Referenzimplementierung vor, die in dieser Arbeit sowohl in der Simulation, als auch im Testdatengenerator eingesetzt wird.

2.1 Anforderungen

Der Zufallszahlengenerator ist eines der zentralen Elemente, sowohl für die Simulation, als auch die Generierung von Testdaten. Die Zusammenstellung der folgenden Anforderungen orientiert sich an Pidd (2004, S. 182), Banks u. a. (2000, S. 253), sowie Law u. Kelton (2000, S. 405) und läßt sich unter folgenden Stichworten zusammenfassen:

- Reproduzierbarkeit
- Qualität der Zufallszahlen

- geringe Belastung der Ressourcen
- Unterteilung eines Zahlenstroms in unabhängige Einzelströme

Reproduzierbarkeit

Die zentrale Anforderung an einen Zufallszahlengenerator ist die Reproduzierbarkeit der erhaltenen Zufallszahlen. Diese ist zwingend notwendig, um Simulationen beliebig oft identisch zu wiederholen (z. B. zur Verifizierung oder Fehlersuche). Der Zufallszahlengenerator muß somit einen inneren Zustand besitzen, der von einem Algorithmus in deterministischer Weise verändert wird, und basierend auf dem inneren Zustand eine Zufallszahl erzeugt. Die Reproduzierbarkeit eines Experimentes erfolgt durch die Wiederherstellung des inneren Zustandes. Durch den deterministischen Ablauf führt jede Wiederholung des Experimentes ausgehend von diesem Zustand zu den gleichen Ergebnissen.

Ausgehend von den Anforderungen scheiden somit Zufallszahlengeneratoren aus, die auf spezieller Hardware basieren, da für diese kein innerer Zustand definierbar oder reproduzierbar ist. Die Anwendungsgebiete von hardwarebasierten Zufallszahlengeneratoren liegen dementsprechend auch im Bereich der Kryptographie bei der Erzeugung von One-Time-Pads (Buchmann 2008, S. 100). Somit bleibt nur noch die Klasse der Pseudo-Zufallszahlengeneratoren, welche als Einzige die Reproduzierbarkeit erfüllt. Ein Pseudo-Zufallszahlengenerator ist ein Algorithmus, der Zahlen erzeugt, welche die statistischen Kriterien einer zufälligen Verteilung erfüllen. Durch Kenntnis der Eingabedaten läßt sich die gleiche Folge an Ausgabewerten beliebig oft reproduzieren, der Algorithmus ist deterministisch. Wenn im Folgenden von Zufallszahlengeneratoren gesprochen wird, bezieht sich dies immer auf Pseudo-Zufallszahlengeneratoren. Auch der Begriff der Zufallszahl bezieht sich im Folgenden immer auf die Zahl, die von einem Pseudo-Zufallszahlengenerator erzeugt wurde.

Durch das deterministische Verhalten besitzt der Zufallszahlengenerator weitere Vorteile in den Programmier- und Entwicklungsphasen einer Simulation. Während der Phase des Debuggens können Fehler, die

während der Programmierung auftreten, leichter identifiziert werden, da der Zustand, in dem ein Fehler aufgetreten ist, reproduzierbar ist. Somit wird einem nur schwer zu beherrschenden Verhalten, in dem Fehler ohne sichtbare Zusammenhänge auftreten, vorgebeugt (Law u. Kelton 2000, S. 405).

Qualität der Zufallszahlen

Unter der Qualität der Zufallszahlen sind mehrere Aspekte vereint. Ein Standard-Zufallszahlengenerator soll Zahlen erzeugen, deren Verteilung einer $[0;1]$ Gleichverteilung entsprechen. Somit betrifft die Qualität sowohl den Aspekt der statistischen Unabhängigkeit der Zahlen untereinander, als auch die Anpassung an eine Gleichverteilung. Zur Überprüfung der Anpassung an eine Gleichverteilung werden statistische Tests eingesetzt, wie sie z. B. in Knuth (1997, S. 38) vorgestellt werden.

Ein weiteres wichtiges Kriterium für die Qualität des Zufallszahlengenerators ist die sogenannte Periodenlänge (L'Ecuyer 2006, S. 56). Diese definiert für einen Pseudo-Zufallszahlengenerator die Anzahl der Iterationen, bevor sich ein innerer Zustand wiederholt und somit die gleichen Zufallszahlen erzeugt werden. Ein guter Generator besitzt eine möglichst große Periodenlänge.

Kriterien zur Erzeugung einer großen Periodenlänge hängen vom verwendeten Algorithmus ab und sind Gegenstand umfangreicher Forschungen. Einen guten Überblick über verschiedene Verfahren, Meßmethoden sowie Probleme bei bestehenden Generatoren wird von Hellekalek (1998) gegeben. Entacher (1998) zeigt, daß auch bei in der Literatur weitverbreiteten Testdatengeneratoren durch eine ungünstige Wahl der Parameter Zahlen erzeugt werden, die sehr regelmäßige Sequenzen enthalten. Eine entsprechende Parametrisierung ist somit eine wichtige Voraussetzung für eine gute Qualität.

Unabhängigkeit von der Hardwareplattform

Die Unabhängigkeit von der verwendeten Hardware ist eine konkretisierte Anforderung an die Reproduzierbarkeit. Aufgrund der Vielzahl an heute verwendeter Hardware ist es nötig, einen Zufallszahlengenerator auszuwählen, dessen Architektur unabhängig von der Hardware ist. Die Implementierung muß somit Datentypen und Operationen benutzen, die auf allen Plattformen zu identischen Ergebnissen führen. Der der Einsatz von Fließkommazahlen ist damit ausgeschlossen, da diese bei arithmetischen Operationen Rundungsoperationen durchführen (IEEE 1985, S. 6). Somit stehen nur noch Operationen aus dem Bereich der Ganzzahlarithmetik und logische Operatoren zur Verfügung.

Unabhängigkeit vom verwendeten Betriebssystem

Ein weiteres wichtiges Kriterium für den praktischen Einsatz ist die Unabhängigkeit vom verwendeten Betriebssystem - sowohl im Bezug auf die Art des Betriebssystems, als auch die Version des Betriebssystems. Dies stellt im Wesentlichen eine Einschränkung auf die verwendeten Bibliotheken dar. Die meisten Betriebssysteme stellen Funktionen zur Verfügung, um gleichverteilte Zufallszahlen zu erzeugen. Da diese Funktionen jedoch meist nur unzureichend dokumentiert sind oder eine zu geringe Periodenlänge besitzen, sind sie für den Einsatz in einer Simulation ungeeignet.

Geringe Belastung der Ressourcen

Im Rahmen einer Simulation werden an mehreren Stellen Zufallszahlen in zum Teil großen Mengen benötigt. Damit der Zufallszahlengenerator nicht zu einem Engpaß wird, ist eine effizienter Algorithmus notwendig. Sowohl die Rechenzeit, als auch der Speicherverbrauch sollten gering im Vergleich zur Simulation sein.

Unterteilung in unabhängige Einzelströme

Jeder Pseudo-Zufallszahlengenerator hat eine definierte maximale Periodenlänge. Der dazugehörige Strom von Zufallszahlen entspricht dem Hauptstrom. Wenn ein Zufallszahlengenerator es aufgrund des Algorithmus zuläßt, daß einzelne Stellen des Stroms gezielt angesprungen werden können, so ist es möglich, den Hauptstrom in definierte Teilströme zu unterteilen. Die Länge eines Teilstroms muß so groß sein, daß dieser allein betrachtet auch die statistischen Tests auf Unabhängigkeit besteht. Weiterhin ist es notwendig, daß die Teilströme untereinander keine Korrelation aufweisen.

Die Zahlenströme können dazu benutzt werden, um den stochastischen Prozessen innerhalb einer Simulation unabhängige Teilströme zuzuweisen (Law u. Kelton 2000, S. 405). Dies ermöglicht den Einsatz von Techniken zur Reduktion der Varianz (Law u. Kelton 2000, S. 581). Zusammen mit der Synchronisation der Teilströme sind diese Ansätze in der Literatur unter dem Namen „common random number streams“ bekannt (Law u. Kelton 2000, S. 582, Pidd 2004, S. 222, Kelton 2006). Die Voraussetzungen für den Einsatz dieser Techniken werden in Law u. Kelton (2000, S. 582) beschrieben.

Ein weiterer Vorteil ergibt sich in der Implementierung. Die Interaktion von Komponenten wird auf im Modell vorgegebenen Möglichkeiten beschränkt. Eine indirekte Beeinflussung durch die Verwendung von nur einem Zufallszahlenstrom wird ausgeschlossen. Programmierfehler, die in einer der Komponenten auftreten, können andere Komponenten nur noch so beeinflussen, wie es vom Modell vorgesehen ist. Die Isolation und Reproduktion von Fehlern wird somit erleichtert.

2.2 Aktuelle Generatoren

Auf Basis der vorangehenden Kriterien kann die Auswahl an möglichen Generatoren bereits eingeschränkt werden. In der aktuellen Forschung lassen sich mehrere Typen identifizieren, die aufgrund ihrer Konstruktion die genannten Kriterien erfüllen. Der folgende Überblick beschränkt

sich auf die beiden größten Gruppen der Linearen-Kongruenz-Automaten und der davon abgeleiteten $\mathbb{F}_2$ Automaten. Weitere Gruppen wie z. B. nicht-lineare Generatoren werden in L'Ecuyer (2006, S. 75) vorgestellt.

Lineare-Kongruenz-Automaten

Die Klasse der Linearen-Kongruenz-Automaten umfaßt sowohl die einfachen, als auch die kombinierten (mehrfachen) rekursiven Automaten. Die einfachen Automaten werden ausführlich in Knuth (1997, S. 9) behandelt. Knuth (1997) zeigt auch, welche Parametereinstellungen zu einer möglichst großen Periodenlänge führen. Verfahren zur Unterteilung des Hauptstroms in Teilströme werden in L'Ecuyer (2006, S. 67) sowie in L'Ecuyer u. Côté (1991, S. 101) vorgestellt.

Die Periodenlänge eines einfachen Automaten wird durch den Modulo-Parameter begrenzt, so daß bei einer n-Bit Arithmetik kein Automat eine größere Periodenlänge als 2^n besitzen kann (Knuth 1997, S. 11). Da in den meisten Programmiersprachen für ganze Zahlen Längen von 32 bit bzw. 64 bit vorgesehen sind, ist die Periodenlänge auf 2^{32} bzw. 2^{64} beschränkt. Zur Verbesserung der Periodenlänge und der Qualität des Generators wurde die Klasse der kombinierten oder mehrfachen rekursiven linearen Kongruenz Automaten entwickelt (L'Ecuyer u. a. 2002b). Ein Beispiel für einen solchen Generator gibt L'Ecuyer (1988, S. 742) zusammen mit Kriterien für die Bestimmung der Parameter, unter denen der Generator große Periodenlängen erzeugt. Eine gute Einführung in die Thematik findet sich in L'Ecuyer (1994). Automaten, die ausführlich getestet wurden und eine sehr große Periodenlänge bieten, finden sich bei L'Ecuyer (1999) oder bei L'Ecuyer u. Andres (1997).

Modulo 2 oder $\mathbb{F}_2$ Automaten

Eine Unterklasse der Linearen-Kongruenz-Automaten basiert auf der binären Darstellung von Zahlen in einem Computer. Der Modulo entspricht somit dem Wert 2, so daß ein Raum $\mathbb{F}_2$ entsteht, der nur zwei

Elemente $\{0, 1\}$ besitzt. Die Vorteile liegen in der Verwendung der logischen Operationen AND und XOR, welche innerhalb von $\mathbb{F}_2$ einer Multiplikation bzw. Addition entsprechen (Knuth 1997, S. 401).

Die Darstellung erfolgt über Bit-Vektoren der Länge k. Ist k größer als ein Register des Prozessors (mit 32 oder 64 bit), so wird der Vektor aus einem Feld zusammengesetzt. Der Speicherbedarf für die Repräsentation des inneren Zustands steigt gleichzeitig an.

Ein aktueller Generator wird von Matsumoto u. Nishimura (1998) vorgestellt, der sog. Mersenne Twister oder kurz MT19937. Dieser besitzt einen inneren Zustand, der sich aus 624 32 bit Vektoren zusammensetzt. Er besteht alle statistischen Tests und besitzt eine Periodenlänge von $2^{19937} - 1$. Da alle Rechenoperationen auf logische Operationen zurückzuführen sind, welche von Prozessoren meist innerhalb von einem Taktschritt berechnet werden können, erreicht der Generator eine hohe Geschwindigkeit. Problematisch ist hier der Speicherbedarf - für eine Instanz ist bereits $\approx$1 Kilobyte Speicher nötig. Für Simulationen mit mehreren Tausend Instanzen ist er nur bedingt geeignet.

2.3 Der Zufallszahlengenerator von L'Ecuyer

Aus den im vorangehenden Abschnitt vorgestellten Gruppen sind zwei Generatoren für Simulationen besonders geeignet: der MRG32k3a Generator, der in L'Ecuyer (1999, S. 163) vorgestellt wird, sowie der von Matsumoto u. Nishimura (1998) entwickelte MT19937. Beide erfüllen die oben aufgeführten Kriterien und besitzen eine Referenzimplementation, die von den Entwicklern zur Verfügung gestellt wird. Der Mersenne Twister hat den Vorteil einer sehr hohen Periodenlänge sowie einer höheren Geschwindigkeit, als der MRG32k3a. Der Vorteil der hohen Geschwindigkeit kann bei der Berechnung von Sprungdistanzen, die zur Unterteilung in Teilströme nötig sind, nicht beibehalten werden. Der Speicherbedarf zur Berechnung der Sprungdistanzen ist ohne weitere Modifikationen sehr hoch (Haramoto u. a. 2008, S. 386). Auch die Weiterentwicklung von Haramoto u. a. (2008) hat noch einen höheren Speicherbedarf, als die Implementierung des MRG32k3a. Bereits im

Normalbetrieb ist der Speicherbedarf des Mersenne Twister wesentlich höher, als der des MRG32k3a (Matsumoto u. Nishimura 1998, S. 8). Die Wahl fiel daher auf den Generator von L'Ecuyer, da dieser einen guten Kompromiß zwischen Geschwindigkeit und Speicherbedarf darstellt, bei einer ausreichend hohen Periodenlänge. Eine Referenzimplementierung wird in L'Ecuyer u. a. (2002b) vorstellt, welche bereits die Unterteilung in Teilströme ermöglicht. Die Quellen hierfür sind auf den offiziellen Seiten von L'Ecuyer (2009) in verschiedenen gängigen Programmiersprachen erhältlich.

Der interne Aufbau des MRG32k3a besteht aus Feldern mit einer maximalen Auflösung von 32 bit. Da nach jeder Iteration die neue Zufallszahl aus den 32 bit Werten abgeleitet wird, hat auch der hier erzeugte Wert eine maximale Auflösung von 32 bit. Für die Erzeugung einer Zufallszahl vom Typ *double* werden jedoch 53 bit für die Mantisse benötigt, so daß es zwei Alternativen gibt: nur eine Ziehung mit geringerer Auflösung oder zwei Ziehungen hintereinander, die kombiniert werden. Beide Alternativen sind in der Referenzimplementation vorhanden und können genutzt werden. Durch die Wahl von anderen Koeffizienten kann die Genauigkeit von 32 bit auf 63 bit erhöht werden. Die im Rahmen dieser Arbeit erstellte Implementierung kann durch den Austausch der Koeffizienten und Matrizen leicht auf eine 63 bit Genauigkeit umgestellt werden. Entsprechende Koeffizienten, die eine hohe Qualität aufweisen, werden von L'Ecuyer (1999, S. 160ff) vorgestellt.

Um den MRG32k3a herum wurde eine Java-Klasse erzeugt, welche den Generator im Hintergrund benutzt und "virtuelle" Generatoren mit Teilströmen zur Verfügung stellt. Die Methoden zur Berechnung der Sprungdistanzen sind eine Weiterentwicklung der Arbeiten von L'Ecuyer u. Côté (1991). Abweichend hiervon erfolgte die neue Implementierung in Java.

Aufspaltung in Teilströme

Der Hauptstrom des Generators entspricht einer Zahlenfolge mit einer Länge von $\approx 2^{191}$ (L'Ecuyer 1999, S. 1074). Dieser Hauptstrom wird unterteilt in mehrere Ebenen mit Teilströmen, deren Länge entspre-

chend geringer ist. Eine graphische Darstellung wird in Abbildung 2.1 auf der nächsten Seite gezeigt. Die Länge der Teilströme wurde von L'Ecuyer so festgelegt, daß die statistischen Tests zur Unabhängigkeit und Gleichförmigkeit einen möglichst hohen Wert erreichen.

Zu beachten ist die minimale Abweichung der Periodenlänge von 2^{191}. Diese wird von L'Ecuyer mit $((2^{32}-209)^3-1)\cdot((2^{32}-22853)^3-1)/2$ angegeben, was nur ungefähr 2^{191} entspricht. Dies bedeutet aber auch, daß nicht exakt 2^{64} Teilströme der Länge 2^{127} erzeugt werden, sondern nur $\approx 2^{64}-2^{48}$. Für die Navigation innerhalb der Teilströme hat es keine Auswirkungen, wenn der Generator nur in eine Richtung bewegt wird. Sobald jedoch von einer zyklischen Struktur ausgegangen wird, welche durch die Teilströme exakt aufgeteilt wird, entsteht ein systematischer Fehler. Dieser beruht auf der Länge des Hauptstroms. Es ist nicht möglich, innerhalb des Stroms um n Teilströme der Ebene 1 voranzuschreiten und einen identischen Zustand zu erreichen, wie bei einem Voranschreiten in entgegengesetzter Richtung um $2^{64}-n$ Teilströme.

Zur Bewegung innerhalb des Hauptstroms und der Teilströme wird eine alternative Darstellung des Zustandes in einem Vektor benutzt. Alle Bewegungen lassen sich so als Multiplikation des Zustandsvektors mit einer Matrix darstellen. Durch die Definition der Rechenvorschrift für das Ziehen einer Zufallszahl ist gleichzeitig die Richtung, in die sich der Hauptstrom bewegt, festgelegt. Die dazugehörige Matrix (im folgenden Ursprungsmatrix genannt) ist in L'Ecuyer (2006, S. 67) beschrieben. Das Voranschreiten um einen Teilstrom entspricht somit dem wiederholten Ziehen der nächsten Zufallszahl. Die Anzahl der Wiederholungen wird durch die Länge des Teilstroms bestimmt, wie er in der Abbildung auf der folgenden Seite definiert ist. Die n-fache Anwendung der Matrix auf den Zustandsvektor kann auch durch das Multiplizieren des Vektors mit einer Sprungmatrix erfolgen. Diese neue Matrix besteht aus der Ursprungsmatrix, welche mit n potenziert wurde. Da die Längen aller Teilströme bereits vorher bekannt sind bzw. bei der Konstruktion des Generators definiert wurden, können die Matrizen für alle Teilströme im Voraus berechnet werden. Zum Einsatz kommt ein "divide-and-conquer" Verfahren, daß die binäre Repräsentation des Exponenten ausnutzt,

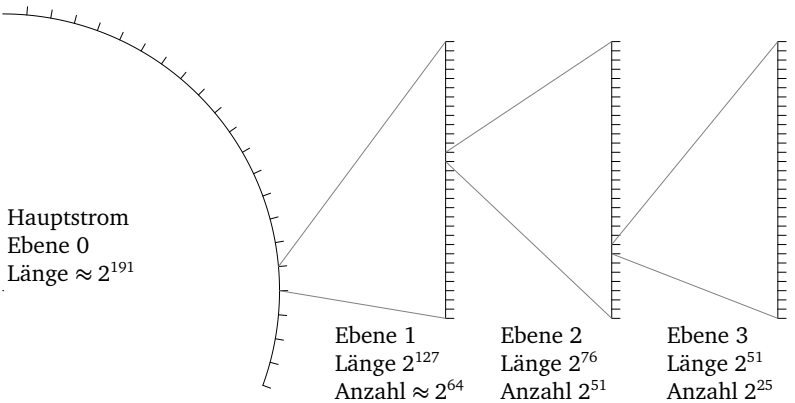


Abbildung 2.1: Aufbau des Zufallszahlengenerators von L'Ecuyer

um die Anzahl der Operationen zu verringern. Beschrieben wird dieses Verfahren in Knuth (1997, Kap. 4.6.3, S. 442).

Implementierung

Die Schnittstelle der von L'Ecuyer u. a. in Java geschriebene Referenzimplementierung enthält einige Inkonsistenzen. Diese betreffen z. B. die Bewegung innerhalb der Teilströme. Die Schnittstelle erlaubt nur Sprünge mit einer Längenangabe, die von einer *int* Variablen dargestellt werden kann. Da ein *int* in Java auf maximalen Wert von $2^{31} - 1$ beschränkt ist, können keine Sprünge in Teilströme mit einem größeren Index durchgeführt werden. Durch eine vollständige Neuimplementierung wurden diese Restriktionen aufgehoben (vgl. Abbildung 2.2 auf Seite 20). Die Schnittstellen wurden in den meisten Fällen beibehalten und erweitert, sowie korrigiert. Die allgemeine Schnittstelle für Zufallszahlengeneratoren *UniformRandomGenerator* wurde von DESMO-J übernommen und erweitert (DESMO-J wird in Kapitel 4 auf Seite 75 vorgestellt).

Die Namen der Methoden zur Manipulation der Teilströme wurden so gewählt, daß aus ihrem Namen klar hervorgeht, welcher Teilstrom auf welchen Wert gesetzt wird. Der gesamte interne Aufbau des Gene-

rators wurde vollständig neu programmiert, mit verbesserten Datenstrukturen, so daß die Implementierung auch für andere Generatoren vom gleichen Typ eingesetzt werden kann. Auch die interne Verwaltung der Zustandsvektoren wurde soweit vereinheitlicht, daß jeder Zufallszahlengenerator seine eigene Kopie des Anfangszustandes erhält. So ist sichergestellt, daß die unterschiedlichen Instanzen sich nicht gegenseitig beeinflussen können.

Nachfolgende Punkte wurden gegenüber der Referenzimplementierung verändert, bzw. korrigiert:

Ansteuerung der Teilströme über *BigInteger* Für jede Methode, die eine Sprungstelle innerhalb des Zufallszahlenstroms ansteuert, sind zwei Varianten implementiert (vgl. Abbildung 2.2 auf der nächsten Seite). Eine Methode akzeptiert *long*-Datentypen, die zweite - allgemeinere - Methode *BigInteger*. Die Aufrufe über *BigInteger* sind nötig, da die Anzahl an Teilströmen in Ebene 1 (2^{64}) den maximalen Wert, den eine Variable vom Typ *long* annehmen kann ($2^{63} - 1$, vgl. Sun 2005, Abschnitt 4.2.1) überschritten wird. Eine exakte Ansteuerung der Teilströme ist somit nur möglich, wenn der eingesetzte Datentyp einen entsprechenden Gültigkeitsbereich besitzt, was auf *BigInteger* zutrifft. Gegenüber der Referenzimplementierung ist dies eine deutliche Verbesserung, da diese zur Ansteuerung nur *int* Variablen zuließ.

***long* als Datentyp für den internen Zustand** für die Darstellung des internen Zustandes wird ein eindimensionales Feld des elementaren Typs *double* verwendet. Der Typ wurde in *long* geändert, um die Geschwindigkeit zu erhöhen (siehe auch UML-Diagramm in Abbildung 2.3 auf Seite 21). Ein weiterer Grund ist die Minimierung von Rundungsfehlern. Auch der letzte Berechnungsschritt, in dem aus dem internen Zustand eine Fließkommazahl erzeugt wird, wurde so modifiziert, daß nur in einem Berechnungsschritt eine Fließkommazahl entsteht. Die Referenzimplementierung hingegeben benutzt zur Normierung noch einen Multiplikator, so daß an zwei Stellen Rundungsfehler entstehen können.

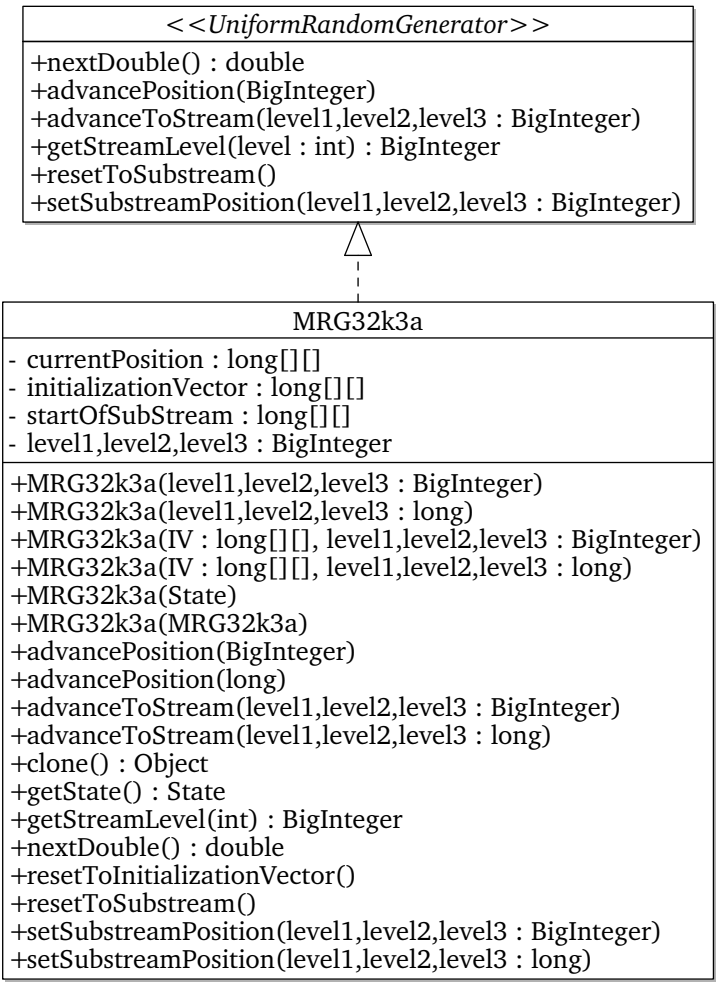


Abbildung 2.2: UML-Klassendiagramm *UniformRandomGenerator*

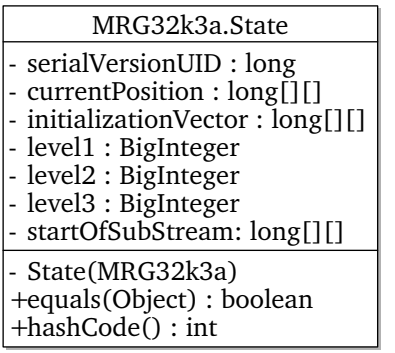


Abbildung 2.3: UML-Klassendiagramm der Zustandsklasse des Zufallszahlengenerators

Verwendung der Standardbibliotheken Die Referenzimplementierung benutzt für die Modulo-Arithmetik eigene Methoden. Diese wurden ersetzt durch die in der Java-Standardbibliothek vorhandenen Methoden. So ist gewährleistet, daß die Berechnungen nur Methoden benutzen, die ausreichend getestet sind (Bloch 2008, S. 215).

Anwendung des Memento-Pattern Zur Speicherung des internen Zustandes wurde eine statische lokale Klasse (State) erstellt, welche alle Informationen übernimmt. Als Entwurfsmuster wurde das sog. Memento-Pattern verwendet, welches das typsichere Speichern und Wiederherstellen von Zuständen erlaubt (Gamma u. a. 1995, S. 283). Die entsprechende Klasse ist im UML-Diagramm in Abbildung 2.3 dargestellt.

Serialisierung und *clone* Der Zufallszahlengenerator und das State-Objekt sind vollständig serialisierbar und klonbar. Die Serialisierbarkeit ist die zwingende Voraussetzung zum Einsatz innerhalb der Simulation (vgl. Abschnitt 4.3 auf Seite 85).

Berechnung der Sprungmatrizen Da die Größe der Teilströme in der Implementierung vorgegeben wird, ist der Wert der Matrizen zur

Navigation innerhalb der Teilströme konstant. Da sie als statische Variablen deklariert sind, berechnet das Programm die Sprungmatrizen nur beim ersten Aufruf.

Kapitel 3

Theorie zur Erzeugung von Testdaten

Testdaten sind einer der Kernbestandteile von rechnergestützten Experimenten, so daß eine sorgfältige Parametrisierung und Erzeugung Voraussetzung für die auf ihrer Grundlage berechneten Auswertungen ist. Im Laufe dieses Kapitels werden verschiedene Ansätze und Algorithmen für Testdatengeneratoren vorgestellt, wobei der Fokus auf Losgrößenproblemen liegt. Da viele Begriffe in einem unterschiedlichen Kontext genutzt werden können, definiert der erste Abschnitt alle hier verwendeten, nicht eindeutigen Begriffe. Nach einer Literaturübersicht werden die Anforderungen an einen Testdatengenerator definiert, sowie Klassifikationskriterien vorgestellt. Der folgende Abschnitt stellt parallel zur Entwicklung einer Simulation den Entwicklungsprozess eines Testdatengenerators vor. Daran schließt ein weiterer Abschnitt an, der verschiedene Arten zur Generierung von Testdaten beschreibt. Der letzte Abschnitt des Kapitels ist weitgehend unabhängig von den vorangehenden Teilen und stellt verschiedene Java-Klassen vor. Diese wurden speziell für den Einsatz in Testdatengeneratoren entwickelt, sowie den darauf aufbauenden Programmen, wie z. B. Simulationen.

3.1 Definitionen

Die Bewertung von Algorithmen zur Lösung von mathematischen Modellen erfolgt meist durch den Vergleich von Datensätzen, die aus der Praxis entnommen wurden und/oder konstruierten (synthetischen) Datensätzen. Letztere werden eingesetzt, um systematisch das Verhalten des Algorithmus in verschiedenen Umweltsituationen zu untersuchen.

Definition 1 (Testdatengenerator)**.** *Ein Testdatengenerator ist ein Algorithmus, der aus einer Menge an Eingabeparametern einen Datensatz erzeugt, der als Eingabe für ein Modell dient. Die Eingabeparameter repräsentieren die vollständige Menge an Merkmalen, um eine Umweltsituation zu beschreiben, die durch das Modell abgebildet werden soll.*

Der Begriff Modell bezeichnet sowohl ein mathematisches Modell zur linearen/gemischt-ganzzahligen Optimierung (auch als MIP=Mixed Integer Programming Modell bezeichnet), als auch ein Modell zur Simulation. Im Bereich der Testdatengeneratoren bezieht sich der Begriff Modell immer auf das lineare/gemischt-ganzzahlige Modell. Zur Unterscheidung wird für den anderen Fall der Begriff Simulationsmodell verwendet.

Ein Merkmal ist für einen Testdatengenerator eine Größe, die eine Eigenschaft der später erzeugten Daten charakterisiert. Jedes Merkmal besitzt unterschiedliche Ausprägungen, die den Merkmalswert angeben. Merkmale lassen sich gemäß ihrer Beziehungen in unabhängige und abhängige Merkmale unterteilen. Die Ausprägungen der abhängigen Merkmale lassen sich aus den Ausprägungen der unabhängigen Merkmale ableiten. Die Menge aller unabhängigen Merkmale bilden die Eingabedaten eines Testdatengenerators. Hierfür können nur quantitativ meßbare Merkmale benutzt werden. Aus der Statistik stammt der Begriff Faktor und bezeichnet hier ein unabhängiges Merkmal, welches in der Menge der Eingabeparameter enthalten ist. Im Zusammenhang mit statistischen Tests wird ebenfalls der Begriff Faktor verwendet. Dieser kann sich jedoch auch auf eine Größe beziehen, die als abgeleitete Größe von einem Testdatengenerator erzeugt wurde.

Der Testdatengenerator erzeugt aus einer Kombination von Ausprägungen aller unabhängigen Merkmale eine Testinstanz. Die Zusammenfassung mehrerer Testinstanzen ist ein (Test-) Datensatz. Dieser entsteht durch die Variation der Ausprägungen eines oder mehrerer Merkmale.

3.2 Literatur zum Einsatz und zur Erzeugung von Testdaten

Der folgende Abschnitt stellt verschiedene Bereiche aus der gemischtganzzahligen Optimierung vor, in denen Testdatengeneratoren oder Daten aus der Praxis verwendet wurden, um die Effizienz von Algorithmen zur Lösung eines Modells zu messen. Vor einem Einsatz von Testdaten müssen einige grundlegende Aspekte zu Computer-gestützten Experimenten beachtet werden. Diese sind auch die Grundlage für die in Abschnitt 3.3.1 aufgeführten Anforderungen an Testdatengeneratoren. Die im Folgenden vorgestellte Literatur ist nicht als vollständig anzusehen, vielmehr soll ein Überblick über unterschiedliche Anforderungen sowie Arten von Testdaten gegeben werden.

3.2.1 Anforderungen an Computer-gestützten Experimente

Im Bereich der Computer-gestützten Experimente werden Testdaten eingesetzt, um die Qualität eines Lösungsalgorithmus zu beurteilen. Die Qualität bemißt sich dabei z. B. an der Zeit, bis eine Lösung gefunden ist oder an dem Abstand der gefundenen Lösung zur optimalen Lösung. Die Daten, die für diesen Vergleich eingesetzt werden, haben einen erheblichen Einfluß auf die Ergebnisse. Daher sind an sie die gleichen Anforderungen zu richten, wie auch an das Experiment selbst.

Einen zusammenfassenden Beitrag zum Design und zur Durchführung von Experimenten geben Crowder u. a. (1978, S. 316). Bevor ein Experiment gestartet wird, muß das Ziel der Untersuchung, sowie die zu untersuchenden Einflußfaktoren definiert werden. Die Testdaten, die

zur Untersuchung herangezogen werden, lassen sich in zwei Kategorien einteilen: Praxisdatensätze („hand-selected-problems“) und zufällig erzeugte Datensätze („randomly generated problems“) (Crowder u. a. 1978, S. 319). Der Vorteil der Praxisdatensätze ist ihre Verbindung zu den in der Realität auftretenden Problemen. Die Nachteile liegen in der Schwierigkeit der Beschaffung, sowie der Einordnung der einzelnen Faktoren in eine stochastische Verteilung, aus der sie stammen. Eine Ableitung von weiteren Datensätzen auf Basis der Praxisdatensätze steht somit immer unter dem Vorbehalt, daß für einige Faktoren die stochastische Verteilung nicht bekannt ist. Im Gegensatz dazu können bei zufällig erzeugten Testdaten die Verteilungen kontrolliert werden, da sie bekannt sind bzw. ein Teil der Untersuchung sind. Im Gegensatz zu Praxisdatensätzen müssen zufällig erzeugte Testdatensätze so konfiguriert werden, daß sie mit den in der Realität auftretenden Anforderungen übereinstimmen.

Ein weiterer wesentlicher Punkt, der von Crowder u. a. (1978, S. 321) angesprochen wird, ist die Reproduzierbarkeit. Ein Experiment muß so dokumentiert sein, daß es zu jedem Zeitpunkt wiederholt werden kann. Voraussetzung hierfür ist die genaue Dokumentation der Experimentierumgebung sowie der Testdaten. In diesem Punkt gibt es keine Präferenz zwischen Praxisdaten und Testdatengeneratoren. Der einzige Vorteil für die Testdatengeneratoren liegt darin, daß die Kennzahlen zu Charakterisierung eines Datensatzes meist schon als Eingabeparameter bekannt sind, während sie bei Praxisdaten ermittelt werden müssen. Konkrete Ausprägungen der Forderung nach Reproduzierbarkeit von Experimenten ist auch die Angabe der Programmiersprache, des Compilers (heute des Bytecode-/Interpreters) sowie eine genaue Beschreibung des Algorithmus oder Angabe des Quellcodes. Crowder u. a. (1978, S. 322) beschränken die Reproduzierbarkeit darauf, daß die Ergebnisse innerhalb einer gewissen Toleranz liegen, die aufgrund der technischen Umgebung erzeugt wird. Eine Kernforderung ist, daß ein Experiment mindestens vom Durchführenden beliebig oft wiederholt werden kann, mit Ergebnissen, die innerhalb der Toleranzgrenzen liegen. Die Forderung, daß jeder das Experiment wiederholen kann, wird nicht erhoben. Als Begründung führen die Autoren an, daß die Daten in

vielen Fällen das Ergebnis eines oder mehrerer Programme sind, deren Erstellung umfangreiche Programmierarbeit erfordert.

Dieser Punkt wird von Jackson u. a. (1991, S. 417) noch einmal aufgegriffen. Sie fordern, daß selbst wenn der Code nicht veröffentlicht wird, die Beschreibung des Algorithmus so detailliert sein muß, daß die Funktionsweise (nicht zu Verwechseln mit der Implementierung) nachvollziehbar ist. Der Einsatz von Testdatengeneratoren wird ebenfalls empfohlen, neben dem Einsatz von Testdatensätzen, die aus der Literatur bekannt sind. Eine alleinige Konzentration auf Datensätze aus der Literatur wird kritisch gesehen, da die Datensätze teilweise zu klein sind, um repräsentativ zu sein. Ein weiterer Einwand besteht aufgrund der Gefahr, daß die Algorithmen auf den Einsatz dieses Datensatzes optimiert werden und nicht auf eine breite Masse an Daten (Jackson u. a. 1991, S. 419). Auch Aussagen zur Performance eines Algorithmus aufgrund eines (kleinen) Testdatensatzes werden kritisch gesehen („Inferring performance characteristics from a test set is similar to inferring the average height of Americans by sampling heights of the presidents.“, Jackson u. a. 1991, S. 419). Abschließend wird keine generelle Empfehlung für oder gegen den Einsatz einer bestimmten Methode gegeben, sondern nur darauf hingewiesen, daß bei jedem Ansatz Vor- und Nachteile existieren, die bereits in der Designphase des Experimentes berücksichtigt werden sollen. Der Einsatz der einen oder anderen Methode muß immer berücksichtigen, welche Methode in der jeweiligen Fachrichtung bevorzugt eingesetzt wird. Nur so lassen sich Vergleiche zwischen unterschiedlichen Arbeiten herstellen.

Die generelle Forderung von empirischen Untersuchungen für Algorithmen wird von Hooker (1994) aufgestellt. Eine Untersuchung soll auf Daten basieren, die der Realität möglichst nahe kommen und nicht nur Grenzfälle oder Extrempunkte beschreiben (Hooker 1994, S. 202). Darin liegt das wesentliche Unterscheidungsmerkmal zwischen einer empirischen und analytischen Untersuchung - die Einbeziehung der Testinstanzen, die keine Extrempunkte abbilden, sondern die mit der Realität vergleichbar sind. Aussagen zur Leistungsfähigkeit eines Algorithmus lassen sich nur treffen, wenn die Charakterisierung der Eingabeparameter zum Gegenstand der Untersuchung wird.

Darauf aufbauend beschreibt McGeoch (1996) Methoden und Ansätze, um Algorithmen experimentell zu untersuchen. Sie übernimmt die Argumentation von Hooker (1994) und setzt sie in vielen Punkten mit konkreten Handlungsempfehlungen fort. Bezogen auf die Konstruktion der Testdaten und der Eingabeparameter, empfiehlt sie, daß die Parameter so gestaltet sein sollen, daß innerhalb einer Testkonfiguration nur geringe Varianzen zwischen den Instanzen auftreten. Mit einer Änderung der Konfiguration („design-point“) müssen auch signifikante Änderungen in den gemessenen Größen feststellbar sein. Damit verbunden ist auch die Fragestellung, welche Eingabeparameter eine entsprechende Signifikanz aufweisen. Die Antwort auf diese Frage läßt sich nur durch Experimentieren herausfinden. McGeoch (1996, S. 6) beschreibt dies als einen evolutionären Prozeß, in dem durch die Experimente Hypothesen aufgestellt und verworfen werden, bis ein entsprechender Satz an Parametern gefunden wird. Da während dieser Phase die Varianz der untersuchten Größen sehr hohe Werte annehmen kann, empfiehlt sie, die Anzahl der Tests zu erhöhen. Sie geht nicht näher darauf ein, wie die Anzahl erhöht wird. Liegen dem Experiment Praxisdatensätze zu Grunde, so ist eine Erhöhung der Anzahl fast immer unmöglich. Der Einsatz eines Testdatengenerators wird hier indirekt empfohlen, da so eine beliebige Anzahl an Testinstanzen erzeugt werden kann. McGeoch (1996, S. 9) spricht aber auch einen weiteren interessanten Punkt an - die Größe des zu untersuchenden Problems. Diese sollte so gewählt werden, daß sie mit der aktuell zur Verfügung stehenden Technologie gerade noch lösbar ist. Damit verbunden ist das Problem, daß Testdatensätze aus der Literatur auch einer Alterung unterworfen sind und nach einem Zeitraum von z. B. zehn Jahren aufgrund der Entwicklung in der Rechengeschwindigkeit und zum Teil auch der Algorithmen keine Herausforderung mehr darstellen. Die Schlußfolgerungen, die aus kleinen Testdatensätzen gezogen werden, sind nicht immer auf größere übertragbar. Sie führt ein Beispiel auf, in dem Effekte erst sichtbar werden, wenn die Problemgröße ein kritisches Maß übersteigt (McGeoch 1996, S. 13). Auch dies spricht indirekt für den Einsatz eines Testdatengenerators, da bei Praxisdatensätzen die Größe festgeschrieben ist.

Ein weiterer wichtiger Punkt ist der Einsatz eines erprobten Zufallszahlengenerators (McGeoch 1996, S. 11). Reproduzierbarkeit erstreckt sich nicht nur auf das Experiment, sondern auch auf die gesamte Umgebung. Ein Zufallszahlengenerator, der vom System bereitgestellt wird, kann auf einem anderen Betriebssystem oder in einer anderen Programmiersprache anders implementiert sein, so daß die Ergebnisse sich auch aufgrund von Rundungsfehlern signifikant unterscheiden können (vgl. Kapitel 2).

3.2.2 Einsatz eines Testdatengenerators

Ein Beispiel aus dem Bereich der **Losgrößenplanung** ist die Arbeit von Trigeiro u. a. (1989). Aufgrund der Nicht-Verfügbarkeit von Testdaten für das Capacitated Lot Sizing Problem (CLSP) mit Rüstzeiten entsteht hier die Notwendigkeit, eigene Daten zu erzeugen. Die Daten werden so generiert, daß auf Basis der Eingabedaten alle Werte, bis auf die Kapazität, aus stochastischen Verteilungen gezogen werden. Die Kapazität ist die letzte abhängige Größe, die keinen Freiheitsgrad besitzt, und deshalb in Abhängigkeit von den anderen Größen bestimmt werden muß. Datensätze, für die sich mit Hilfe einer Heuristik keine Lösung finden läßt, werden ausgeschlossen. Für Probleme mit einer hohen Kapazitätsauslastung muß dieser Schritt häufiger durchgeführt werden, um die geforderte Anzahl an Testinstanzen zu erhalten. Die Testdaten enthalten daher nur die leichter zu lösenden Probleme. Trigeiro u. a. (1989, S. 359) erkennen, daß diese Selektion eine Verfälschung der Datensätze erzeugt. Diese wird jedoch als unvermeidlich akzeptiert.

Ein weiteres Problem ergibt sich durch die Konstruktion der Anfangsbedingungen. Für jedes Produkt wird kein Anfangslagerbestand vorgegeben, so daß die Gefahr besteht, daß die Kapazität zu Beginn nicht ausreicht, um die Nachfrage zu erfüllen. Zur Verringerung der Nachfrage wird diese in einer der ersten vier Perioden auf Null gesetzt. Die entsprechende Nachfrage wird auf die letzten Perioden innerhalb des Planungszeitraums verteilt, um die mittlere Nachfrage zu erhalten. Es entsteht eine Nachfragereihe mit einem steigenden Trend. Die erzeugten Testdaten wurden von anderen Autoren übernommen, um

weitere Lösungsverfahren zu Testen (z. B. von Gopalakrishnan u. a. 2001) oder um verwandte Modellformulierungen zu untersuchen. Ein Beispiel hierfür ist die Capacitated Lot Sizing Problem with Linked lot sizes (CLSPL) Formulierung von Suerie u. Stadtler (2003).

Das von Trigeiro u. a. (1989) eingesetzte Verfahren zur Erzeugung der Testdaten ist nicht unproblematisch, da es die Nachfragereihe mit einem Muster versieht, was zu einer Verfälschung der Daten führt. Damit widerspricht es der von Hall u. Posner (2001, S. 855) aufgestellten Anforderung, daß die erzeugten Daten unverfälscht sein sollen. Eine weitere Verfälschung der Daten wird durch die Überprüfung der Lösbarkeit erzeugt. Trigeiro u. a. (1989) verwenden eine Heuristik, um die Lösbarkeit zu überprüfen. Probleme, die von der Heuristik nicht gelöst werden können, werden verworfen und es werden neue Probleme erzeugt. Es wird nicht angegeben, wie viele dieser Probleme eine zulässige Lösung besitzen und die Heuristik versagt hat. Sie erkennen, daß es zu einer Verfälschung kommt, die speziell bei Problemen auftritt, die eine hohe Ressourcenbelastung haben (Trigeiro u. a. 1989, S. 359). Der Testdatensatz enthält somit keine repräsentative Menge an Daten, sondern nur Daten, die eine geringe Ressourcenbelastung besitzen.

Ein weiterer Testdatensatz aus dem Bereich der mehrstufigen Planung wurde von Tempelmeier u. Helber (1994) entwickelt. Auch hier fehlte wiederum ein Referenzdatensatz, so daß die Autoren einen neuen Testdatensatz entwickelt haben. Für jedes Merkmal wurde eine Gruppe von Profilen (Ausprägungen) erzeugt. Nach Kombination der Ausprägungen aller Merkmale entstand ein umfangreicher Testdatensatz. Die Lösbarkeit jedes Datensatzes wurde jedoch nicht aufgrund der Konstruktion erzwungen, sondern wie bei Trigeiro u. a. (1989) mit Hilfe einer Heuristik oder eines Programms zum Lösen von gemischt-ganzzahligen Problemen gezeigt.

In einer späteren Arbeit setzen Tempelmeier u. Derstroff (1996) Methoden ein, die mit denen von Tempelmeier u. Helber (1994) sowie Trigeiro u. a. (1989) vergleichbar sind. Für die Erzeugung des Endkundenbedarfs wird das Verfahren von Trigeiro u. a. (1989) übernommen. Damit verbunden ist gleichzeitig die Verfälschung durch den ansteigenden Bedarf. Tempelmeier u. Helber (1994, S. 750) erkennen diesen,

bieten jedoch keine Alternative an. Gleichzeitig wird, um die Lösbarkeit sicherzustellen, die Nachfrage der einzelnen Perioden so vertauscht, daß die verfügbare Kapazität nicht überschritten wird (Tempelmeier u. Helber 1994, S. 750). Dies stellt eine weitere Verfälschung der Daten dar, die einen Trend in der Nachfragereihe erzeugt. Der Testdatensatz wird durch ein Voll-Faktorielles-Design erzeugt. Die Faktoren umfassen die Struktur des Produktionsnetzwerkes, die Nachfragereihe, die TBO und die Zielauslastung. Die Nachfrage wird über den Variationskoeffizienten charakterisiert, so daß für jedes Niveau fünf zufällige Nachfragereihen erzeugt werden. In einem anderen Testdatensatz, der mit den gleichen Methoden erzeugt wurde, konnten einige Testinstanzen von der Heuristik nicht gelöst werden, obwohl eine ganzzahlige Lösung existiert. Bei anderen Testinstanzen wiederum konnte weder die Heuristik, noch ein exaktes Verfahren innerhalb der vorgegebenen Zeitschranke eine Lösung finden (Tempelmeier u. Derstroff 1996, S. 754). Eine genauere Untersuchung der Ursachen wird nicht durchgeführt. Aufgrund der angegebenen Parameter kann jedoch davon ausgegangen werden, daß mit steigender Auslastung die Heuristik Schwierigkeiten hat, einen Lösung zu finden. Instanzen, bei denen auch mit dem exakten Verfahren keine gültige Lösung gefunden wurde, wurden nicht weiter untersucht.

Im Bereich der Losgrößenplanung ist es generell ein Problem, Daten mit einer vorgegebenen Auslastung zu erzeugen, wenn Rüstzeiten berücksichtigt werden müssen. Die Auslastung, die während der Datenerzeugung angenommen wird, stellt meist nur eine obere Schranke dar. Die tatsächliche Auslastung der Ressourcen kann erst nach der Lösung des Problems ermittelt werden. Diese hängt von der Anzahl der Rüstvorgänge ab, die nach der Zusammenfassung von Losen entsteht (vgl. Tempelmeier u. Helber 1994, S. 750).

Die für den Bereich der Losgrößenplanung angesprochenen Probleme bei der Erzeugung von unverfälschten Testdaten existieren auch für andere gemischt-ganzzahlige Modelle. Im Bereich der **Projektplanung** wurde von Kolisch u. a. (1995) ein Testdatengenerator entwickelt, der umfangreiche Einstellungsmöglichkeiten bietet. Das Ziel war die Erstellung einer Bibliothek mit Testproblemen, die mit geringem Aufwand erweitert werden kann, indem die Größe der Probleme an die aktuelle

Entwicklung angepaßt wird (Kolisch u. Sprecher 1996, S. 205). Merkmale, die von diesem Generator als Eingabefaktor verarbeitet werden, sind die minimale/maximale Anzahl an Aufträgen, Anzahl der Modi (je Knoten) und Dauer eines Auftrags auf einem Knoten. Der Aufbau des Netzes geschieht durch Vorgabe einer maximalen/minimalen Anzahl an Start-/Endvorgängen. Das weitere Netzwerk wird so erzeugt, daß zunächst jedem Knoten ein Vorgänger- und Nachfolgerknoten zugewiesen wird. Anschließend werden zusätzliche Kanten hinzugefügt, wobei redundante Kanten entfernt werden, bis die vorgegebene Netzdichte erreicht ist. Die weitere Erzeugung des Ressoucenverbrauchs, sowie der Zuordnung Auftrag-Modus-Ressource erfolgt unter Hinzunahme weiterer Kennziffern.

Die Konstruktion des Testdatensatzes erfolgt durch ein Voll-Faktorielles-Design von drei Merkmalen, die als Kennziffern vorgegeben werden Kolisch u. a. (1995, S. 1699). Hervorzuheben ist, daß die Autoren für den Testdatengenerator einen eigenen Zufallszahlengenerator verwenden, um eine vollständige Reproduzierbarkeit der erzeugten Testinstanzen zu erreichen (Kolisch u. a. 1995, S. 1695). Der Zufallszahlengenerator beruht auf einer Implementierung von Schrage (1979, S. 132), der in Fortran einen einfachen linearen Kongruenzautomaten implementiert hat.

In einem Vergleich von verschiedenen Testdatengeneratoren aus dem Bereich der Projektplanung zeigen Vanhoucke u. a. (2008) Designfehler in der Erzeugung der Testdaten sowie Möglichkeiten zur Darstellung des Raums, aus dem die Testdaten stammen. Für ihren Vergleich verwenden Vanhoucke u. a. (2008) sechs Indikatoren zur Beschreibung der Topologie eines Netzwerkes. Die Größe des Netzwerkes wird auf 30 Aktivitäten beschränkt. Bei der Erzeugung eines Netzwerkes gelten für jeden Generator identische Beschränkungen: die Zeit zum Erzeugen des Netzwerkes, sowie die maximale Anzahl an Wiederholungen (Erzeugung eines bereits vorhandenen Netzwerkes) bevor die Erzeugung abgebrochen wird. Die Indikatoren werden miteinander kombiniert, und es wird überprüft, ob für jede Kombination an Indikatoren ein Netzwerk erzeugt wurde. Die Autoren merken kritisch an,

daß sich die Vergleiche nur auf die Werte der Indikatoren beschränken und nicht auf die inhaltliche Identität der erzeugten Netzwerke.

Der Generator von Kolisch u. a. (1995) wird in diesen Vergleich mit einbezogen. Es zeigt sich, daß dieser Generator nicht alle Parameterkombinationen abdecken kann. Ursache hierfür ist die Art, in der das Netzwerk erzeugt wird. Diese verhindert, daß bestimmte Topologie erzeugt werden, was wiederum Auswirkungen auf andere Kennziffern hat. Interessant ist auch der Vergleich der existieren Testinstanzen aus diesem Bereich. Eine systematische Einordnung anhand der Kennziffern zeigt, daß diese sich in einem kleinen Bereich konzentrieren und somit eine ähnliche Charakteristik aufweisen. Im Rahmen dieses Vergleichs zeigt sich, daß der Generator von Vanhoucke u. a. (2008), der auf den Arbeiten von Demeulemeester u. a. (2003) aufbaut, die größte Vielfalt an Topologien erstellen kann. Das Grundprinzip der Erzeugung beruht im Gegensatz zu Kolisch, Sprecher, u. Drexl darauf, daß von einem vollständig verbundenen Netzwerk ausgegangen wird, bei dem Kanten gelöscht werden, bis die gegebenen Kennzahlen erreicht werden. Dabei wird darauf geachtet, daß jeder Auftrag weiterhin mit dem Netzwerk verbunden bleibt.

Der Vergleich der verschiedenen Testdatengeneratoren zeigt, daß selbst bei einer sorgfältigen Konstruktion eine Verfälschung der Testdaten auftreten kann. Der von Vanhoucke u. a. (2008) präsentierte Ansatz zur Konstruktion von Netzwerken kann auch auf andere Bereiche, in denen Testdaten Netzwerke mit komplexen Abhängigkeiten verwenden, übertragen werden.

Ein anderes Feld, in dem Testdatengeneratoren eingesetzt werden, ist das **Cutting-Stock** Problem. Dieses behandelt die Zuteilung von Aufträgen auf Werkstücke mit einer vorgegebenen Länge. Das Ziel ist die Minimierung der verbrauchten Werkstücke und damit auch gleichzeitig die Minimierung des Verschnitts. Ein Generator für das eindimensionale Zuschnittproblem wird von Gau u. Wäscher (1995, S. 572) vorgestellt. Die Autoren identifizieren insgesamt 5 Parameter, um eine Testinstanz zu charakterisieren. Der Generator übernimmt diese Parameter als Eingabedaten und erstellt hieraus die Testdaten. Im Unterschied zu den vorangehenden Generatoren können bei diesem Generator keine un-

zulässigen Lösungen erzeugt werden, da das Modell die Anzahl der verfügbaren Werkstücke nicht beschränkt. Für jedes dieser Merkmale werden unterschiedliche Ausprägungen vorgegeben, die durch eine Voll-Faktorielles-Design zu einem Testdatensatz kombiniert werden. Der Testdatengenerator verwendet wie auch einige der vorher genannten einen eigenen Zufallszahlengenerator, der ebenfalls auf dem einfachen linearen Kongruenzautomaten von Schrage (1979, S. 132) basiert. Der Generator wird in späteren Arbeiten eingesetzt, um die Leistungsfähigkeit von Heuristiken miteinander zu vergleichen (Wäscher u. Gau 1996, S. 131). Ein Generator der gleichen Bauart wird von Schwerin u. Wäscher (1997, S. 377) vorgestellt. Dieser erzeugt Daten für eine Variante des **Bin-Packing** Problem, welches mit dem Cutting-Stock Problem identisch ist.

Ein weiteres Modell, für das ein bekannter Testdatengenerator entwickelt wurde, ist das **General-Assignment** Problem. Dieses beschreibt die Zuweisung zwischen Aufgaben und Ressourcen, so daß jede Aufgabe von exakt einer Maschine ausgeführt wird. Die Kosten und der Ressourcenverbrauch können für jeden Auftrag und jede Maschine unterschiedlich sein. Trick (1992, S. 137) entwickelt eine neue Heuristik, die auch für größere Testinstanzen geeignet ist. Dabei stellt er fest, daß die bisher in der Literatur verwendeten Generatoren nur für kleine Testinstanzen geeignet sind. Für größere Testinstanzen verfälschen diese aufgrund ihrer Konstruktion die Daten, indem sie die Kapazität der Maschinen zu groß ansetzen (Trick 1992, S. 146). Der neue Ansatz besteht darin, daß nur noch eine Größe zur Konstruktion der unteren Schranke verwendet wird und diese in mehreren Schritten so weit zu lockern, daß die Probleme gerade noch lösbar sind. In unterschiedlichen Testdatensätzen werden Modifikationen des Generators getestet. Als Eingabeparameter werden immer die Anzahl an Aufträgen/Maschinen sowie die verfügbare Kapazität gewählt. Für jedes dieser Merkmale existieren unterschiedliche Ausprägungen, die durch ein Voll-Faktorielles-Design zu einem Testdatensatz zusammengestellt werden. Diese Testdatensätze werden auch von anderen Autoren eingesetzt, um neue Lösungsverfahren zu untersuchen.

Aufgrund der Struktur des General-Assignment Problems entwickeln

Romeijn u. Morales (2001, S. 866) ein stochastisches Modell, welches den Prozeß der Datenerzeugung nachbildet und eine analytische Schranke für die Lösbarkeit des Modells enthält. Anhand dieses Modells vergleichen sie fünf Generatoren und können eine Aussage zur Qualität des Generators treffen. Die Qualität wird definiert über den Abstand der von einem Generator erzeugten Daten zu der Schranke. Die analytischen Ergebnisse werden durch einen experimentellen Vergleich belegt. Es wird gezeigt, daß die Erzeugung der Daten einen nachweisbaren Effekt auf die Lösbarkeit des Modells durch eine Heuristik hat.

Ein mit dem General-Assignment Problem verwandtes Problem ist die Klasse der **Nurse-Scheduling** Probleme. Modelle aus dieser Klasse erzeugen einen Dienstplan für z. B. ein Krankenhaus, bei dem die Arbeitnehmer auf einzelne Schichten verteilt werden. Jeder Arbeitnehmer kann für eine Schicht eine Präferenz angeben, die das Modell zu berücksichtigen versucht. Das Ziel ist die Minimierung der Strafkosten für die Nicht-Einhaltung der gewünschten Präferenzen unter Berücksichtigung von harten Nebenbedingungen, wie z. B. eine Mindestbesetzung je Schicht oder Abstände zwischen zwei Schichten eines Arbeitnehmers. Da die meisten Arbeiten auf diesem Gebiet für einen Anwendungsfall erstellt wurden, sind die Nebenbedingungen inhomogen und somit nicht miteinander vergleichbar. Für einen objektiven Vergleich der unterschiedlichen Lösungsalgorithmen ist eine gemeinsame Datenbasis notwendig, wie sie von Vanhoucke u. Maenhout (2009, S. 457) gefordert wird. In ihrer Arbeit entwickeln sie daher Merkmale zur Charakterisierung von Dienstplänen und erstellen einen Testdatengenerator, der diese Merkmale als Eingabe verwendet. Außer den Standardgrößen (wie Anzahl der Arbeitnehmer, Schichten, Periodenlänge) werden auch Komplexitätsmaße entwickelt. Diese beschreiben die Homogenität von Präferenzen bezogen auf Arbeitnehmer, Schichten und Tage. Ein weiteres Komplexitätsmaß ist die Abdeckung der Tage und Schichten durch das Personal. Der Generator arbeitet so, daß anhand einer Kennzahl ein zufälliger Schichtplan erstellt wird, bei dem die Schichten/Tage solange vertauscht werden, bis auch die anderen Kennzahlen erfüllt werden. Der Austausch der Schichten/Tage erfolgt mit Hilfe einer lokalen Suche, die sich in Richtung der besseren Kennzahlen hin entwickelt.

Für jede der sechs Kennzahlen werden unterschiedliche Ausprägungen gewählt, die durch ein Voll-Faktorielles-Design zu einem Datensatz miteinander kombiniert werden. Der gesamte Datensatz wird mit Hilfe eines gemischt-ganzzahligen Modells und Anwendung eines Branch & Bound Verfahrens untersucht. Als Beurteilungsmaß für die Schwierigkeit der Testinstanz dient die Lösungsdauer. Durch ein Regressionsverfahren, welches unterschiedliche Faktoren berücksichtigt, kann auf den Einfluß der einzelnen Faktoren geschlossen werden. Diese Werte werden überprüft, indem der gesamte Datensatz vorher in einen Trainingsteil und einen Testteil aufgespalten wird (Vanhoucke u. Maenhout 2009, S. 464). Die Anwendung dieser statistischen Verfahren zur Untersuchung der Einflußfaktoren auf die Effizienz eines Algorithmus stellt eine Umsetzung der von Hooker (1994) geforderten empirischen Untersuchung dar.

Ein weiterer Testdatengenerator wird von Kimms u. Müller-Bungart (2007) für den Bereich der **Network-Revenue-Management** Probleme vorgestellt. Die Motivation zur Entwicklung des Generators ist auch hier wieder die Nicht-Existenz einer einheitlichen Testumgebung. Der Generator besteht aus mehreren Komponenten: einem Netzwerk, den Produkten, die Kapazität auf den Kanten sowie die Nachfrage der Kunden nach den einzelnen Produkten. Das Netzwerk wird nicht automatisch erstellt, sondern muß durch den Benutzer von Hand erstellt werden. Das Programm leistet Unterstützung, indem es eine visuelle Entwicklungsumgebung bereitstellt. Auch die Daten für die Produkte und Kapazitäten müssen vom Benutzer vorgegeben werden. Das Programm stellt lediglich für die Erzeugung der stochastischen Nachfrage Unterstützung in Form eines Algorithmus bereit (Kimms u. Müller-Bungart 2007, S. 17). Im Unterschied zu den bisher betrachteten Generatoren wird hier ein Zufallszahlengenerator neuerer Bauart von L'Ecuyer (1999) eingesetzt (vgl. Abschnitt 2.3 auf Seite 15).

3.2.3 Praxisdatensätze

Im Gegensatz zu Datensätzen, die von Testdatengeneratoren erzeugt wurden, spiegeln Praxisdatensätze Probleme wider, die in der Realität so

oder so ähnlich vorkommen. Sie haben gegenüber einem Testdatengenerator den Vorteil, daß Algorithmen und Modelle an den realistischen Anforderungen getestet werden und nicht an synthetischen Datensätzen, die zum Teil auch Grenzfälle enthalten. Der wesentliche Nachteil ist, daß sie veralten. Durch die Fortschritte im Bereich der Computer, als auch der Lösungsalgorithmen (Bixby 2002) werden Probleme, die zum Zeitpunkt der Entstehung nur schwer lösbar waren, nach einigen Jahren zu leicht, um noch als Referenzdatensätze zu dienen.

Eine aktuelle Sammlung an Praxisdatensätzen bietet die sog. MIPLIB (Achterberg u. a. 2009). Diese besteht seit 1992 und wurde seither immer wieder aktualisiert. Probleminstanzen, die schneller gelöst werden konnten, wurden gegen neue Instanzen ausgetauscht. Die aktuelle Version ist MIPLIB 2003, die zum jetzigen Zeitpunkt immer noch gepflegt und aktualisiert wird. Das Problem der veralteten Datensätze wird hier durch den kontinuierlichen Austausch und die Bereitstellung von neuen Praxisdatensätzen umgangen. Einen aktuellen Überblick der Probleme und ihrer besten bekannten Lösungen gibt Laundy u. a. (2009, S. 304). Weitere Datensätze aus dem Bereich der Losgrößenplanung stellen Belveaux u. Wolsey (2001) vor.

3.3 Testdatengenerator

3.3.1 Anforderungen an Testdatengeneratoren

Die im folgenden definierten Anforderungen leiten sich aus den Anforderungen ab, die in Abschnitt 3.2.1 für Computer-gestützte Experimente festgelegt wurde.

Definition 2 (Anforderungen an einen Testdatengenerator)**.** *Hall u. Posner (2001, S. 855) definieren folgende Designprinzipien, die für jeden Testdatengenerator befolgt werden sollen, unabhängig vom Einsatzbereich:*

1. *Bestimmtheit*
2. *Vergleichbarkeit*

3. Erzeugung von unverfälschten Werten

4. Reproduzierbarkeit

Bestimmtheit fordert, daß die Menge an Daten erzeugt wird, die für die Durchführung des Experimentes notwendig ist. Voraussetzung ist, daß die Ziele definiert werden (vgl. Abschnitt 3.4 auf Seite 42) und die Menge an Eingabeparametern. Hall u. Posner (2001, S. 855) führen auf, daß die Menge an Daten im Wesentlichen vom Verwendungszweck abhängt. Für eine ausführliche empirische Untersuchung, wie sie von Hooker (1994) gefordert wird, sind mehr Daten notwendig, als für eine Untersuchung mit einem eng begrenzten Rahmen. Da die empirische Untersuchung ein evolutionärer Prozeß ist, in dem sich die Menge an Eingabeparametern ändern kann (McGeoch 1996, S. 6), ist diese Anforderung nicht als statisch anzusehen, sondern als allgemeines Designprinzip.

Vergleichbarkeit fordert, daß innerhalb einer Studie die gleichen Testdatensätze verwendet werden und keine grundlegenden Änderungen vorgenommen werden. Ähnlich wie von McGeoch (1996, S. 4) vorgeschlagen, sollte eine Referenzkonfiguration (von ihr „design-point" genannt) erstellt werden, deren Eingabeparameter entsprechend dem Gang der Untersuchung angepaßt werden.

Bei der Erzeugung der Daten ist darauf zu achten, daß **unverfälschte Werte** erzeugt werden. Hall u. Posner (2001) verstehen hierunter die Vermeidung von ungewollten Abhängigkeiten zwischen einzelnen Werten. Sie identifizieren zwei Ursachen: die Nachbildung von Lösungsstrukturen und den daraus resultierenden Abweichungen sowie die Korrelation von Daten aufgrund des Konstruktionsprozesses. Das Problem der Korrelation von Daten wird von Reilly (2009, S. 458) detaillierter beschrieben. Reilly unterscheidet zwischen impliziter und expliziter Korrelation. Implizite Korrelation tritt auf, wenn die Daten miteinander korreliert sind, ohne daß die Höhe der Korrelation kontrolliert werden kann. Die explizite Korrelation hingegeben macht die Korrelation zu einem Eingabeparameter und damit auch zu einem Untersuchungsgegenstand. Die Auswirkungen der Korrelation von Koeffizienten werden von Reilly (2009, S. 460) anhand von Beispielen näher erläutert. Es zeigt

sich, daß z. B. eine positive Korrelation zwischen den Gewichten und den Preisen das Knapsackproblem schwieriger macht. Die Lösungsdauer verlängert sich bei der Anwendung von Branch & Bound Verfahren signifikant. Aber auch andere Probleme, wie das General-Assignment Problem oder das Set-Covering Problem sind von der Korrelation betroffen. Die Forderung lautet daher, die Korrelation explizit kenntlich zu machen, um sie als Einflußfaktor zu identifizieren.

Die **Reproduzierbarkeit** umfaßt bei einem Testdatengenerator die Eigenschaft einen Datensatz aus einer definierten Menge an Eingabeparameter mit einem deterministischen Algorithmus zu erzeugen. Daraus folgt, daß der Algorithmus ausschließlich die zur Verfügung gestellten Eingabeparameter verwenden darf, so daß die Erzeugung des Datensatzes unabhängig von äußeren Einflüssen ist. Als Konkretisierung ist zu nennen: die Unabhängigkeit von der Hardwareplattform oder dem Betriebssystem. Diese ist notwendig, da die Ergebnisse von Fließkommaoperationen aufgrund der Rundungsfehler von der eingesetzten Hardwareplattform abhängen. Die Programmiersprache muß daher die entsprechenden Standards definieren bzw. eine Bibliothek bereitstellen, die z. B. alle Operationen gemäß IEEE Standard durchführt.

Falls der Testdatengenerator stochastische Verteilungen einsetzt, indem ein Zufallszahlengenerator verwendet wird, ist darauf zu achten, daß dieser Bestandteil des Algorithmus und der Implementierung ist (dieser Punkt wurde bereits in den vorangehenden Abschnitten z. B. von McGeoch 1996 gefordert). Andernfalls ist er von der Implementierung der Programmiersprache oder des Betriebssystems abhängig und damit nicht reproduzierbar.

Auf Basis dieser Anforderungen beschreiben Hall u. Posner (2001, S. 856) Eigenschaften, die von den Testdaten erfüllt werden müssen. Eine wichtige Eigenschaft ist die Skalen- und Größen-Invarianz. Diese besagt, daß die Eigenschaften der Daten erhalten bleiben, wenn sich die Größe des Modells oder die Skalen ändert. So sollen Daten, die aus einer stochastischen Verteilung gezogen wurden, die Form ihrer Verteilung beibehalten, unabhängig von der Größenskala. Ein entsprechendes Verfahren wird in Abschnitt 3.5.5 auf Seite 57 vorgestellt. Weitere Eigenschaften werden von Hall u. Posner (2001, S. 857) in Form einer

Tabelle vorgestellt. Diese enthält eine detaillierte Beschreibung der Abhängigkeiten der Eigenschaften von den Anforderungen.

3.3.2 Klassifikationskriterien

Zur Klassifikation des Designs von Testdatengeneratoren wurden zwei wesentliche Kriterien identifiziert:

- Abhängigkeit der Merkmale
- feste oder lose Bindung von Ausprägungen eines Merkmals an die erzeugten Daten

Definition 3 (Abhängigkeit eines Merkmals)**.** *Die Abhängigkeit der Merkmale eines Testdatensatzes kann entweder interdependent oder intradependent sein (Hall u. Posner 2001, S. 860).*

Definition 4 (Bindung der Ausprägungen an Merkmale)**.**

a. *Eine feste Bindung eines Merkmals liegt vor, wenn bei identischen Ausprägungen eines Merkmals unabhängig von den Ausprägungen aller anderer Merkmale eine identische Menge an Daten erzeugt wird.*

b. *Eine feste Bindung für einen Datensatz liegt vor, wenn die erste Aussage für alle Merkmale gilt.*

Das erste Kriterium unterscheidet, ob während der Datengenerierung die Ausprägungen eines Merkmals von einer oder mehrerer Ausprägungen anderer Merkmale abhängig sind (interdependent oder auch gegenseitige Abhängigkeit). Die Art der Abhängigkeit kann einen beliebigen funktionalen oder logischen Charakter haben. Hall u. Posner (2001, S. 860) führt als Beispiel die Maschinenbelegungsplanung auf, bei der die Endzeiten der Aufträge von der Bearbeitungsdauer und den Startzeitpunkt abhängig sind (Intra- und Interdependenz). Der früheste Startzeitpunkt eines Auftrags liegt nach dem Endzeitpunkt des vorangehenden Auftrags (Interdependenz). Ein anderes Beispiel betrifft

die verfügbare Kapazität in der Losgrößenplanung. Die Summe der in einer Perioden produzierten Aufträge kann die verfügbare Kapazität nicht übersteigen. Bei Kenntnis der anderen Größen kann die abhängige Größe immer abgeleitet werden.

Eine Intradependenz liegt in jedem Fall vor, wenn eine Nebenbedingung berücksichtigt werden muß, die nicht von einem anderen Merkmal abhängig ist. So muß für das General-Assignment Problem sichergestellt sein, daß jeder Auftrag von mindestens einer Maschine erfüllt werden kann. Falls von n-Maschinen $n-1$ Maschinen einen Auftrag nicht erfüllen können, muß automatisch die letzte Maschine diesen Auftrag bearbeiten können.

Das zweite Kriterium betrifft die Art der Erzeugung des Datensatzes. Eine feste Bindung wird erzeugt, indem für alle Merkmale die Ausprägungen erzeugt werden und nur noch durch Kombination der Ausprägungen die Menge der Testinstanzen generiert wird. Der Vorteil dieses Verfahrens ist der geringe Aufwand in der Durchführung: es muß nicht für jede Testinstanz ein eigener Satz an Ausprägungen erstellt werden. Dieser Vorteil ist aber auch gleichzeitig ein Nachteil, da die Lösbarkeit so nicht für alle Testinstanzen gewährleistet werden kann. Beispiele hierfür sind die Testdatensätze von Tempelmeier u. Derstroff (1996) oder von Tempelmeier (2002) und Reith-Ahlemeier (2002).

Der Datensatz von Tempelmeier u. Derstroff (1996) entsteht durch Kombination aller Ausprägungen der untersuchten Merkmale. Dabei wurde festgestellt, daß nicht alle Testinstanzen Lösungen besitzen (vgl. Abschnitt 3.2.2 auf Seite 29). Ein weiteres Beispiel zeigt ein Vergleich der Arbeiten von Tempelmeier (2002) und Reith-Ahlemeier (2002). Die Daten in Reith-Ahlemeier (2002) sind die verkürzte Version der Daten von Tempelmeier (2002). Die ungekürzte Version ließ sich unter Berücksichtigung von zusätzlichen Kapazitätsrestriktionen von Reith-Ahlemeier nicht mehr anwenden. Für eine genauere Untersuchung wurden keine neuen Daten erzeugt, sondern nur noch eine Auswahl an Daten benutzt (und zum Teil auch verkürzt), bei der alle Instanzen eine zulässige Lösung besitzen. Ein anderes Beispiel, daß die Erzeugung der Daten von Tempelmeier u. Derstroff (1996) übernimmt, ist die Arbeit von Tempelmeier u. Buschkühl (2009). Auch hier wurden wieder Test-

instanzen erzeugt, die keine zulässige Lösung besitzen (Tempelmeier u. Buschkühl 2009, S. 399).

Im Gegensatz dazu kann bei Generatoren, die eine lose Datenbindung einsetzen die Lösbarkeit bereits während der Erzeugung der Daten berücksichtigt werden. Gemeinsam ist diesen Generatoren, daß sie als Eingabe Kennzahlen verwenden, auf deren Basis die Ausprägungen ermittelt werden. Der Testdatensatz wird durch Kombination aller Kennzahlen erstellt, wobei die Testinstanz für jede Kombination neu berechnet wird. Ein Beispiel hierfür ist der Generator von Vanhoucke u. Maenhout (2009).

3.4 Entwicklungsprozess

Die Erzeugung von Testdaten für gemischt-ganzzahlige Modelle ist in der Literatur bisher nur spärlich betrachtet worden. Im Gegensatz zur Simulation wurde bisher keine Systematik entwickelt, welche z. B. Designkriterien festlegt oder Verfahren vorstellt. Eine Ursache ist die Bestimmtheit des Testdatengenerators, da er immer eine Individualentwicklung für eine begrenzte Modellgruppe darstellt. Trotz dieses wesentlichen Unterschiedes lassen sich Schnittmengen zur Simulation feststellen. Im Folgenden wird ein Ansatz vorgestellt, der mit Ansätzen aus der Simulation verwandt ist.

Pidd (2004, S. 29) beschreibt den Weg zur Umsetzung eines Simulationsmodells als die simultane Durchführung eines Projektes sowie eines Problemlösungsprozesses. Beide Prozesse laufen während der gesamten Projektdauer parallel und werden mit den hierzu vorhandenen Methoden gesteuert und weiterentwickelt. Da das Projektmanagement organisatorische und wirtschaftliche Ursprünge hat, wird hier ausschließlich der Problemlösungsprozeß betrachtet. Für einen analytischen Vergleich der Entwicklung einer Simulation mit der eines Testdatengenerators ist das Projektmanagement ungeeignet.

Die in Tabelle 3.1 auf Seite 45 aufgeführten Prozeßschritte lassen sich auch auf die Erstellung eines Testdatengenerators anwenden. Der erste Schritt - die Problem-Strukturierung - entspricht der Eingren-

zung auf die Modellgruppen, für die Testdaten erzeugt werden sollen. Diese kann z. B. kapazitierte Losgrößenprobleme, Bestellmengenprobleme oder Standortplanungsprobleme umfassen. Innerhalb dieser Problemgruppen ist es möglich, daß die erzeugten Daten für ähnliche Problemstellungen verwendet werden können. Die Daten, die für ein Standortplanungsproblem erzeugt wurden, lassen sich auch für verschiedene Varianten einsetzen, die z. B. unterschiedliche Transporttarife modellieren.

Der zweite Prozeßschritt umfaßt die Modellierung. Die Modellierung im Bereich der Simulation umfaßt die Identifikation aller relevanten Ereignisse und Entitäten (Pidd 2004, S. 35). Für einen Testdatengenerator bedeutet dieser Schritt, daß ausgehend von der Problemgruppe alle Ein- und Ausgabewerte gesammelt werden und die Abhängigkeiten zwischen diesen Größen ermittelt oder definiert wird. Ausgehend von dieser Datenbasis werden anschließend die Methoden entwickelt, um alle End- und Zwischengrößen zu erzeugen.

Im letzten Schritt wird die Implementierung in einer Programmiersprache durchgeführt. Hierin unterscheiden sich Simulation und Testdatengenerator erheblich. Im Bereich der Simulation gibt es heute Werkzeuge, die mit Hilfe einer graphischen Oberfläche eine intuitive Modellierung von z. B. Produktionsanlagen erlauben. Die meisten dieser graphischen Oberflächen erlauben auch eine detailliertere Einflußnahme durch Verwendung einer Programmiersprache, mit der das Verhalten des Modells gezielt beeinflußt werden kann. Die Implementierung von Testdatengeneratoren ist hingegen noch nicht mit einer graphischen Oberfläche möglich. Somit muß auf eine höhere Programmiersprache zurückgegriffen werden.

Die von Pidd (2004, S. 36) weiter aufgeführten Unterpunkte Validierung und Experiment lassen sich für einen Testdatengenerator ähnlich umsetzen. Für ein Simulationsmodell ist die Validierung der Prozeß, in dem die Übereinstimmung des Modells mit der Realität überprüft wird. Die Validierung eines Testdatengenerators verläuft ähnlich wie bei einem Simulationsmodell. Ausgehend von den Eingabewerten werden statistische Kennzahlen für die erzeugten Testdaten oder den daraus berechneten Lösungen ermittelt, die mit den realen Werten verglichen

werden können. Stimmen diese mit den erwarteten Kennzahlen überein, kann davon ausgegangen werden, daß der Generator Daten erzeugt, die mit der Realität vergleichbar sind.

Im letzten Schritt einer Simulation wird das Experiment durchgeführt, in dem verschiedene Szenarien simuliert werden. Die Ergebnisse sind das Ziel der Simulation. Für Testdatengeneratoren wird der Schritt des Experimentierens ersetzt durch die Erzeugung von Datensätzen. Das Ziel dieses letzten Schrittes ist identisch mit der Zielsetzung für den Bau eines Testdatengenerators: die Bereitstellung von Daten für die Modellierung oder Simulation.

Dieser Prozeß enthält bisher keinen Bezug zu den in den vorangehenden Abschnitten vorgestellten Designaspekten. Da diese jedoch einen entscheidenden Einfluß auf die auszuwählenden Verfahren sowie auf die Qualität haben, müssen sie entsprechend berücksichtigt werden. Die Entscheidung für eine lose oder feste Datenbindung hat Auswirkungen auf den gesamten weiteren Prozeß und sollte daher parallel zur Problemstrukturierung erfolgen. Die Entscheidung für eine implizite oder explizite Korrelation muß nicht in einer frühen Phase erfolgen, sondern kann auf einen späteren Zeitpunkt verlegt werden. Da die Korrelation Gegenstand der Untersuchung sein kann, ist es auch möglich, sie erst in einer der letzten Phasen zu berücksichtigen.

Außer dem hier betrachteten Ansatz von Pidd (2004) gibt es auch weitere Ansätze zur Durchführung einer Simulationsstudie, die auch einen größeren Detaillierungsgrad aufweisen. Als Beispiel ist hier der Ansatz von Law u. Kelton (2000, S. 84) zu nennen. Der Prozeß zur Durchführung einer Simulationsstudie besteht dabei aus zehn Schritten. Die vorgestellte Äquivalenz beider Prozesse läßt sich auch bei diesem Ansatz in fast allen Schritten nachweisen. Eine Erweiterung für die Entwicklung von Testdatengeneratoren ist somit grundsätzlich möglich. Weitere Möglichkeiten zur Darstellung der Prozesse ergeben sich aus den Standardmodellen zur Softwareentwicklung. Diese allgemein gehaltenen Ansätze lassen sich auf beide Problemstellungen anwenden. Da sich die Prozesse für Simulationsstudien von den allgemeinen Ansätzen der Softwareentwicklung ableiten, läßt sich auch der hier vorgestellte Prozeß von diesen ableiten. Auf eine weitere Vorstellung

	Testdatengenerator	Simulation
Problem-Strukturierung	Definition der Modellgruppe, für die der Testdatengenerator Daten erzeugen soll.	Erfassung des Problems und aller für die Modellierung notwendigen Problemgruppen, die modelliert werden sollen (müssen).
Modellierung	Entwicklung der Methoden, mit der die Daten erzeugt werden.	Identifikation der für die Simulation relevanten Ereignisse/Entitäten sowie Festlegung der Rahmenbedingungen, in denen die Simulation stattfindet.
Implementation	Die konkrete Umsetzung in einer Programmiersprache mit der Definition der Schnittstellen zur Ein- und Ausgabe.	Umsetzung aller Ereignisse und Entitäten innerhalb einer vorher vereinbarten Modellierungsumgebung.
Validierung	Überprüfung der gemessenen Kennzahlen mit den durch die Eingabewerte gegebenen Erwartungswerten	Vergleich des Modells mit der Realität
Experiment	Erzeugung von Datensätzen	Simulation und Analyse von Szenarien

Tabelle 3.1: Vergleich des Modellierungsansatzes von Pidd (2004, S. 32ff) mit der Erstellung eines Testdatengenerators

der allgemeinen Ansätze wird daher verzichtet und auf die Literatur verwiesen (vgl. Balzert 2008, S. 515 und Balzert 2000).

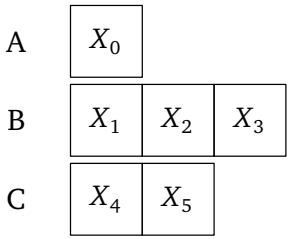
3.5 Methoden zur Datengenerierung

3.5.1 Zufallszahlengenerator mit Teilströmen

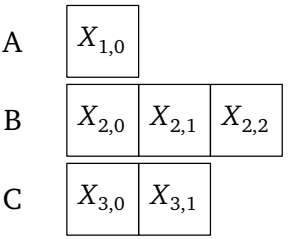
Für die Erstellung von Testdaten lassen sich die Eigenschaften eines Zufallszahlengenerators mit Teilströmen besonders gut einsetzen. Wie in Abschnitt 2.1 auf Seite 13 beschrieben, ist eine der wesentlichen Eigenschaften, daß die Teilströme nicht miteinander korreliert sind. Die Teilströme können dazu verwendet werden, Ausgabedaten unabhängig voneinander zu erzeugen. Unabhängigkeit bedeutet hier, daß die Daten nur über ihre formal definierten Abhängigkeiten beeinflußt werden und nicht über den (technischen) Erzeugungsprozeß.

Folgendes Beispiel soll dies zeigen. Abbildung 3.1 auf der nächsten Seite stellt die Unterschiede im Ablauf zwischen einem Zufallszahlengenerator mit (3.1b und 3.1d) und ohne Teilströme (3.1a und 3.1c) dar. Verbraucht werden die Zufallszahlen von drei Ausprägungen eines Merkmals mit den Bezeichnungen A, B und C. Die Anzahl der für A erzeugten Werte beeinflusst auch die Werte für B und C, falls ein konventioneller Zufallszahlengenerator eingesetzt wird (Abbildung 3.1a). Nach einer Änderung der Rahmenbedingungen benötigt A mehr Zufallszahlen, so daß sich die Werte von B und C ändern (Abbildung 3.1c). Dies ist nicht immer gewünscht, da z. B. bei einer Nachfragereihe nur die Periodendauer verlängert werden soll, ohne die bisher erzeugte Nachfrage zu verändern. Testinstanzen lassen sich so nicht mehr direkt miteinander vergleichen, da sie ja keine gemeinsamen Daten mehr besitzen.

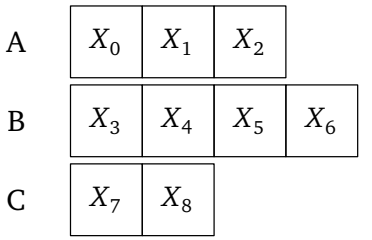
Im Gegensatz hierzu sind bei einem Zufallszahlengenerator mit Teilströmen die für A, B und C erzeugten Werte ($X_{i,j}$) unabhängig (Abbildung 3.1b). Eine Veränderung der Rahmenbedingungen hat hier keinerlei Auswirkungen (Abbildung 3.1d). Der einzige Fall, bei dem sich die Zufallszahlen gegenüber der Ausgangssituation verändern kön-



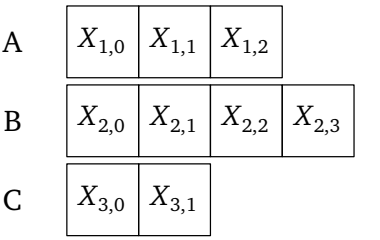
(a) Ausgangssituation bei einem konventionellen Zufallszahlengenerator



(b) Ausgangssituation bei einem Zufallszahlengenerator mit Teilströmen



(c) Veränderung durch zusätzliche Ziehungen bei einem konventionellen Zufallszahlengenerator



(d) Veränderung durch zusätzliche Ziehungen bei einem Zufallszahlengenerator mit Teilströmen

Abbildung 3.1: Verwendung von Zufallszahlen (X_i) in einem Modell mit Teilströmen ($X_{i,j}$) und ohne Teilströme

nen, besteht darin ein Produkt zu entfernen oder allgemein die Zuweisung der Teilströme an die Ausprägungen der Merkmale oder die Merkmale zu verändern. Veränderungen der Anzahl an Ausprägungen eines Merkmals haben immer so zu erfolgen, daß die Zuweisung der Teilströme an die Ausprägungen nicht verändert wird. Auf das Beispiel übertragen heißt dies, daß Ausprägung C entfernt wird, wenn weniger Ausprägungen gefordert sind.

3.5.2 Verteilung von Zufallszahlen auf einem n-dimensionalen Datenfeld

In vielen Modellen sind Daten vorhanden, deren Struktur sich durch ein n-dimensionales Datenfeld beschreiben läßt. Um dieses Feld mit Zufallszahlen zu füllen, gibt es zwei Verfahren:

- Jede Dimension des Feldes bekommt eine Ebene des Zufallszahlengenerators zugewiesen. Die Indexierung im Feld entspricht der Position im Teilstrom bezogen auf den übergeordneten Teilstrom.
- Ein einziger Strom von Zufallszahlen wird auf ein Datenfeld verteilt. Die maximale Anzahl an Zufallszahlen je Feldeintrag muß vorher definiert sein. Die Dimensionen des Feldes sind festgelegt, so daß sie der maximalen Anzahl an Ausprägungen entsprechen.

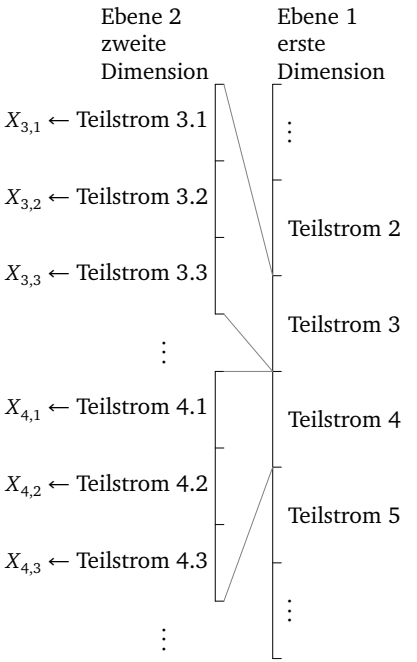
Im ersten Fall steigt die Anzahl an Ebenen mit der Anzahl an Dimensionen. Die Position im Feld wird durch die Indizes auf die Adressierung der Teilströme abgebildet, vgl. Abbildung 3.2 auf der gegenüberliegenden Seite. Der Teilstrom der untersten Ebene wird genutzt, um den Eintrag im Feld zu berechnen. Die Länge des Teilstroms muß dabei so gewählt werden, daß durch die Berechnung des Feldeintrages nicht mehr Zufallszahlen verbraucht werden, als zur Verfügung stehen. Der Verbrauch hängt von der Art des Verfahrens ab und kann deterministisch (Inversionsmethode, z. B. Gleichverteilung, diskrete Verteilung) oder nicht deterministisch (Acceptance-Rejection-Verfahren, z. B. Gammaverteilung) sein. Bei nicht deterministischen Verfahren sollte die Länge so groß gewählt werden, daß eine Überschreitung nur mit einer sehr geringen Wahrscheinlichkeit auftreten kann.

Im zweiten Fall wird die Position im Zufallszahlenstrom anhand der maximalen Dimensionen und der Anzahl der Zufallszahlen je Feldeintrag berechnet und anschließend aus dem Strom ausgelesen. Für die Anzahl an Zufallszahlen je Feldeintrag gilt das Gleiche, wie für das erste Verfahren.

Das erste Verfahren ist geeignet, wenn bei der Konstruktion des Testdatengenerators noch nicht abschließend feststeht, wie groß die

	1	2	3	4	5	6
1	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$X_{1,5}$	$X_{1,6}$
2	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$X_{2,5}$	$X_{2,6}$
3	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$X_{3,5}$	$X_{3,6}$
4	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$X_{4,5}$	$X_{4,6}$
5	$X_{5,1}$	$X_{5,2}$	$X_{5,3}$	$X_{5,4}$	$X_{5,5}$	$X_{5,6}$
6	$X_{6,1}$	$X_{6,2}$	$X_{6,3}$	$X_{6,4}$	$X_{6,5}$	$X_{6,6}$

(a) Zweidimensionales Datenfeld



(b) Zuweisung der Teilströme an die Einträge im Datenfeld

Abbildung 3.2: Verteilung von Zufallszahlen auf einem zweidimensionalen Datenfeld mit Teilströmen

	A	B	C	D	E	F
A	R_0	R_{25}	R_{50}	R_{75}	R_{100}	R_{125}
B	R_{150}	R_{175}	R_{200}	R_{225}	R_{250}	R_{275}
C	R_{300}	R_{325}	R_{350}	R_{375}	R_{400}	R_{425}
D	R_{450}	R_{475}	R_{500}	R_{525}	R_{550}	R_{575}
E	R_{600}	R_{625}	R_{650}	R_{675}	R_{700}	R_{725}
F	R_{750}	R_{775}	R_{800}	R_{825}	R_{850}	R_{875}

(a) Rüstmatrix mit drei aktiven Produkten

	A	B	C	D	E	F
A	R_0	R_{25}	R_{50}	R_{75}	R_{100}	R_{125}
B	R_{150}	R_{175}	R_{200}	R_{225}	R_{250}	R_{275}
C	R_{300}	R_{325}	R_{350}	R_{375}	R_{400}	R_{425}
D	R_{450}	R_{475}	R_{500}	R_{525}	R_{550}	R_{575}
E	R_{600}	R_{625}	R_{650}	R_{675}	R_{700}	R_{725}
F	R_{750}	R_{775}	R_{800}	R_{825}	R_{850}	R_{875}

(b) Erweiterung der Rüstmatrix um ein viertes Produkt

Abbildung 3.3: Verteilung von Zufallszahlen auf einem zweidimensionalen Datenfeld mit einem Zufallszahlenstrom

einzelnen Dimensionen werden oder welches Verfahren zur Berechnung der Feldeinträge eingesetzt werden soll. Die Grenzen werden hier nur durch die Anzahl und Länge der Teilströme gesetzt, die wesentlich höher sind, als im zweiten Verfahren (vgl. z. B. Abbildung 2.1 auf Seite 18).

Falls die Dimensionen und das Verfahren zur Berechnung der Feldeinträge bereits vorher bekannt sind, empfiehlt sich die Anwendung des zweiten Verfahrens. Speziell bei einer fest vorgegebenen (geringen) Anzahl an Zufallszahlen je Feldeintrag (z. B. einer für die Gleichverteilung) benötigt dieses Verfahren weniger Zufallzahlen.

Ein Beispiel für die Anwendung des zweiten Verfahrens bei der Erstellung einer Rüstmatrix ist in Abbildung 3.3 zu sehen. Der Testdatengenerator beschränkt die Anzahl an Produkten auf A bis F, die Anzahl der Felddimensionen beträgt 2. Der Zufallszahlenstrom R startet mit R_0 für die Kombination A-A. Für jeden Eintrag im Datenfeld wurden 25 Zufallszahlen reserviert, so daß $6 \cdot 6 \cdot 25 = 900$ Zufallszahlen vom Strom verbraucht werden.

Das zweite Verfahren ist ein Sonderfall des ersten Verfahrens. Beide Verfahren sind identisch, wenn die Anzahl der Dimensionen der Anzahl an Ebenen entspricht und die Größe jeder Dimension mit der Anzahl an direkt untergeordneten Teilströmen der jeweiligen Ebene übereinstimmt. Die Unterschiede ergeben sich in der Implementierung und Verwendung.

Beide Verfahren eignen sich zur Erzeugung von Daten mit den Eigenschaften der „common random numbers“. Diese Eigenschaft bleibt erhalten, auch wenn bei einer anderen Konfiguration des Testdatengenerators die Größe der Dimensionen verändert wird. Voraussetzung ist, daß der Prozeß der Zuweisung von Teilströmen/Zufallszahlen an die Dimensionen erhalten bleibt. Eine weitere Einschränkung ist die Größe der Dimensionen. Diese darf die vom Generator vorgegebenen Grenzen nicht überschreiten.

Abbildung 3.3 auf der vorherigen Seite zeigt ein Beispiel für eine Erweiterung, wenn ein Zufallszahlenstrom eingesetzt wird (die Zuweisung mit Teilströmen verläuft analog). Eine Konfiguration des Testdatengenerators verwendet drei Produkte A,B und D (vgl. Abbildung 3.3a). Die 3×3 Matrix wird aus den hervorgehobenen Feldern zusammengesetzt. Der Testdatengenerator beschränkt die maximale Größe auf eine 6×6 Matrix (in der Abbildung grau hinterlegt). Die Erweiterung um ein viertes Produkt F in Abbildung 3.3b führt zu keiner Veränderung der Rüstzeiten der anderen drei Produkte. Das vierte Produkt F konnte hinzugefügt werden, obwohl es an sechster Stelle steht. Solange die gleiche Abbildung der Produkte auf die Indizes der Matrix eingesetzt wird, können beliebige Kombinationen an Produkten erzeugt werden.

3.5.3 Variation der Ausprägungen mehrerer Merkmale

Im Rahmen eines computergestützten Experimentes werden die Einflüsse eines oder mehrerer Merkmale auf verschiedene Kennzahlen untersucht. Einfachster Fall ist die Variation eines Merkmals, die hier nicht weiter erläutert wird. Sollen hingegen mehrere Merkmale variiert werden, so gibt es unterschiedliche Verfahren, um systematisch Testinstanzen zu erzeugen, die für spätere statistische Untersuchungen

geeignet sind. Die hier vorgestellten Verfahren sind identisch zu den in der Simulation benutzten Methoden. Ein Überblick über verschiedene Verfahren und ihre Anwendungen wird in Law u. Kelton (2000, S. 622) gegeben.

Voll-Faktorielles-Design

Die Untersuchung eines Modells mit Hilfe eines Voll-Faktoriellen-Designs von k-Faktoren umfaßt die Kombination der Ausprägungen eines Merkmals mit allen Ausprägungen der anderen Merkmale. Die Menge der Testinstanzen wird durch die vollständige Enumeration der Ausprägungen aller Merkmale erzeugt. Die Anzahl der erzeugten Testinstanzen entspricht dem Produkt aus der Anzahl der Ausprägungen jedes Merkmals.

Zur Untersuchung der Einflüsse von einzelnen Merkmalen wird das Design der Testdaten ausgenutzt. Die Testinstanzen werden in Untersuchungsgruppen zusammengefaßt und auf den ermittelten Kennzahlen statistische Signifikanztest durchgeführt. In Abbildung 3.4 auf der gegenüberliegenden Seite sind drei Merkmale dargestellt, die als Eingabegrößen einer Testreihe dienen. Die Anzahl an Kombinationen ergibt sich zu $4 \cdot 3 \cdot 3 = 36$.

Für eine Untersuchung der Unterschiede in Merkmal 2 läßt sich die Menge aller Kombinationen in drei disjunkte Teilmengen aufspalten, die jeweils 12 Elemente enthalten M_1={(a,1,w), (a,1,x) ... (d,1,y)}, M_2={(a,2,w), (a,2,x) ... (d,2,y)} und M_3={(a,3,w), (a,3,x) ... (d,3,y)}. Die Kennzahlen jeder Teilmenge lassen sich mit den Kennzahlen der anderen Teilmengen vergleichen, so daß Aussagen zur Wirkung von Merkmal 2 getroffen werden können.

Wenn die Ausprägungen des untersuchten Merkmals unabhängig von den Werten der anderen Merkmale sind, also eine feste Bindung vorliegt, können durch spezielle statistische Verfahren die Konfidenzintervalle verringert werden. Entsprechende Verfahren werden unter dem Begriff der „common random numbers“ von Pidd (2004, S. 222), Kelton (2006, S. 186), Law u. Kelton (2000, S. 581) und Fishman (2001, S. 312) vorgestellt.

Merkmal	Elemente
1	{a,b,c,d}
2	{1,2,3}
3	{w,x,y}

(a) Ausprägungen

Menge aller Kombinationen =
{(a,1,w), (a,1,x), ...,
(a,2,w), (a,2,x), ...,
(b,1,w), (b,1,x), ... , (d,3,y)}

(b) Vollständige Enumeration

Abbildung 3.4: Beispiel zur vollständigen Enumeration von Merkmalen

Ebenenmodell - Common Random Numbers

In Ergänzung zu den bisher vorgestellten Konzepten der Bindung von Daten an Ausprägungen von Merkmalen gibt es auch die Möglichkeit, ein Ebenenmodell zur Erzeugung von Daten zu verwenden. Die Merkmale werden auf Ebenen verteilt, die den Ebenen der Teilströme eines Zufallszahlengenerators entsprechen. Die Verteilung erfolgt anhand der gewünschten Variabilität. Merkmale, die weniger variiert werden sollen, werden auf einer höheren Ebene angeordnet als Merkmale, die häufiger variiert werden sollen. Merkmale auf höheren Ebenen werden als „strukturelle Merkmale“ bezeichnet, Merkmale auf niedrigeren Ebenen sind hingegen „veränderliche Merkmale“. Durch die Verwendung von unabhängigen Teilströmen ergibt sich eine Abhängigkeit zwischen den Ausprägungen eines veränderlichen Merkmals und der Ausprägung der übergeordneten strukturellen Merkmale (vgl. Abbildung 3.5 auf der nächsten Seite). Da die Teilströme jedoch nicht korreliert sind, ist dies eine Abhängigkeit, die nur aufgrund der Konstruktion besteht und keine Auswirkungen auf die Qualität der erzeugten Daten hat.

Ein weiterer wichtiger Punkt ist die Vermeidung von Überschneidungen in der Auswahl von Teilströmen aus unterschiedlichen Ebenen. Wird ein Merkmal einer unteren Ebene zugewiesen, so muß sichergestellt sein, daß kein Teilstrom der darüberliegenden Ebenen auf diese Teilströme der unteren Ebene zugreift. Andernfalls würden zwei oder mehrere Merkmale die gleichen Zufallszahlen als Quelle für die Berechnung einzelner Ausprägungen benutzen. Somit würde eine Korrelation von

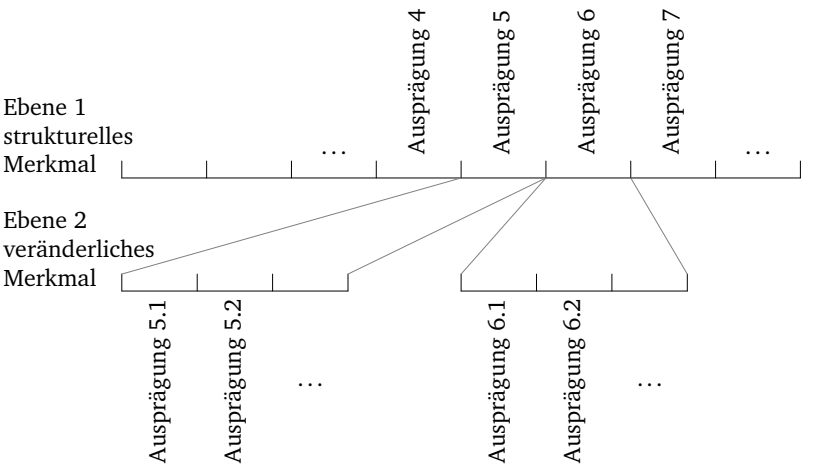


Abbildung 3.5: Aufbau eines Ebenenmodells

vorher unabhängigen Merkmalen entstehen. In der Implementierung muß somit für veränderliche Merkmale von unteren Ebenen jeweils ein eigenes aggregiertes Stellvertretermerkmal auf den darüberliegenden Ebenen eingesetzt werden. Die Aggregation wird nach oben fortgesetzt, bis die Ebene unmittelbar unterhalb der strukturellen Merkmale erreicht ist.

Umgekehrt können aber auch Merkmale aus einer höheren Ebene die ihnen zugeteilten unteren Ebenen zur strukturierten Erzeugung von Testdaten benutzen. Veränderungen der Anzahl an Ausprägungen eines Merkmals haben so keinen Einfluß auf die Ausprägungen des Merkmals. Ein Beispiel hierfür wird in Abschnitt 7.2.1 auf Seite 254 vorgestellt.

Die Anwendung des Ebenenmodells ist für beide Arten der Datenbindung möglich und bietet den Vorteil, daß größere und kleinere Testinstanzen auch anhand der erzeugten Daten verglichen werden können. Ein weiterer Vorteil des Ebenenmodells ist die Möglichkeit, Testdatensätze zu vergrößern. Durch die unabhängigen Teilströme können mehr Zufallszahlen gezogen werden, ohne daß die Ausprägungen anderer Merkmale direkt beeinflußt werden (vgl. Abbildung 3.1 auf Seite 47). Der größere Testdatensatz baut somit auf dem kleineren

Datensatz auf. Einzelne Merkmale enthalten die Ausprägungen des kleineren Datensatzes als Untermenge, so daß direkte Vergleiche zwischen beiden Datensätzen möglich sind.

3.5.4 Ziehen ohne Zurücklegen

Zur Auswahl von n Elementen aus einer Grundgesamtheit von m unterschiedlichen Elementen wird der Algorithmus aus Abbildung 3.6 auf der nächsten Seite verwendet.

Der Algorithmus entfernt jedes gezogene Element aus der Grundgesamtheit, so daß diese kleiner wird. So ist sichergestellt, daß kein Element doppelt gezogen werden kann. Ein alternatives Verfahren, welches häufig eingesetzt wird, beläßt das Element in der Grundgesamtheit und überprüft bei jedem neu zu ziehenden Element, ob es schon gezogen wurde und wiederholt in diesem Fall die Ziehung. Dies wirkt sich nachteilig aus, wenn der Anteil der zu ziehenden Elemente im Vergleich zur Grundgesamtheit groß ist, da es häufiger zu Kollisionen kommt und die Ziehung erneut durchgeführt werden muß.

3.5.5 Stochastische Verteilungen als Eingabedaten

Ein Teil der Daten, die von einem Testdatengenerator erzeugt werden, basieren auf Stichproben, die aus einer stochastischen Verteilung gezogen werden. Die Art der stochastischen Verteilung und die charakteristischen Merkmale stellen die Ausprägungen eines Merkmals dar, daß ein Eingabeparameter des Testdatengenerators ist. Innerhalb des Testdatengenerators kann es nötig sein, verschiedene Operationen auf dieser stochastischen Verteilung durchzuführen.

Grundoperationen mit stochastischen Verteilungen

Es gibt zwei Grundoperationen, die auf einer stochastischen Verteilung durchgeführt werden können: Verschiebung um einen Wert $b \in \mathbb{R}$ und Skalierung mit einem Faktor $a \in \mathbb{R}$. Zusammenfassen läßt sich dies als eine lineare Transformation der Zufallsvariablen X auf $a \cdot X + b$. Der

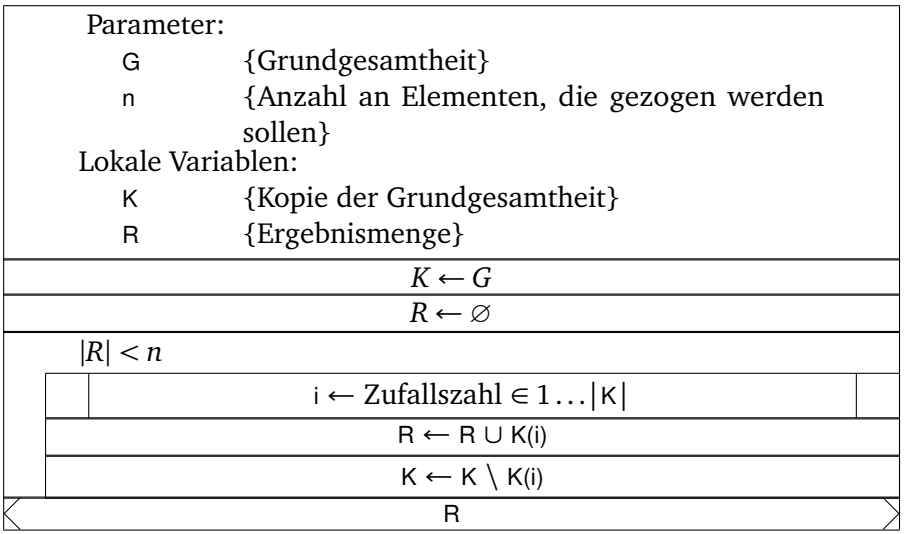


Abbildung 3.6: Algorithmus „Ziehen ohne Zurücklegen“

Erwartungswert $E(X)$ wird transformiert zu $a \cdot E(x) + b$. Für die Varianz gilt folgende Beziehung $Var(a \cdot X + b) = a^2 \cdot Var(X)$ (Heike u. Târcolea 2000, S. 303).

In vielen Fällen wird für ein Merkmal nur die Art der Verteilung vorgegeben, sowie der Variationskoeffizient (dieser ist definiert als der Quotient aus der Standardabweichung und dem Mittelwert, vgl. Hartung 2002, S. 117). Die Auswirkungen einer linearen Transformation lassen sich durch folgende Rechnung zeigen:

$$V = \frac{s}{\overline{x}} = \frac{\sqrt{Var(a \cdot X + b)}}{E(a \cdot X + b)} = \frac{\sqrt{a^2 \cdot Var(X)}}{a \cdot E(X) + b}$$

$$= \begin{cases} \frac{\sqrt{Var(X)}}{E(X)} & b = 0 \\ \frac{a \cdot \sqrt{Var(X)}}{a \cdot E(X) + b} & b \neq 0 \end{cases}$$

Für den Fall, daß $b = 0$ ist, bleibt der Variationskoeffizient erhalten. Somit läßt sich jede stochastische Verteilung mit einem Faktor $a \in \mathbb{R} \setminus 0$ skalieren, ohne den Variationskoeffizienten zu verändern. Sobald jedoch eine Verschiebung um einen Wert $b \neq 0$ vorgenommen wird, verändert sich der Variationskoeffizient. Daraus ergibt sich für die Anwendung in einem Testdatengenerator folgender Satz:

Ist der Variationskoeffizient für eine Verteilung vorgegeben, so darf die Verteilung nur skaliert werden. Ist hingegen die Varianz als absoluter Wert vorgegeben, so sind nur Verschiebungen zulässig.

Erfüllung von Nebenbedingungen

Mathematische Modelle zur linearen Programmierung bestehen aus Ungleichungen, die Restriktionen darstellen und so den Bereich der zulässigen Lösungen einschränken. Diese Restriktionen sind als Linearkombinationen von Variablen modelliert. Zur Erzeugung von zulässigen Testdaten ist es notwendig, daß diese Restriktionen bereits bei der Konstruktion eingehalten werden. Wenn aber gleichzeitig eine Teilmenge dieser Variablen als Stichprobe aus einer stochastischen Verteilung stammen soll, muß sichergestellt sein, daß auch diese Stichprobe die Restriktionen einhält.

In einem gemischt-ganzzahligen Modell lassen sich alle Nebenbedingungen auf die Form $\sum_i a_i \cdot x_i \leq c \quad a_i, x_i \in \mathbb{R} \forall\, i, c \in \mathbb{R}$ zurückführen (Hillier u. Lieberman 2007, S. 33). Bei der Konstruktion einer Testinstanz werden entweder die Koeffizienten a_i oder die Variablen x_i aus einer stochastischen Verteilung gezogen. Der folgende Algorithmus berücksichtigt dies, indem er die Werte der betroffenen Variablen entsprechend anpaßt und garantiert, daß der stochastische Charakter der Variablen erhalten bleibt. Gleichzeitig achtet er auf die Kapazitätsrestriktion c, die nicht verletzt werden darf.

Parameter: X {Menge der Variablen x_i} A {Menge der Koeffizienten a_i} c {konstanter Wert} Lokale Variablen: k {Korrekturfaktor}
$C \leftarrow \sum_i a_i \cdot x_i$
$k \leftarrow \frac{c}{C}$
$x_i \leftarrow k \cdot x_i \; \forall i$

Abbildung 3.7: Algorithmus „Erfüllung von Nebenbedingungen“

Abbildung 3.7 enthält den Transformationsalgorithmus. Die Transformation entspricht einer Skalierung, wie sie im vorangehenden Abschnitt vorgestellt wurde. Damit sind auch gleichzeitig die Eingabegrößen definiert. Nur die Art der Verteilung und der Variationskoeffizient dürfen vorgegeben sein, da diese als einzige nach der Transformation erhalten bleiben. Als weitere Einschränkung ist die Transformation nur auf eine Nebenbedingung anwendbar. Der Algorithmus unterstützt nicht die Kopplung mehrerer Nebenbedingungen.

3.6 Implementierung

Die Implementierung des Testdatengenerators erfolgt wie auch der Zufallszahlengenerator und die Simulationssoftware in der Programmiersprache Java. Speziell für den Bereich der Erzeugung von Testdaten und zum Teil auch für die Simulation wurden mehrere Klassen entwickelt, die allgemein in Testdatengeneratoren eingesetzt werden können. Der hier verwendete Zufallszahlengenerator wurde bereits in Abschnitt 2.3 auf Seite 18 vorgestellt. Er ist ebenfalls so allgemein gehalten, daß er unabhängig vom Testdatengenerator eingesetzt werden kann.

3.6.1 Stochastische Verteilungen

Stochastische Verteilungen sind ein essentieller Bestandteil von Testdatengeneratoren und Simulationen, da sie zur Erzeugung der zufälligen Ausprägungen eines oder mehrerer Merkmale dienen. Für jede Verteilung existieren ein oder mehrere Algorithmen, um entsprechend verteilte Zufallszahlen zu erzeugen. Als Eingabewerte werden die Ausgabewerte eines Standard-Zufallszahlengenerators (z. B. der Generator aus Abschnitt 2.3) verwendet. Dieser erzeugt Zahlen aus einer $[0;1]$ Gleichverteilung, so daß der Algorithmus die Transformation auf die gewünschte Verteilung durchführt. Die folgenden Gruppen stellen einen Überblick der am meisten eingesetzten Verfahren dar. Beispiele zu den hier vorgestellten Gruppen finden sich in Banks u. a. (2000, S. 272), Law u. Kelton (2000, S. 437) und in Pidd (2004, S. 193).

Berechnung der Umkehrfunktion Existiert eine geschlossene mathematische Form für die Beschreibung der Umkehrfunktion einer Verteilung, so kann durch Einsetzen der gleichverteilten Zufallszahlen eine Zufallszahl aus dieser Verteilung gewonnen werden.

Acceptance-Rejection Diese Gruppe von Verfahren umfaßt alle Algorithmen, bei denen durch das Ziehen von einer oder mehrerer gleichverteilter Zufallszahlen ein Abbruchkriterium erreicht werden soll. Aus der Menge an gezogenen Zufallszahlen wird mit Hilfe einer Transformation die Zufallszahl der gewünschten Verteilung gewonnen. Typische Anwendungen sind Verteilungen, bei denen keine geschlossene Form der Umkehrfunktion bekannt ist.

Faltung Die Faltung wird angewendet, wenn eine Verteilung durch eine Linearkombinationen anderer Verteilungen dargestellt werden kann. Ein Beispiel ist die Erlang-Verteilung, die sich auf exponential verteilte Zufallszahlen zurückführen läßt.

Andere Verfahren Läßt sich ein Verfahren nicht den vorangehenden Gruppen zuordnen, so wird es hier eingeordnet. Entsprechend gibt es hier keine homogene Menge an Verfahren, sondern Sonderfälle,

die für eine Verteilung oder nur für eine Parameterkombination einer Verteilung zutreffend sind.

Einen Überblick der für die verschiedenen Verteilungen eingesetzten Algorithmen geben Law u. Kelton (2000, S. 471). Weitere Verfahren, wie z. B. korrelierte Zufallszahlen, die hier nicht weiter betrachtet werden, finden sich in Law u. Kelton (2000, S. 478). Von den in dieser Arbeit verwendeten Verteilungen werden die ersten beiden Verfahren am häufigsten eingesetzt. Die Berechnung der Umkehrfunktion wird z. B. für empirische Verteilungen sowie die Exponentialverteilung eingesetzt. Acceptance-Rejection-Verfahren kommen bei der Gammaverteilung sowie der Normalverteilung zum Einsatz (Law u. Kelton 2000, S. 465). Im folgenden Abschnitt wird die Betaverteilung vorgestellt, die durch eine Transformation aus zwei Gammaverteilungen erzeugt wird und somit zur letzten Gruppe gehört. Nicht kontinuierliche Verteilungen (wie z. B. die Bernoulliverteilung) werden im Rahmen dieser Arbeit nicht eingesetzt, da auf sie die in Abschnitt 3.5.5 vorgestellten Transformationen nicht allgemein einsetzbar sind.

Klassenstruktur

Die Klassenstruktur folgt einem Muster, wie es von Bloch (2008, S. 93) beschrieben wird. Basierend auf der Schnittstelle *RealDist* wurde eine abstrakte Referenzimplementierung *AbstractDistribution* erstellt (vgl. Abbildung 3.8 auf der nächsten Seite). Diese beinhaltet die gesamte Infrastruktur für die Kommunikation mit dem Zufallszahlengenerator, so daß die abgeleiteten Klassen hierauf zurückgreifen können und nur die Transformation von der Gleichverteilung auf eine spezifische Verteilung durchführen. Im Gegensatz zur Implementierung in DESMO-J, sowie im SSJ-Framework erfolgt hier eine Typisierung der Schnittstelle über den Generics Mechanismus, um den Charakter der Verteilung zu definieren. Dies hat den Vorteil, daß nur eine Schnittstelle für alle Verteilungen verwendet werden kann und keine zusätzlichen Schnittstellen für jeden Zahlenraum (reelle Zahlen, ganze Zahlen oder nur 0,1 Werte) notwendig sind. Die entsprechenden Verteilungen (z. B. Normalvertei-

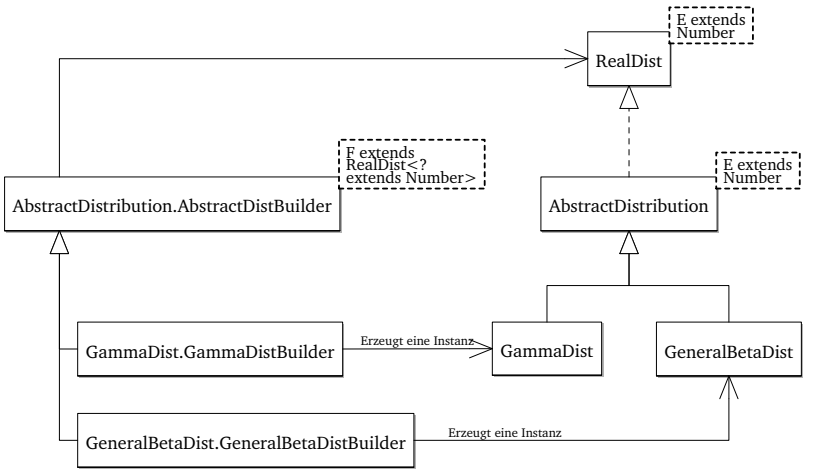


Abbildung 3.8: Klassenhierarchie der stochastischen Verteilungen

lung, Binomialverteilung oder Bernoulliverteilung) implementieren die Schnittstellen und legen damit auch gleichzeitig den Typ fest. Ein weiterer Vorteil ergibt sich bei allen Methoden, die Werte aus der Verteilung zurückliefern (z. B. eine Stichprobe oder das Maximum, Minimum). Die Rückgabewerte dieser Methoden werden durch die Typisierung ebenfalls festgelegt. Zusätzliche Abfragen nach dem Typ der Verteilung entfallen somit vollständig, da bereits während der Compilierung der Typ definiert ist.

Die Erzeugung von Instanzen einer Verteilung erfolgt über das Builder-Pattern (vgl. Gamma u. a. 1995, S. 97, Bloch 2008, S. 11). Dieses Pattern ist für Klassen mit mehreren Parametern geeignet, wurde hier jedoch eingesetzt, um in der Simulation eine späte Bindung einer Verteilung an ein Modell zu erreichen (vgl. Abschnitt 4.4). Die Builder basieren alle auf einer abstrakten Oberklasse *AbstractDistBuilder*, die als statische lokale Klasse von *AbstractDistribution* implementiert wird, um die Kapselung des Konstruktors zu gewährleisten. Von dieser Klasse wird für jede Verteilung ein Nachkommen gemeinsam mit der Verteilung implementiert, der sowohl die notwendigen Parameter, als auch

die optionalen Parameter übernimmt. Auch die Builder sind typisiert (*F extends RealDist<? extends Number>*), so daß der Rückgabewert durch die entsprechende Typisierung bei der Implementierung festgelegt wird.

Die in Abschnitt 3.5.5 auf Seite 55 vorgestellten Methoden wurden in der Oberklasse der Builder *AbstractDistBuilder* als abstrakte Methoden implementiert, die von allen Nachkommen überschrieben werden müssen. Die Ausführung der Transformationen ist somit im Builder vor der Erzeugung möglich und nicht mehr, nachdem eine Instanz der Verteilung erzeugt wurde.

Zusätzlich zu den Builder-Klassen wurden auch statische Factory-Methoden in jeder Klasse angelegt, um unterschiedliche Wege der Initialisierung zu ermöglichen. Soweit es möglich war, wurde für jede Verteilung eine Initialisierung anhand der Form- und Lageparameter, als auch anhand der charakteristischen Werte wie Mittelwert, Varianz sowie obere und untere Grenze implementiert. In Zusammenarbeit mit den Builder-Klassen lassen sich so aus einer bestehenden Verteilung die in Abschnitt 3.5.5 auf Seite 55 beschriebenen Transformationen ausführen, um einen angepaßte Verteilung zu erhalten.

Im Folgenden werden einige Verteilungen vorgestellt, bei denen die Implementierung von DESMO-J abweicht.

Gammaverteilung Zur Erzeugung von Zufallszahlen, die aus einer Gammaverteilung stammen, wurde eine Vielzahl von Algorithmen entwickelt (vgl. Ahrens u. Dieter 1982, 1988; Best 1983; Fishman 1976; Wallace 1974), die meist auf einer Zweiteilung der Erzeugung basieren. Es wird ein Algorithmus für $\alpha > 1$ und ein weiterer für $\alpha \leq 1$ benötigt. Implementiert wurde der Algorithmus von Tadikamalla (1978) für $\alpha > 1$ sowie der Algorithmus von Best (1983) für $\alpha \leq 1$.

Zur Erzeugung der Builder stehen zwei Factory-Methoden zur Verfügung - eine, die als Eingabe die Parameter α und β der Verteilung übernimmt, sowie eine weitere, welche den Mittelwert μ und die Varianz σ^2 akzeptiert. Zur Ermittlung der Parameter α und β wird folgende Beziehung verwendet (Heike u. Târcolea 2000, S. 448):

$$\begin{aligned} \mu &= \alpha\beta \\ \sigma^2 &= \alpha\beta^2 \end{aligned} \quad \Rightarrow \quad \begin{aligned} \alpha &= \frac{\mu^2}{\sigma^2} \\ \beta &= \frac{\sigma^2}{\mu} \end{aligned}$$

Betaverteilung Die Erzeugung von Zufallszahlen, die einer Betaverteilung entsprechen, kann auf den Einsatz von zwei Gammaverteilungen mit angepaßten Parametern reduziert werden (vgl. Fishman 2001, S. 375 und Law u. Kelton 2000, S. 467). Außer der Betaverteilung, die auf das Intervall $[0,1]$ beschränkt ist, wurde auch die verallgemeinerte Form implementiert, die auf einem Intervall $[a,b]$ definiert ist. Diese Form ergibt sich aus einer Lineartransformation der einfachen Betaverteilung, so daß im Folgenden nur noch die allgemeine Form betrachtet wird.

Für die Betaverteilung werden zwei Factory-Methoden bereitgestellt - eine akzeptiert die Parameter α und β, die Zweite den Mittelwert μ und die Varianz σ^2. Durch Umformung ergibt sich folgende Beziehung zwischen den Parametern:

$$\begin{aligned} \mu &= \frac{\alpha}{\alpha+\beta} \\ \sigma^2 &= \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \end{aligned} \quad \Rightarrow \quad \begin{aligned} \alpha &= \frac{\mu^2-\mu^3-\mu\sigma^2}{\sigma^2} \\ \beta &= \frac{\mu-2\mu^2+\mu^3-\sigma^2+\mu\sigma^2}{\sigma^2} \end{aligned} \tag{3.1}$$

Da $\alpha > 0$ und $\beta > 0$ sein muß, sind nur solche Parameterkombination von μ und σ^2 zulässig, bei denen die Zähler der Brüche auf der rechten Seite von Gleichung (3.1) ebenfalls größer als 0 sind.

Spezielle Verteilungen Um den Programmcode zu testen, wurden drei Verteilungen implementiert, die keine Zufallszahlen erzeugen, sondern deterministische Zahlenfolgen. Die einfachste Verteilung gibt immer einen konstanten Wert zurück, sie entspricht somit einer empirischen Verteilung mit nur einer Klasse, deren Wahrscheinlichkeit 100% beträgt.

Eine weitere Klasse erzeugt eine vorgegebene Zahlenfolge in zyklischer Folge. Als Variante wird die Folge nur einmal durchlaufen, so daß die Anzahl an Aufrufen beschränkt ist. Beide Klassen repräsentieren keine stochastischen Verteilungen, sondern implementieren nur die Schnittstelle *RealDist*. Für Testzwecke lassen sich so deterministische Daten verwenden, ohne daß am Programmcode Änderungen durchgeführt werden müssen.

3.6.2 Ersatz für numerische Bezeichner

In vielen Modellen zur gemischt ganzzahligen Optimierung werden die Produkte und Maschinen nicht mit eindeutigen Bezeichnern referenziert, sondern über Zahlen. Wird diese Art der Namensgebung in der Implementierung übernommen, so ergibt sich eine potentielle Fehlerquelle. Sowohl das Programm zum Lösen des Modells, als auch alle anderen Programme, die diese Daten verarbeiten, können z. B. nicht zwischen einem Produkt mit dem Namen „42“ und einer Maschine mit dem Namen „42“ unterscheiden. Um diese Fehlerquelle zu beseitigen wird empfohlen, sowohl bei der Implementierung des Modells, als auch bei allen anderen beteiligten Programmen, speziell dem Testdatengenerator, mit eindeutigen, nicht numerischen Bezeichnern zu arbeiten. Auf Seiten der Modellierungssoftware wird dies von Xpress-MP teilweise unterstützt (FairIsaac Corporation 2008, S. 16,17). So ist es möglich, nicht numerische Mengen als Indizes für Variablen oder Daten zu benutzen. Diese nicht numerischen Mengen sind jedoch nicht typsicher, da sie durch Zeichenketten repräsentiert werden und nicht durch einen eigenen Datentyp. Das Problem der identischen Bezeichner existiert somit weiterhin, jedoch wird dem Benutzer die Möglichkeit gegeben, durch eindeutige Bezeichnungen die Indexmengen voneinander abzugrenzen.

Die Repräsentation von Bezeichnern innerhalb von Java wird durch die in Abbildung 3.9 auf der gegenüberliegenden Seite dargestellte Klassenhierarchie erreicht. Die Infrastruktur wird über eine abstrakte Oberklasse bereitgestellt, von der alle spezifischen Bezeichner abgeleitet sind. Im Folgenden wird die Funktionsweise dieser Klasse genauer be-

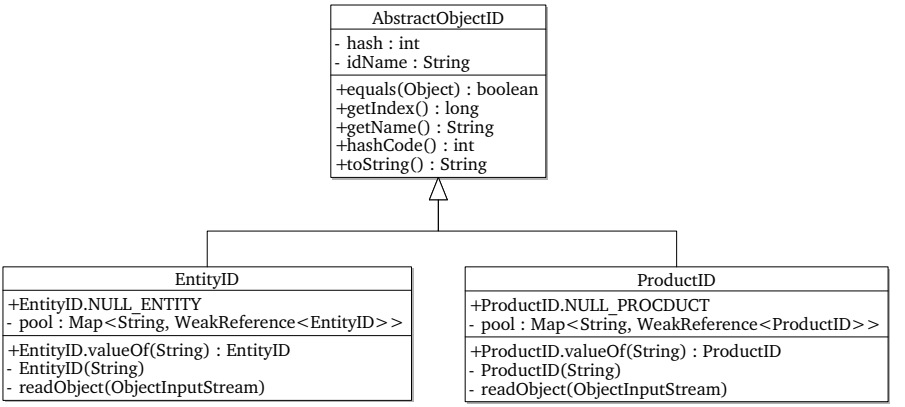


Abbildung 3.9: UML-Klassendiagramm *AbstractObjectID*

schrieben, da sie eine der wesentlichen Komponenten für die Erzeugung von Testdaten darstellt.

Die Klasse *AbstractObjectID* wurde als sogenanntes Value-Objekt erstellt (Value-Objekte werden detailliert in Abschnitt 4.3 auf Seite 84 vorgestellt). Der Wert von *AbstractObjectID* wird durch den von außen gegebenen Bezeichner in Form einer Zeichenkette (in Java ein Objekt vom Typ *String*) definiert. Da die Sichtbarkeit dieses Feldes eingeschränkt wurde, kann auch keiner der Nachkommen das Feld verändern. Innerhalb des Konstruktors erfolgt die Berechnung des Hashwertes, da sich der Wert nach der Initialisierung nicht mehr verändern kann. Der Hashwert wird in einem als *final* deklarierten Feld gespeichert. Ein ähnliches Vorgehen wird von Bloch (2008, S. 49) vorgestellt. Die Speicherung von Nachkommen dieser Klasse in einer Datenstruktur, die zur Verwaltung Hashwerte verwendet (wie z. B. eine *HashMap* oder ein *HashSet*), erfolgt mit hoher Geschwindigkeit, da nur auf das interne Feld zurückgegriffen werden muß. Die Geschwindigkeit ist vergleichbar mit der Speicherung der Objekte, welche als Wrapper um die elementaren Datentypen von Java gelegt wurden. Diese sind ebenfalls als Value-Objekte deklariert und benutzen die gleichen Mechanismen (Bloch 2008, S. 50).

Die Implementierung der *equals* und *hashCode* Methode erfolgt gemäß den Vorgaben der Spezifikation (Sun 2006, *java.lang.Object*) und den Empfehlungen von Bloch (2008, S. 33). Da das Value-Objekt den Hashwert bereits berechnet hat, kann bei der Implementierung von *equals* gleichzeitig folgende Logik zur Verringerung der Rechenzeit eingeführt werden. Es gilt die Aussage: *Wenn die Hashwerte von zwei Objekten identisch sind, so können ihre Inhalte identisch sein, sie müssen es aber nicht, da es auch zu einer Kollision des Hashwertes kommen kann* (Bloch 2008, S. 47). *Sind die Hashwerte nicht identisch, so müssen die Objekte unterschiedlich sein*. Somit kann bei unterschiedlichen Hashwerten bereits verkürzend die Ungleichheit der Objekte festgestellt werden. Da die Hashwerte als elementarer Datentyp gespeichert sind, erfolgt dieser Vergleich sehr schnell.

Eine weitere Möglichkeit zur Beschleunigung der Geschwindigkeit und zur Verringerung des Speicherbedarfs besteht in der Einführung einer statischen Factory-Methode für jede abgeleitete Klasse, welche auf einen Objekt-Pool zugreift. Der Objekt-Pool orientiert sich am Singleton-Pattern (Gamma u. a. 1995, S. 127). Die Factory-Methode wurde so entworfen, wie sie in Gamma u. a. (1995, S. 107), sowie in Bloch (2008, S. 5) beschrieben wird. Der Konstruktor wurde umgewandelt in einen privaten Konstruktor, so daß Instanzen nur durch einen Aufruf der statischen Factory-Methode erzeugt werden können. Diese hat Zugriff auf den Objekt-Pool, der alle bisher erzeugten Objekte dieses Typs enthält. Wenn für den nachgefragten Bezeichner noch kein Objekt existiert, wird eine neue Instanz erzeugt. Somit kann im gesamten Programm nur eine Instanz des Bezeichner-Objektes eingesetzt werden - dies entspricht dem *Flyweight* Pattern, wie es in Gamma u. a. (1995, S. 195) beschrieben ist. Ein Geschwindigkeitsvergleich für den Test auf Identität mit *equals* hat ergeben, daß die von *AbstractObjectID* abgeleiteten Klassen um den Faktor 2 langsamer sind als die in Java vorhandenen Wrapper-Klassen für elementaren Datentypen. Die Beschleunigung der Implementierung mit Factory gegenüber der ohne Factory liegt ebenfalls ungefähr bei Faktor 2.

Der Pool wurde so implementiert, daß intern schwache Referenzen für die Instanzen der ID-Objekte verwendet werden (*WeakReferences*).

Der Garbage-Collector ignoriert *WeakReferences*, so daß der Pool automatisch schrumpft, wenn eine Instanz außerhalb des Pools nicht mehr verwendet wird. Ein Überlaufen des Pools durch ungenutzte Referenzen wird so verhindert. Die Deserialisierung wurde entsprechend angepaßt, damit die neuen Objekte sich im Pool anmelden. Die aktuelle Implementierung enthält noch keine Überprüfungen für den Multi-Thread Einsatz. Eine entsprechende Überprüfung des Pools mit „Double Checked Locking“ ist hierfür nötig, wurde aber aufgrund der Laufzeit noch nicht implementiert.

In der Implementierung wurden weiterhin sämtliche Methoden der Klasse *AbstractObjectID* (auch *equals*, *hashCode* und *toString*) als *final* deklariert, so daß für abgeleitete Klassen keine Möglichkeit besteht, deren Verhalten zu ändern. Eine abgeleitete Klasse kann nur noch den Konstruktor überschreiben, in dem der Name des Bezeichners festgelegt wird. Die Erweiterung der abstrakten Oberklasse muß für jeden Bezeichnertyp separat erfolgen. Somit ist auch die Anforderung der Typsicherheit erfüllt. Die Vorteile ergeben sich durch den Einsatz in den Datenstrukturen bei der Konstruktion von Testdaten. Eine der Hauptanwendungen wird im folgenden Abschnitt vorgestellt.

3.6.3 Verwendung von assoziativen Arrays

In Verbindung mit der Einführung von ID-Objekten als Ersatz für Bezeichner wird auch eine Klasse benötigt, welche bei Berechnungen den Einsatz der Bezeichner in einem Testdatengenerator unterstützt. Die Klasse soll die Erzeugung und Berechnung von Testdaten so vereinfachen, daß der Benutzer auf einer Abstraktionsebene arbeitet ohne direkten Kontakt zu den Datenstrukturen. Die konkrete Zuordnung von Bezeichnern und Werten wird so verdeckt.

Für die Datenstruktur wurde eine Implementierung der Schnittstelle *Map* gewählt. Eine *Map* implementiert in Java ein assoziatives Array, welches eine allgemeine Zuordnung zwischen einem Schlüssel und einem Wert vornimmt (Sun 2006, java.util.Map). Bei der Implementierung wird der seit Java 1.5 vorhandene Generics Mechanismus verwendet, um die Typsicherheit sicherzustellen (Sun 2005, Kap. 4.4,

4.5 und 8.1.2). Als Schlüssel wird ein Nachkomme von *AbstractObjectID* festgelegt, der Wert muß ein Nachfolger der Klasse *java.lang.Number* sein. Diese dient in Java als Oberklasse für alle Wrapper der elementaren numerischen Datentypen. Für eine typsichere Verwendung der Werte werden daher sog. *Wildcard Generics* eingesetzt, vgl. Sun (2005, Kap. 4.5 und 8.1.4). Die Datenstruktur kann somit die Abbildung von Bezeichnern auf jeweils eine Zahl vornehmen. Gegenüber der klassischen Verwendung von Indizes an Stelle von Bezeichnern entspricht dies einem eindimensionalen Feld.

Die Implementierung besteht aus zwei Klassen, *MapMath* und *MapMathImpl*, die aufeinander aufbauen (vgl. Abbildung 3.10 auf der gegenüberliegenden Seite, die Typisierung wurde im UML-Diagramm nicht aufgeführt). *MapMath* ist eine Klasse, die nur statische Methoden für Rechenoperationen mit assoziativen Arrays enthält. Zentrale Methode von *MapMath* ist die Methode *operateOnMaps*. Diese Methode benutzt den von *Map* bereitgestellten Iterator über alle Einträge, um einen *NumberTransformer* auf die *Map* anzuwenden. *NumberTransformer* ist eine zu *MapMath* gehörende statische Schnittstelle, die einen Funktionszeiger nachbildet (Bloch 2008, S. 103). Alle Methoden, die eine Operation auf der gesamten *Map* durchführen, rufen *operateOnMaps* mit einer anonymen Klasse auf. Diese anonyme Klasse implementiert die *NumberTransformer* Schnittstelle und führt die Operation auf allen Elementen aus. Die Reihenfolge der Bearbeitung ist nicht definiert, so daß die Operation nur vom Wert des aktuellen Elements abhängig sein darf. Dies ist für alle Methoden der Klasse gewährleistet. Der Rückgabewert jeder Methode ist eine neue *Map*, welche eine identische Menge an Schlüsseln besitzt und jedem Schlüssel den Wert zuweist, der nach der Transformation entstanden ist.

Auf diese Weise lassen sich assoziative Arrays ähnlich wie Vektoren verarbeiten, d. h. alle unären und binären Operationen sind auf anderen Vektoren oder Skalaren durchführbar. Bei der Ausführung der binären Operationen wird immer überprüft, ob die Menge an Schlüsseln identisch ist, andernfalls ist die Operation nicht definiert. Implementiert wurden alle elementaren Grundoperationen, sowie die für den in Abschnitt 3.5 vorgestellten Testdatengenerator benötigten Operationen.

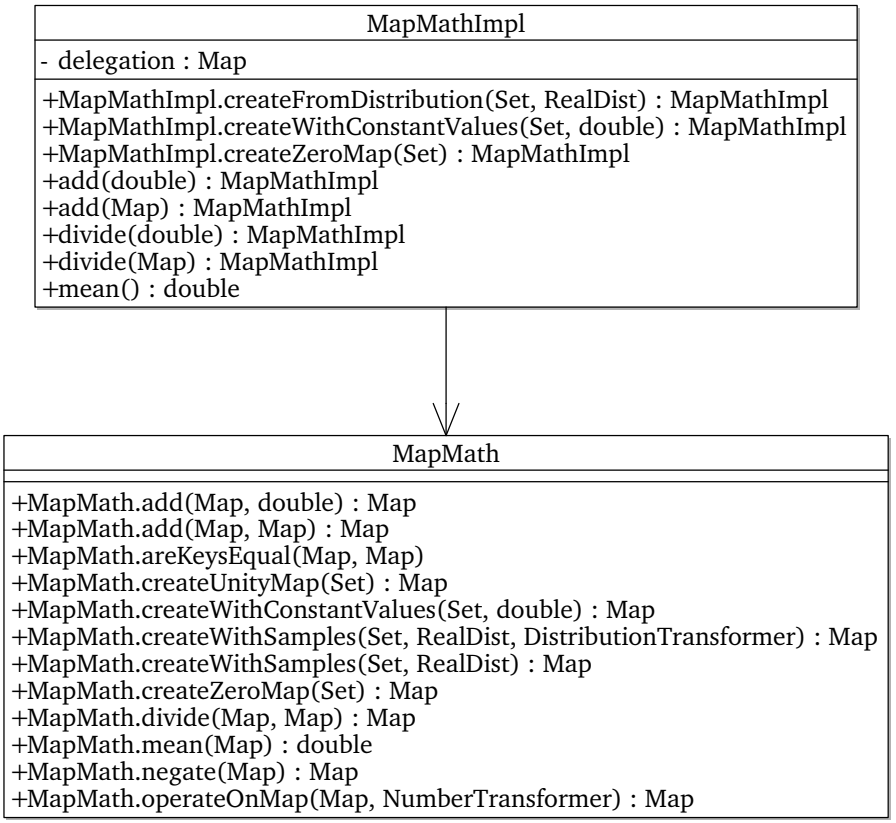


Abbildung 3.10: UML-Klassendiagramm *MapMath*

Die Nomenklatur der Grundoperationen entspricht den Methoden der Klasse *java.lang.Math*. Die Erweiterung um zusätzliche Operationen ist mit geringem Aufwand durch den Einsatz von *operateOnMaps* möglich.

Zur vereinfachten Bearbeitung dieser Datenstruktur wurde die Klasse *MapMathImpl* erstellt. Diese implementiert das *Map* Interface mit den oben beschriebenen Schlüssel- und Werttypen. Intern werden alle Aufrufe der *Map* Schnittstelle an eine *HashMap* delegiert (Fowler 1999, S. 352).

Außer den Methoden der Schnittstelle *Map* werden auch alle in *MapMath* definierten Methoden umgesetzt. Alle Methoden geben eine neue Instanz von *MapMathImpl* zurück, so daß eine Verkettung der Aufrufe zur Nachbildung von Berechnungen möglich ist. Die gleiche Technik wird auch in der Klassenbibliothek von Java eingesetzt, z. B. bei *java.lang.BigInteger* oder *java.lang.BigDecimal*.

Als Beispiel wird die Berechnung der Fixkosten betrachtet. Die entsprechende Formel lautet (vgl. Abschnitt 7.2.6 auf Seite 262)

$$A = TBO^2 \cdot \frac{h \cdot D}{2}$$

Die Abbildung auf der gegenüberliegenden Seite zeigt den entsprechenden Code in Java. Durch die Verkettung der Aufrufe entsprechend der Berechnungsreihenfolge kann auf die Erstellung von Hilfsvariablen für Zwischenergebnisse verzichtet werden, was die Anfälligkeit für Programmierfehler verringert. Der Nachteil dieser Implementierung ist die Geschwindigkeit. Da bei jedem Schritt eine neue Instanz von *MapMathImpl* angelegt wird, ist diese Berechnung langsamer und speicherintensiver, als eine speziell für diesen Fall erstellte Methode. Da die Designziele auf einer allgemein anwendbaren, leicht zu verstehenden Klasse liegen, kann die Geschwindigkeit vernachlässigt werden. Dies wird auch vom Einsatz der Klasse unterstützt, da es sich bei der Erstellung von Testdaten nicht um eine Anwendung handelt, bei der die Geschwindigkeit eine kritische Größe ist.

```
private MapMathImpl<ProductID> calcFixcosts() {
  return meanDemandRate
    .times(
      Math.pow(data.timeBetweenOrders,2d)
    )
    .times(holdingCosts)
    .divide(2d)
    .round(SIGNIFICANT_NUMBERS);
}
```

Abbildung 3.11: Codebeispiel in Java für den Einsatz von assoziativen Arrays

Teil II

Simulation kontinuierlicher Materialflüsse

Kapitel 4

Ebene 0 - Simulationskern

Einer der Hauptbestandteile dieser Arbeit ist die Entwicklung von Algorithmen zur effizienten Simulation kontinuierlicher Prozesse. Die folgenden beiden Kapitel stellen die Software und die Neuentwicklungen vor. Im Folgenden wird von einem Ebenenmodell ausgegangen, das in Tabelle 4.1 dargestellt ist.

Das Ebenenmodell besteht aus vier aufeinander aufbauenden Schichten. Die unterste Schicht (Ebene 0) enthält den Kern der Simulation, der nur aus einer allgemeinen, diskreten ereignisorientierten Simulationsbibliothek besteht. Diese Ebene ist auch Gegenstand dieses Kapitels. Auf dem Kern baut die erste Ebene auf, die alle Elemente zur Simulation von kontinuierlichen Materialflüssen in einer Produktion enthält. Die auf der ersten Ebene eingesetzten Algorithmen werden in Kapitel 5 detail-

Ebene	Bezeichnung	Bestandteile
3	Planung	Planungsalgorithmen
2	Steuerung	Fertigungsleitstand
1	Materialfluß	Bausteine der Produktion
0	Simulationskern	diskrete ereignisorientierte Simulation

Tabelle 4.1: Ebenenmodell

liert vorgestellt. Die darüberliegende zweite Ebene hat direkten Zugriff auf die Elemente der ersten Ebenen (z. B. Ressourcen) und verwaltet die Produktionspläne, die von der dritten Ebenen bereitgestellt werden. Eine weitere Aufgabe der zweiten Ebene ist die Bereitstellung von Informationen über die Maschinen, Aufträge und Lagerbestände. Oberhalb der zweiten Ebene werden diese Informationen von der dritten Ebene genutzt, um Produktionspläne zu erstellen. Die für die dritte Ebene eingesetzten Modelle und Algorithmen werden in Kapitel 6 vorgestellt.

Im vorliegenden Kapitel werden zu Beginn die Grundlagen der eingesetzen Simulationssoftware beschrieben. Nach einem kurzen Überblick über die am Markt vorhandenen Simulationsbibliotheken und den Auswahlkriterien werden die Pakete im Bezug auf die vorgestellten Anforderungen verglichen. Anschließend wird eine Bibliothek ausgewählt, die als Referenzimplementierung der im folgenden Kapitel vorgestellten Erweiterung dient.

Der folgende Abschnitt 4.3 stellt die Modifikationen vor, die an der Bibliothek vorgenommen wurden. Darauf aufbauend stellt Abschnitt 4.4 auf Seite 89 die Implementierung aller Komponenten der Ebene 0 vor. Anhand von UML-Diagrammen werden die wichtigsten Modifikationen und Designentscheidungen beschrieben.

4.1 Aktuelle Simulationspakete

Der Markt für Produkte im Bereich der Simulation umfaßt eine Vielzahl von Anbietern, sowohl kommerzielle, als auch frei erhältliche Pakete, die ihren Ursprung meist im akademischen Bereich haben. Eine Übersicht über kommerzielle Produkte findet sich in Swain (2007). Diese Übersicht wurde auf Basis einer Selbstauskunft von 48 Herstellern erstellt und enthält eine umfassende Übersicht der auf dem Markt verfügbaren Softwarelösungen. Kommerzielle Produkte sollen in dieser Arbeit nicht weiter betrachtet werden, da die Anbieter die Quellcodes ihrer Programme nicht offenlegen und eine Erweiterung mit den in den Programmen vorhandenen Sprachen nur sehr eingeschränkt möglich ist. Ein weiterer nicht zu vernachlässigender Aspekt ist die Geschwindigkeit.

Da die meisten Produkte eine graphische Oberfläche haben, wird die Geschwindigkeit durch die Darstellung der Elemente stark verringert. Als letztes Argument gegen den Einsatz von kommerzieller Software innerhalb dieser Arbeit ist die nicht vorhandene Plattformunabhängigkeit aufzuführen. Die meisten Hersteller konzentrieren die Entwicklung auf eine Plattform und vernachlässigen die anderen nicht so verbreiteten Plattformen.

Bei der Auswahl einer Simulationsbibliothek, die als Grundlage für weitere Entwicklungen dienen soll, wurden folgende Kriterien eingesetzt:

Codepflege Die Bibliothek soll aktiv gepflegt werden. Bibliotheken, deren letzte Revision bereits drei Jahre oder älter sind und die nur sporadisch von einzelnen Personen gepflegt wurden, sind ausgeschlossen worden.

Lizenz Alle hier aufgeführten Bibliotheken sind frei verfügbar. Frei bedeutet entweder, daß sie unter einer Open-Source Lizenz stehen oder für akademische Zwecke inklusive des Quellcodes zur Verfügung stehen.

Plattformunabhängigkeit Eine der Kernanforderungen besteht in der Unabhängigkeit vom eingesetzten System. Es wurden daher nur Bibliotheken betrachtet, die in einer Sprache geschrieben sind, die auf unterschiedlichen Plattformen (Betriebssystem und Hardware) verfügbar ist und bei der garantiert werden kann, daß die Programme sich identisch verhalten. Vorzugsweise wurden Bibliotheken ausgewählt, die in der Sprache Java geschrieben sind, lediglich eine auf C++ basierende Bibliothek wurde aufgenommen (OMNet++).

Modellierungsarten Die Modellierung einer Simulation kann auf unterschiedliche Weisen erfolgen, die sich teilweise ineinander überführen lassen. Die Unterschiede zwischen einer diskreten und kontinuierlichen Simulation liegen in der Repräsentation der Zeitachse (vgl. Law u. Kelton 2000, S. 3, Page u. Kreutzer 2005, S. 11,

Pidd 2004, S. 25ff). Bei Vorliegen einer diskreten Zeitachse werden die unterstützten Modellierungen unterschieden in diskrete-ereignisorientierte Simulation und prozeßorientierte Simulation (zu den Unterschieden vgl. Law u. Kelton 2000, S. 11, Page u. Kreutzer 2005, S. 97ff, Pidd 2004, S. 101).

Tabelle 4.2 auf der nächsten Seite zeigt einen Ausschnitt der aktuell verfügbaren Simulationsbibliotheken, unter Berücksichtigung der oben genannten Kriterien. Die Auswahl erhebt keinen Anspruch auf Vollständigkeit, sondern stellt nur den Teil dar, der eine größere Verbreitung hat. In dieser Übersicht wurden auch nicht die Bibliotheken berücksichtigt, die im Bereich der Kommunikationsnetzwerke eingesetzt werden, wie z. B. ns2 (ISI 2009) oder JiST (Barr u. Haas 2009). Diese erfüllen zwar auch die obigen Kriterien, sind jedoch teilweise bereits im Simulationskern auf den Anwendungsbereich ausgerichtet, so daß die Umstellung auf eine allgemein einsetzbare Bibliothek mit einem erhöhten Aufwand verbunden ist.

Die Bibliothek **D-SOL** der TU Delft ist ein umfangreiches Paket zur Simulation von Modellen aus unterschiedlichen Bereichen (Jacobs u. a. 2002, S. 793). Diese umfassen sowohl diskrete Ereignisse, als auch prozessorientierte und kontinuierliche Bestandteile. Kontinuierliche Komponenten werden durch Differentialgleichungen dargestellt und können über einen entsprechenden Integrator berechnet werden. Der Schwerpunkt des Paketes und der Entwicklungen liegt im Bereich der verteilten Simulation.

Einen ähnlichen Funktionsumfang besitzt **Ptolemy-II** (Eker u. a. 2003, S. 127), dessen Hauptanwendungsbereich die Simulation von Modellen mit heterogenen Komponenten ist. Ptolemy-II definiert hierfür sogenannte „domains“, die mit dem zentralen Modell zusammenarbeiten. Es existieren Domains für rein ereignisorientierte Simulationen, kontinuierliche, prozessorientierte Simulation sowie für parallel laufende Prozesse mit unterschiedlichen Synchronisationsmechanismen und auch Zustandsmaschinen. Der Schwerpunkt der Anwendungen liegt in der Simulation von physikalischen, elektrischen und informationstechnischen Systemen.

Paket	Version	Lizenz	Homepage	Sprache	Arten der Modellierung		
					diskret ereignisorientiert	prozeßorientiert	kontinuierlich
DESMO-J	2.1.4b	Apache License 2.0	www.desmoj.de	Java	●	●	○
D-SOL	2.0.9	GPL	sk-3.tbm.tudelft.nl	Java	●	●	●
JavaSim	0.3	LGPL	javasim.codehaus.org	Java	○	●	○
JSL	—[a]	GPL	www.uark.edu/~rossetti	Java	●	○	○
OMNet++	4.0.1	frei[b]	www.omnetpp.org	C++	●	◑	○
Ptolemy II	7.0.1	UC Berkeley [c]	ptolemy.eecs.berkeley.edu	Java	●	●	●
SSJ	2.1.3	GPL	www.iro.umontreal.ca/~simardr/ssj	Java	●	●	●

[a] Für das Paket wurde keine Versionsnummer angegeben, da es sich um die erste Version handelt.
[b] Für nicht kommerzielle Anwendungen frei.
[c] Vergleichbar mit einer BSD Lizenz

Tabelle 4.2: Auswahl an freien Simulationspaketen in Java (Stand: 27.10.2009)

Von L'Ecuyer u. a. (2002a, S. 234) stammt die Bibliothek **SSJ**, welche den Schwerpunkt auf die Simulation von stochastischen Elementen legt. Auch hier werden alle drei Arten der Modellierung unterstützt, wobei die Simulation kontinuierlicher Elemente im Vergleich zu den vorher genannten Paketen nur in den Grundzügen vorhanden ist. Hervorzuheben ist die hohe Anzahl an statistischen Verteilungen und Zufallszahlengeneratoren, die kein anderes Paket in dieser Vielfalt bereitstellt. Die Struktur der Klassen und die Wiedergabe der an einer Simulation beteiligten Elemente sind im Vergleich zu den anderen Paketen nicht so deutlich herausgearbeitet.

DESMO-J ist eine Bibliothek, die in Java implementiert wurde und sowohl die rein ereignisorientierte, als auch prozessorientierte Sicht unterstützt (Page u. Kreutzer 2005, S. 98 und S. 263). Der Schwerpunkt liegt auf der Bereitstellung von einem Framework, dessen Klassenstruktur die verschiedenen Elemente einer Simulation möglichst exakt abbildet. Die Anwendungsbereiche sind im Gegensatz zu den meisten anderen Bibliotheken nicht auf ein Gebiet beschränkt, sondern können unterschiedlichste Gebiete umfassen. Die Trennung der Klassen, die den Kernbereich einer ereignisorientierten Simulation bilden, von den Klassen für die Anwendungsbereiche, ist hier sehr deutlich ausgeprägt.

Eine Kombination von Teilen aus SSJ und DESMO-J stellt **JSL** dar. JSL versucht aus beiden Bibliotheken die besten Komponenten zu vereinen. Dabei wurde der Bereich bisher auf eine rein diskrete ereignisorientierte Sicht beschränkt. Als Sprache wurde ebenfalls Java eingesetzt, wobei die neuesten Spracheigenschaften wie Generics nur teilweise eingesetzt wurden. Trotzdem stellt es einen interessanten Ansatz dar, da sich DESMO-J und SSJ sehr gut ergänzen.

Javasim ist eine Bibliothek, die aus wenigen Klassen besteht und sich auf Prozesse spezialisiert hat. Der Quellcode wurde noch nicht an die neuen Fähigkeiten von Java 5 und Java 6 angepaßt, so daß die Threads hier nicht von den neuen Synchronisationsmechanismen profitieren können (vgl. auch Abschnitt 4.3 auf Seite 82).

OMNet++ wurde als einziges Paket in dieser Übersicht in C++ programmiert (Varga 2001). Die Lizenz ist so gestaltet, daß sie für akademische Zwecke frei ist und der Quellcode bereitgestellt wird.

Kommerzielle Lizenzen müssen getrennt erworben werden. OMNet++ besitzt eine auf Eclipse basierende plattformunabhängige Programmierumgebung, mit der sich bereits mitgelieferte Komponenten in einer graphischen Oberfläche verbinden und konfigurieren lassen. Gleichzeitig ist es möglich, eigene Komponenten in C++ zu schreiben und diese ebenfalls einzubinden. Außer einer diskreten ereignisorientierten Simulation lassen sich auch prozessorientierte Elemente simulieren. Hierfür existieren allerdings keine speziellen Klassen, sondern nur Hilfsklassen, die Kanäle und Botschaften für den Informationsaustausch bereitstellen. Auch die parallele bzw. verteilte Simulation wird durch entsprechende Klassen sehr gut unterstützt. Anwendungsbereiche sind hier allgemeine Warteschlangensysteme sowie hardwarenahe Simulationen von Kommunikationsnetzwerken und deren Komponenten.

4.2 Auswahl einer Simulationsbibliothek

Das Entwicklungsziel dieser Arbeit ist die Bereitstellung einer Bibliothek, die kontinuierliche Prozesse effizient simuliert. Der Anwendungsbereich der Bibliothek ist die Simulation einer hierarchische Produktionsplanung in der Prozeßindustrie. Die Wiedergabe der kontinuierlichen Prozesse innerhalb der Produktionsplanung erfolgt unter der Annahme, daß jeder kontinuierliche Prozeß ausschließlich von der Zeit abhängig ist (vgl. Kapitel 5). Durch diese Vereinfachung kann auf den Einsatz von Differentialgleichungen zur Beschreibung des Modells verzichtet werden, so daß die Bibliothek nur eine diskrete ereignisorientierte Simulation unterstützen muß. Im folgenden Abschnitt wird begründet, warum die prozeßorientierte Modellierung hier keine Vorteile gegenüber einer diskret-ereignisorientierten Modellierung besitzt, so daß auch dieser Aspekt bei der Auswahl nicht berücksichtigt werden muß. Weitere einschränkende Kriterien sind die Plattformunabhängigkeit und eine einfache, klare Klassenhierarchie.

Für die Unterstützung unterschiedlicher Plattformen und Rechnerarchitekturen wurde bereits in Abschnitt 2.3 bei der Implementierung eines Zufallszahlengenerators die Sprache Java ausgewählt. Diese Ent-

scheidung fiel nicht isoliert, sondern unter Berücksichtigung der in Abschnitt 4.1 beschriebenen Bibliotheken. Aus den in Tabelle 4.2 auf Seite 79 aufgeführten Bibliotheken entfällt somit das Paket OMNet++. Die Pakete D-SOL und Ptolemy-II erfüllen zwar auch die Anforderungen, sind jedoch aufgrund ihrer Ausrichtung und den damit verbundenen Anpassungen im Kernel nur schwer anzupassen. JavaSim wiederum hat zwar eine einfache Struktur, diese läßt jedoch in einigen Bereichen Lücken offen, so daß es auch ausscheidet. JSL war zu dem Zeitpunkt, an dem diese Arbeit begonnen wurde noch nicht veröffentlicht, so daß nur noch SSJ und DESMO-J zur Wahl standen. SSJ stand zu dem Zeitpunkt, an dem mit der Entwicklung begonnen wurde, noch nicht unter einer freien Lizenz, und der Quellcode war somit nicht verfügbar. Die Entscheidung für eine Bibliothek fiel somit auf DESMO-J. Nach der Änderung der Lizenz von SSJ und der Freigabe des Quellcodes konnte auch die Struktur der Klassen im Vergleich zu DESMO-J untersucht werden. Es stellte sich heraus, daß die Entscheidung für DESMO-J richtig war, da die Stärken von SSJ in der Vielzahl an verfügbaren stochastischen Verteilungen und Zufallszahlengeneratoren liegt und nicht in der exakten Abbildung von Elementen, Konzepten und Sichten einer Simulation innerhalb einer Klassenbibliothek. In zukünftige Entwicklungen kann jedoch die in SSJ vorhandene Bibliothek mit stochastischen Verteilungen und Zufallszahlengeneratoren in DESMO-J integriert werden, um speziell für den akademischen Bereich das Einsatzspektrum zu erweitern. Die ersten Schritte in Richtung eines integrierten Paketes wurden bereits mit dem Paket JSL gemacht.

4.3 Modifikationen von DESMO-J

Entwicklung eines Minimalsystems

Das Paket DESMO-J ermöglicht es, innerhalb einer Simulation gleichzeitig ereignisorientierte und prozeßorientierte Elemente zu verarbeiten. Die Modellierung von Elementen als Prozeß setzt eine Darstellung der Zustände und Transitionen in Form von Zustandsdiagrammen (vgl. Pa-

ge u. Kreutzer 2005, S. 98, Object Management Group 2009, Kap. 15) voraus. Diese Form der Modellierung ist für alle Entitäten innerhalb einer Materialflußsimulation möglich. Folgende Gründe sprechen jedoch gegen diese Darstellung:

- Jedes prozeßorientierte Element wird in Java durch einen eigenen Systemprozeß (Thread) abgebildet. Die Performance der von Java gebildeten Threads ist im Vergleich zur reinen ereignisorientierten Simulation, die innerhalb eines einzigen Threads abläuft, sehr schlecht. Threads werden in Java als native System-Threads erzeugt und unterliegen somit auch der Prozeßverwaltung des Betriebssystems. Innerhalb eines Multi-Thread Programms müssen alle Ressourcen, auf die gemeinsam schreibend (und teilweise auch lesend) zugegriffen wird, aufwendig synchronisiert werden, was erheblich Zeit kostet. Ein weiterer Nachteil ist die erhöhte Fehleranfälligkeit in der Programmierung durch die zusätzlichen Synchronisationsmechanismen.

- Nach der Einführung von Java 5 wurde der Mechanismus zur Steuerung von Threads und zur Synchronisation vollstängig überarbeitet (und verbessert), so daß auch eine Anpassung des Quellcodes von DESMO-J nötig ist.

- Die Darstellung als Zustandsmaschinen ist insbesondere für die Lager bei Berücksichtigung aller Fälle (Backlog, Lostsales) sehr komplex. Eine Vereinfachung der Zustandsmaschine auf wenige Zustände und Transitionen führt hingegen zu einer Logik, wie sie auch bei der ereignisorientierten Sichtweise implementiert werden muß, so daß keine Verbesserung durch die andere Sichtweise erreicht werden kann.

Daher wurden sämtliche Bestandteile der prozeßorientierten Simulation aus DESMO-J entfernt, um ein rein ereignisorientiertes Minimalsystem zu erhalten. Dieses System wird im Folgenden auch als Kernsystem bezeichnet. Sämtliche Erweiterungen bauen auf diesem System auf. Gleichzeitig werden im Kernsystem auch einige Neuerun-

gen eingeführt, die entweder die Modellierung erleichtern, oder eine Vereinfachung der internen Datenstrukturen darstellen.

Einsatz von Value-Objekten

„Value Objects are instantiated to represent elements of the design that we care about only for what they are, not who or which they are.“ (Evans 2004, S. 98). Value-Objekte (oder auch Value-Typen genannt) sind Klassen, die ihren Zustand bei der Initialisierung erhalten und ihn nicht mehr ändern können. Im Gegensatz zu Entitäten (Evans 2004, S. 89) ist Ihre Identität irrelevant, da sie nur genutzt werden, um eine Menge an Attributen auszudrücken. Liegen zwei inhaltlich identische Instanzen vor, so ist der Programmablauf identisch, unabhängig davon, welche der beiden Instanzen verwendet wird.

Da sie nur einen Zustand besitzen und ihre Identität für den Einsatz nicht verwendet wird, reicht eine Instanz innerhalb eines Programms aus. Auf diese Instanz kann von anderen Objekten beliebig oft verwiesen werden, ohne das Verhalten der anderen Objekte zu beeinflussen. Dieses „Object-Sharing“ ist auch einer der Gründe, für das Anlegen von Value-Objekten (Fowler 1999, S. 179). Aufgrund einer aufwendigen Initialisierung oder einer hohen Anzahl an Referenzen kann durch den Einsatz eines „Flyweight“ Entwurfsmusters die Erzeugung der Instanzen konsolidiert werden (vgl. Evans 2004, S. 100, Gamma u. a. 1995, S. 195).

Eine der wichtigsten Anforderungen an ein Value-Objekt ist die Unveränderlichkeit (im engl. immutable, Evans 2004, S. 99, Ausnahmen hiervon werden in Evans 2004, S. 101 erläutert). Auf Java bezogene Anforderungen zur Umsetzung der Designaspekte werden von Bloch (2008, S. 73) vorgestellt. Zur Erzeugung wird eine Builder-Klasse eingesetzt, wenn viele Konfigurationsoptionen zur Verfügung stehen (vgl. Gamma u. a. 1995, S. 97 und Bloch 2008, S. 11). Ist die Anzahl an Konfigurationsmöglichkeiten gering, so werden statische Factory-Methoden verwendet. In beiden Fällen wird der Konstruktor verborgen, um eine direkte Initialisierung durch den Benutzer zu vermeiden. Nach der Erzeugung ist der Zustand nicht mehr veränderlich. Eine äquivalente

Darstellung von Value-Objekten ist das Konzept der *immutable class* von Bloch (2008, S. 73).

Der Aspekt der inhaltlichen Identität wird in Java durch die Methoden *equals* und *hashCode* ausgedrückt. Ihre Implementierung wird grundsätzlich für alle Klassen empfohlen, die zum Informationsaustausch zwischen zwei Klassen oder zur Charakterisierung einer Klasse eingesetzt werden (Bloch 2008, S. 33,45). Eine Implementierung ist daher für Value-Objekte zwingend erforderlich.

Da ein Value-Objekt bereits bei der Instanziierung seinen Zustand erhält, kann der Hashwert berechnet und zwischengespeichert werden (Bloch 2008, S. 49) - spätere Berechnungen werden so überflüssig. Innerhalb von *equals* wird dieser Wert eingesetzt, um verkürzend festzustellen, daß der Inhalt zweier Instanzen nicht identisch ist.

Value-Objekte werden sowohl auf dieser Ebene, als auch auf der Materialflußebene eingesetzt. Ihr Hauptanwendungsbereich liegt im Informationsaustausch (z. B. die Klasse *AbstractObjectID*) sowie in der Darstellung komplexer Datenstrukturen und den damit verbundenen Berechnungen (z. B. in *AbstractRecipe*). Die sich hieraus ergebenden Vorteile werden detailliert in Evans (2004, S. 100) und Bloch (2008, S. 73) aufgeführt. In den Abschnitten zur Implementierung der Schnittstelle *Recipe* durch *AbstractRecipe* auf Seite 149 und *AbstractObjectID* auf Seite 64 werden diese anhand der konkreten Implementierung detailliert beschrieben.

Serialisierung

Ein wesentliches Designziel der ersten Ebene ist die Implementierung der Vervielfältigung von Simulationsmodellen. Diese kann eingesetzt werden, um ein Modell mit unterschiedlichen Zufallszahlen zu simulieren und so eine breitere statistische Basis zu erhalten. DESMO-J bietet hierfür keinen Mechanismus, so daß ein entsprechender Mechanismus entwickelt und implementiert werden mußte. Eine wesentliche Anforderung war eine einfache Implementierung, welche möglichst wenige Anpassungen am Code benötigt und die auch für alle darauf aufbauenden Ebenen einsetzbar ist.

Java stellt zwei elementare Methoden zur Vervielfältigung zur Verfügung: Serialisierung und Klonen. Die Serialisierung ist ein Vorgang, der den Inhalt eines Objekts in einen Datenstrom überführt, um ihn z. B. über ein Netzwerk zu übertragen oder in einer Datei abzuspeichern (Sun 2006, Interface *Serializable*, Bloch 2008, S. 289). Durch die Deserialisierung wird der Vorgang umgekehrt und neue Instanzen, die unabhängig von den serialisierten Originalen sind, werden erzeugt. Das Klonen von Objekten hingegen führt direkt zu einer Kopie des Originals im Speicher und nicht zu einer anderen Darstellung, die davon getrennt gespeichert werden kann. Der Aufwand zur Herstellung eines Klons ist wesentlich höher als bei der Serialisierung, da für jedes Objekt individuell die Methode zum Klonen überschrieben werden muß (Bloch 2008, S. 54). Gleichzeitig muß darauf geachtet werden, daß alle Referenzen zu anderen Objekten ebenfalls geklont werden, um vollständig unabhängige Kopien zu erhalten. Vergleichbare Probleme gibt es bei der Serialisierung nicht. Die für die Serialisierung spezifischen Probleme und Designaspekte werden in Bloch (2008, S. 289) beschrieben. Diese treten innerhalb der Simulation jedoch nicht auf, so daß der Serialisierung der Vorzug gegenüber dem Klonen gegeben wurde.

Voraussetzungen für den Einsatz der Serialisierung ist die Verwendung der Schnittstelle *Serializable* in jedem Objekt, welches serialisiert wird, inklusive aller in dem Objekt verwendeten Felder. Sind diese Felder von einem elementaren Datentyp, so ist dies unproblematisch. Felder, die ein anderes Objekt repräsentieren, müssen hingegen ebenfalls serialisierbar sein, sofern sie nicht als transient deklariert sind (Sun 2005, Kap. 8.3.1.3). Der gesamte Simulationskern wurde vollständig umgestellt, so daß jede Klasse serialisierbar ist. Die Serialisierung wurde so implementiert, daß jede Klasse den Wert von *serialVersionUID* auf 1 setzt. Dieser Standardwert signalisiert dem Programmierer, daß die Klasse nicht für eine permanente Speicherung auf einem Datenträger geeignet ist. Dadurch wird vermieden, daß nach Änderungen der internen Datenrepräsentation einer Klasse auch die Methoden zur Serialisierung-/Deserialisierung angepaßt werden müssen (Bloch 2008, S. 289). Die Serialisierung wird nur für die Vervielfältigung im Spei-

cher während eines Programmlaufs eingesetzt, so daß die von Bloch beschriebenen Einschränkungen hier nicht zutreffen.

Außer der Vervielfältigung von Testläufen gibt es ein weiteres Anwendungsgebiet, die verteilte Simulation. Zukünftige Anwendungen können die in Abschnitt 4.4.1 auf Seite 98 vorgestellte Klasse *Testrunner* so erweitern, daß die serielle Ausführung von mehreren Testläufen parallel erfolgt. Da die Serialisierung eigenständige Objekte erzeugt, kann dieser Mechanismus problemlos zur Vervielfältigung und anschließender paralleler Ausführung der Instanzen genutzt werden. Für die Behandlung von gemeinsamen Ressourcen muß der Code noch so angepaßt werden, daß diese nicht dupliziert werden.

Umstieg auf Java 5.0

Mit der Veröffentlichung Java 5.0 im Jahr 2005 hat Sun mehrere neue Sprachelemente eingeführt. Die Einführung von Generics (Sun 2005, Abs. 8.1.2, 8.4.4., 8.8.4, 9.1.2) ermöglicht die Überprüfung der Typsicherheit bereits während der Compilierung. Dieses Sprachelement wird auf allen Ebenen der Simulation eingesetzt.

Innerhalb des Kerns wurden sämtliche Vorkommen von Klassen aus dem Java-Collections-Framework typisiert. Durch den Einsatz von Methoden aus diesem Framework konnten viele Klassen stark vereinfacht werden. Ein Beispiel sind die Operationen zum Sortieren der Ereignisliste in der Klasse *EventPriorityQueue*, die vom *Scheduler* genutzt wird (siehe Abschnitt 4.4.1 auf Seite 105).

Zufallszahlengenerator

Das Paket DESMO-J enthält zwei verschiedene Generatoren - zum einen eine Adapter-Klasse für den Zufallszahlengenerator, der von Sun im Paket *java.utils* bereitgestellt wird, sowie eine Implementierung des Mersenne Twister von Matsumoto u. Nishimura (1998). Die Implementierung von Sun entspricht einem einfachen linearen Kongruenzautomaten, wie sie von Knuth (1997, Kap. 3.2.1) beschrieben wird. Die Periodenlänge wird durch den Modulo begrenzt (Knuth 1997, Kap.

3.2.1.2), welcher den Wert $2^{48} \approx 2,8 \cdot 10^{14}$ annimmt. Diese Periodenlänge ist für Simulationen mittlerer Größe ausreichend hoch. Der Generator erfüllt jedoch nicht das Kriterium der Reproduzierbarkeit, da Sun keine Garantie für die Beibehaltung der Implementierung abgibt, sondern nur allgemeine Spezifikationen vorgibt und implementiert. Auch wenn diese Veränderungen in der Implementierung von grundlegenden Klassen selten sind, wurden sie hier bereits durchgeführt (Sun 2006, *java.utils.Random.nextFloat*).

Die zweite Implementierung, die in DESMO-J zur Verfügung steht - der Mersenne Twister von Matsumoto u. Nishimura (1998) - besitzt eine Periodenlänge von $2^{19937} - 1 \approx 4,31 \cdot 10^{6001}$, welche für eine Simulation vollkommend ausreichend ist. Wie bereits in Abschnitt 2.3 dargestellt hat dieser Generator einen sehr hohen Speicherbedarf, so daß er (vorerst) nicht verwendet wird. Zur Anwendung kommt daher der Generator MRG32k3a von L'Ecuyer u. a. (2002b), dessen Schnittstellen so modifiziert wurden, daß sie eine einfache Behandlung der Teilströme ermöglichen. Diese Teilströme werden verwendet, um Testläufe mit veränderten Anfangsbedingungen zu replizieren. Details hierzu erläutert Abschnitt 4.4.1 auf Seite 98.

Etablierung eines Ebenenmodells

Zur Unterstützung des Refactoring hin zu einem Simulationskern und zur besseren Systematisierung wurden alle Elemente der Simulationsbibliothek in ein Ebenenmodell eingeordnet. Das Ziel dieser Klassifikation ist die Zuordnung von Funktionsbereichen zu Ebenen. Gleichzeitig soll die Sichtbarkeit der Klassen so verringert werden, daß jede Ebene Kontakt zu anderen Ebenen nur über öffentliche Schnittstellen aufnimmt. Alle anderen Elemente, die nur zur Bereitstellung von Diensten innerhalb der Ebene dienen, werden somit verdeckt. Die Konstruktion hat den Vorteil, daß die interne Datenorganisation verändert werden kann, ohne die Schnittstellen anzupassen. Das Designprinzip lautet: Jede Ebene soll nur Elemente aus der unmittelbar darunterliegenden Ebene benutzen. Nur die Elemente einer Ebene, die eine grundlegende Infrastruktur bereitstellen, dürfen von allen darüberliegenden Ebenen

verwendet werden. Der Aufbau der Ebenen entspricht dem in Tabelle 4.1 auf Seite 75, so daß hier nur kurz die Designziele vorgestellt werden:

Ebene 0 enthält die Kernelemente einer Simulation. Die Ebene stellt die gesamte Infrastruktur für eine diskrete ereignisorientierte Simulation bereit.

Ebene 1 enthält alle Elemente zur Simulation von diskreten und kontinuierlichen Materialflüssen.

Ebene 2 ist eine Zwischenschicht, die Elemente von Ebene 1 zu Einheiten zusammenfaßt für Planungsaufgaben der darüberliegenden Ebene. Typische Beispiele hierfür sind Fertigungsleitstände, die für eine Maschinengruppe verantwortlich sind. Elemente der Ebene 2 haben direkten Zugriff auf einzelne Ressourcen oder Lager und stellen Informationen hierüber der dritten Ebene zur Verfügung.

Ebene 3 stellt den Kontakt zur Planung her oder enthält Planungsalgorithmen, um den Materialfluß zu steuern. Die Pläne werden von dieser Ebene aus an die Fertigungsleitstände weitergegeben, so daß die dritte Ebene keinen direkten Kontakt zu den einzelnen Maschinen erhält.

In den folgenden Kapiteln werden die wichtigsten Elemente der ersten beiden Ebenen vorgestellt, sowie ein Überblick der Funktionsweise und Koordination der Elemente innerhalb einer Ebene und zwischen den Ebenen gegeben. Die Funktionen der Ebenen zwei und drei werden in Kapitel 6 vorgestellt, ohne näher auf die Implementierung einzugehen, da diese im Wesentlichen auf den dort vorgestellten Algorithmen aufbaut.

4.4 Implementierung

Die unterste Ebene stellt eine allgemeine Infrastruktur für diskrete, ereignisorientierte Simulationen bereit. Die Elemente dieser Infrastruktur

wurden von DESMO-J übernommen und in der Sichtbarkeit, sowie Funktionsweise angepaßt. Durch die Neuordnung wurde das gesamte Framework stark vereinfacht, so daß es als Referenzimplementierung der Erweiterungen, die in Kapitel 5 vorgestellt werden, dient. Die Übertragung der dort vorgestellten Erweiterungen auf andere Bibliotheken ist aufgrund des generischen Charakters, den diese Ebene besitzt, mit geringem Aufwand möglich.

Folgende Designziele waren bei der Durchführung des Refactoring und der Neustrukturierung vorgegeben:

- Trennung des Simulationskerns von den anderen Simulationsobjekten
- konsequente Verwendung der in Java 5 eingeführten Sprachelemente
- Benutzung von Bibliotheksfunktionen von Java für Standardaufgaben
- Ausrichtung auf eine rein ereignisorientierte Simulation, ohne die Elemente der prozeßorientierten Simulation.

Außer den klassenspezifischen Modifikationen wurden auch folgende Änderungen im Design vorgenommen:

- Durch den Wegfall der prozeßorientierten Simulation haben sich bei vielen Komponenten die inneren Abläufe vereinfacht. Dies betrifft auch die Schnittstellen, da diese nicht mehr mit Prozessen arbeiten müssen.
- Die interne Repräsentation der Daten wurde stärker in Richtung der von Bloch (2008, S. 73) vorgeschlagenen Empfehlungen weiterentwickelt. Alle Felder, bei denen ihre Funktion es erlaubte, wurden als *final* deklariert.
- Klassen, bei denen alle Felder als *final* deklariert werden konnten (z. B. *SimTime*), werden je nach Einsatzbereich als Value-Objekt

Infrastruktur	Modellkomponenten
Model	ModelComponent
Experiment	Schedulable
Testseries	Entity
SimTime	Event
SimClock	ExternalEvent
EventNote	ExternalEventStop
Scheduler	
EventQueue	
DistributionManager	

Tabelle 4.3: Einteilung der Klassen in Ebene 0

behandelt (vgl. Abschnitt 4.3 auf Seite 84). Für diese Klassen wurden die Methoden zur Berechnung des Hashwertes und zur Überprüfung der inhaltlichen Identität mit *equals*, wie von Bloch (2008, S. 33) beschrieben, implementiert.

Die in den folgenden Abschnitten vorgestellten Klassen lassen sich in zwei Gruppen einteilen: Infrastruktur-Klassen und Modellkomponenten. Die Tabelle auf dieser Seite zeigt die wichtigsten Vertreter beider Gruppen.

Infrastruktur-Klassen stellen den Rahmen bereit, in dem die Simulation stattfindet. Sie werden entweder nur einmal pro Modell eingesetzt (z. B. *Scheduler* oder *DistributionManager*) oder stellen grundlegende Operationen zur Verfügung, daß sie von nahezu allen Klassen benutzt werden (z. B. *SimTime*). Ein weiteres wesentliches Merkmal ist der Grad der Bindung an ein Modell. Das Modell besitzt eine Infrastruktur und nutzt diese, die Infrastruktur ist aber nicht von einem Modell abhängig. Ausnahmen gibt es nur bei den Klassen Experiment und Testseries. Da diese die Modelle verwalten, gehören sie jedoch auch zur Infrastruktur. Im Gegensatz hierzu besitzen Modellkomponenten eine direkte Verbindung zu einem Modell. Modellkomponenten können nur innerhalb eines Modells existieren und werden bei der Erzeugung fest an das

Modell gebunden. Sie benutzen die Verbindungen zum Modell, um mit der Infrastruktur zu interagieren. Eine Einordnung der Klassen, aus denen die Modellkomponenten entstehen, zeigt die Abbildung auf der nächsten Seite.

Die Klassen, die für die Erzeugung von stochastischen Verteilungen verantwortlich sind, wurden hier nicht weiter aufgeführt. Die Implementierung erfolgt über eine generische Wrapper-Klasse, welche direkt die Schnittstelle *RealDist* verwendet und eine Klassenhierarchie aufbaut, die identisch zu der in Abschnitt 3.6.1 ist. Die Wrapper-Klasse enthält die Delegation aller Aufrufe an die entsprechende Verteilung sowie die Referenz auf das Modell, in dem die Verteilung eingesetzt wird. Die Bindung an das Modell erfolgt erst durch die entsprechenden Builder-Klassen, die eine eigene parallele Klassenhierarchie bilden. Da außer diesen Modifikationen keine weitere Funktionalität hinzugefügt wurde, wird auf eine weitere Darstellung verzichtet.

4.4.1 Infrastruktur-Klassen

Model

Das Modell beinhaltet in DESMO-J alle statischen Modellkomponenten, stochastische Verteilungen sowie Untermodelle (vgl. Abbildung 4.2 auf Seite 94). Im Minimalsystem beinhaltet ein Modell weiterhin alle statischen Modellkomponenten, sowie falls benötigt, auch stochastische Verteilungen und Elemente zur Erfassung von statischen Daten. Die Möglichkeit, ein Modell in mehrere Untermodelle aufzuteilen, wird nicht weiter unterstützt. Auch die Funktion, ein Modell auf seinen Anfangszustand zurückzusetzen, entfällt. Diese Funktionalität wird vom Minimalsystem durch den Einsatz der Serialisierung in der Klasse *Testrunner* erreicht. Details zur Implementierung finden sich dort.

Eine weitere Funktion der *Model*-Klasse ist die Initialisierung aller Modellkomponenten. Hierzu überschreiben Nachkommen der Klasse die *init*-Methode und führen dort alle Initialisierungsarbeiten durch. Das Modell wird an ein *Experiment* gebunden. Nach Aufruf der *start* Methode im Experiment wird *doInitialSchedules* im Modell aufgerufen.

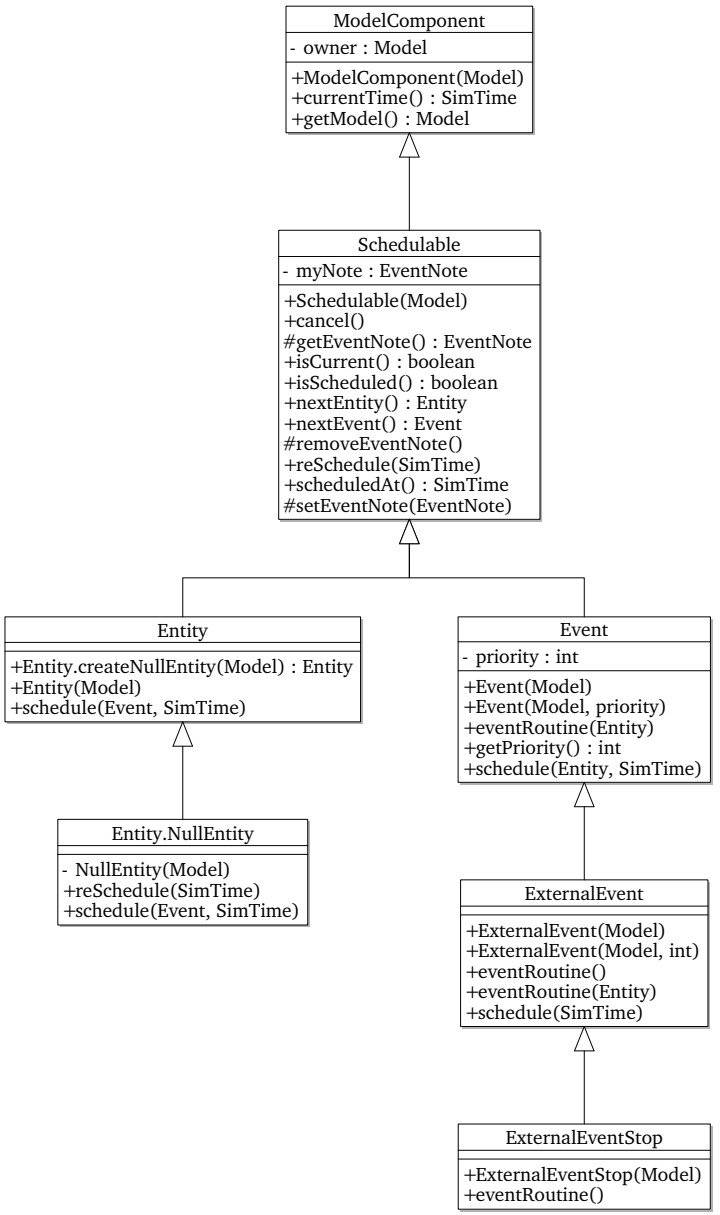


Abbildung 4.1: UML-Klassendiagramm für Modellkomponenten der Ebene 0

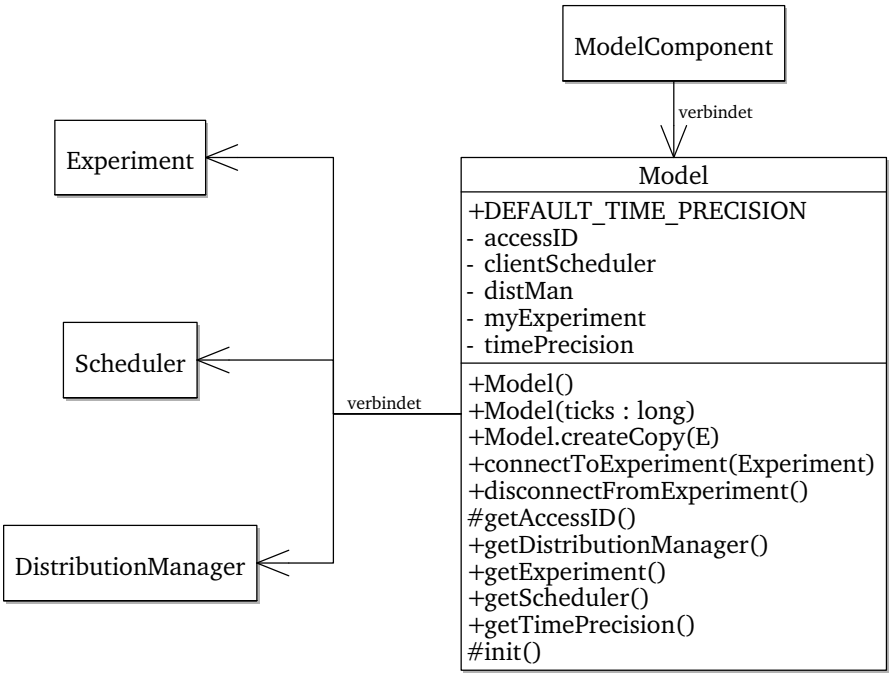


Abbildung 4.2: UML-Klassendiagramm *Model*

Die ersten Ereignisse werden so dem *Scheduler* (vgl. Seite 105) mitgeteilt. Ohne diese ersten Ereignisse würde die Simulation nicht laufen und wäre sofort beendet.

Sowohl die Struktur, als auch die Abfolge der Initialisierung wurden verändert. Die wichtigste Änderung betrifft den *Scheduler*. Im Gegensatz zu DESMO-J wird dieser nicht mehr der *Experiment* Klasse zugeordnet, sondern dem Modell. Dieses Refactoring war eine Konsequenz aus dem Verzicht auf Untermodelle und der Neuinterpretation der *Experiment* Klasse. Der *Scheduler* wird jetzt während der Initialisierung des Modells erzeugt und steht somit vor der Bindung an ein Experiment zur Verfügung. Hierdurch können bereits in der Initialisierungsphase des Modells Ereignisse an den Scheduler weitergegeben werden. In DESMO-J ist dies nicht möglich, da der Scheduler dem Experiment zugeordnet ist,

welches erst zum Zeitpunkt der Anbindung bekannt ist. Modellkomponenten, die einen Zugriff auf den Scheduler benötigen, erhalten diesen erst, nachdem das Modell an das Experiment angebunden wurde.

Eingesetzt wird diese neue Struktur in der Initialisierung von Modellbestandteilen. Komponenten, die während der Erzeugung ein Ereignis für sich oder eine verwaltete Komponente terminieren müssen, können dies bereits im Konstruktor ausführen. Gleichzeitig ist es jedoch auch möglich, daß diese Komponenten eine private nicht statische innere Klasse erzeugen, die bei der Erzeugung auf den Zeitpunkt 0 mit einer hohen Priorität terminiert wird. Im Gegensatz zu einer Initialisierung im Konstruktor steht hier immer eine Referenz zum umgebenden Modell zur Verfügung.

Da die Ereignisse zur Initialisierung eine hohe Priorität besitzen, wird garantiert, daß sie vor allen anderen Ereignissen abgearbeitet werden. Durch entsprechende Vergabe der Prioritäten kann sichergestellt werden, daß das Modell korrekt initialisiert ist, bevor der normale Simulationsablauf beginnt. Ist es jedoch nötig, daß die Initialisierung einer Komponente von einer anderen Komponente direkt abhängig ist, so müssen diese von einem sie umgebenden, übergeordneten Objekt verwaltet und initialisiert werden. Eine Alternative ist auch hier der Einsatz von Prioritäten, so daß bestimmte Komponenten vor anderen Komponenten initialisiert werden. Falls diese auch keinen Bezug zu anderen Komponenten herstellen müssen, ist diese Lösung vorzuziehen.

Die Erzeugung der Ereignisse, die für die Initialisierung einer Simulation benötigt werden, kann somit auf die Konstruktoren der Komponenten oder auf spezielle Klassen innerhalb der Komponenten verlagert werden. Die Methode *doInitialSchedules* ist in diesem Modell nicht mehr vorhanden und wird auch nicht unterstützt. Für den Programmierer ist dies vielleicht von Nachteil, da keine ausdrücklich definierte Methode zur Initialisierung mehr vorhanden ist. Dies wird jedoch durch die Möglichkeit, die Initialisierung im Konstruktor durchzuführen ausgeglichen. Der Aufwand für die Implementierung ist vergleichbar, so daß im Bezug auf das Design keine Nachteile entstehen.

Eine weitere Änderung betrifft die Verbindung von Simulationszeit und Modell. In Abschnitt 4.4.1 auf Seite 100 wird die diskrete

Simulationszeit vorgestellt, die eine vordefinierte Auflösung besitzt. Die Auflösung ist eine Eigenschaft des Modells (*timePrecision*), so daß alle Komponenten eines Modells mit der gleichen Zeitbasis arbeiten. In der Implementierung der Simulationszeit wird ausschließlich auf diese Zeitbasis zurückgegriffen, so daß alle Zeitangaben mit der vom Modell angegebenen Genauigkeit arbeiten müssen.

Experiment

Ein *Experiment* ist in DESMO-J eine Klasse, die ein einzelnes Modell umgibt und die Steuerung des Modells übernimmt. Es besitzt Methoden, mit denen ein Modell an das Experiment gebunden werden kann. Zwischen einem Experiment und einem Modell herrscht eine 1:n Beziehung, d. h. ein Modell kann nur bei einem Experiment angemeldet sein, ein Experiment kann jedoch mehrere Modelle verwalten. Die Experiment-Klasse hat die Funktion eines Containers, der mehrere Modelle parallel verwalten und steuern kann. Ein Modell muß sich bei einem Experiment registrieren, damit es gestartet werden kann. Es darf sich nur bei exakt einem Experiment registrieren, damit eindeutig festgelegt ist, welches Experiment die Steuerung übernimmt. Es ist jedoch zulässig, daß mehrere Modelle sich bei einem Experiment registrieren. Für zukünftige Erweiterungen hin zu einer parallelen Verarbeitung von mehreren Modellen (oder Kopien eines Modells) werden hiermit die Schnittstellen geschaffen.

Das Experiment steuert das Modell, indem es Methoden zum Start, Stop und zur Wiederaufnahme nach einem Stop anbietet. Die Verwaltung erfolgt im Experiment intern über eine Zustandsvariable, die auch das Verhalten des Experimentes bei Methodenaufrufen bestimmt. Die Abläufe zeigt das UML-Zustandsdiagramm in Abbildung 4.3 auf der gegenüberliegenden Seite.

Wie bereits im vorangehenden Abschnitt zur *Model*-Klasse beschrieben, wurde die Interaktion zwischen *Model* und *Experiment* Klasse verändert. Die interne Arbeitsweise der *Experiment*-Klasse ist vergleichbar mit der ursprünglichen Implementierung, durch die neue Aufgabenverteilung haben sich nur die Verantwortlichkeiten bei einigen Aufrufen

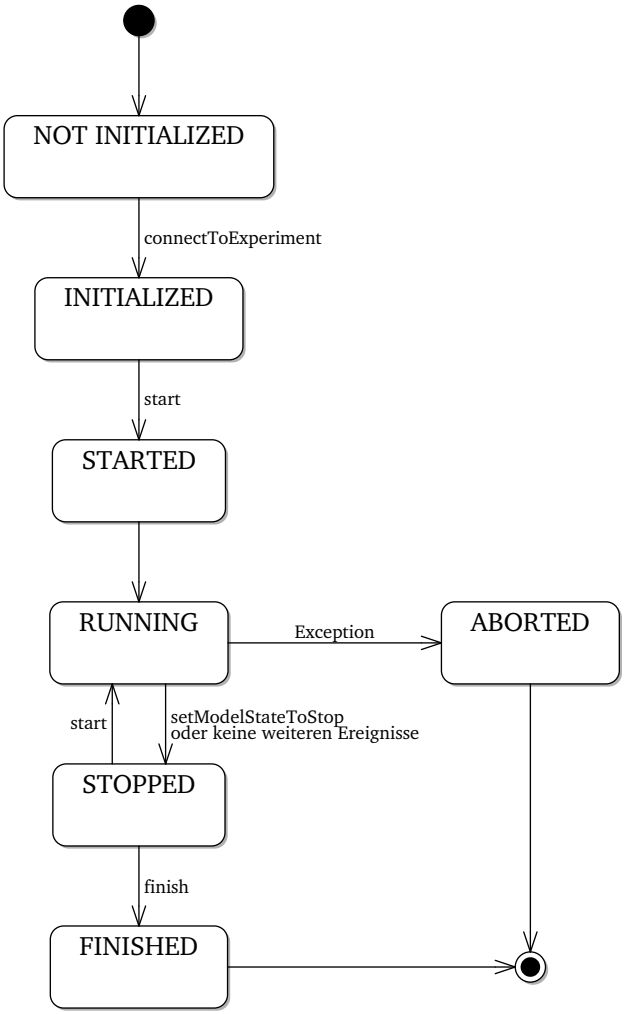


Abbildung 4.3: UML-Zustandsdiagramm eines Modells in einem *Experiment*

geändert. Der Zustand RUNNING wurde von der Originalimplementierung übernommen, kann jedoch entfallen, da der Übergang zwischen STARTED und RUNNING automatisch erfolgt. In der aktuellen Implementierung wird er nur intern benutzt. Der Übergang zwischen RUNNING und STOPPED wird durch den Aufruf von *setModelStateToStop* ausgelöst, welcher von *ExternalEventStop* durchgeführt wird. Weitere Änderungen wurden bereits in der Beschreibung der *Model*-Klasse erwähnt. Eine kleine technische Änderung betrifft die Darstellung der Zustände eines Modells. Diese können seit Java 5 durch eine *enum* Klasse typsicher dargestellt werden, so daß die bisherige Repräsentation durch ganzzahlige Konstanten entfällt.

Testseries

Die Schnittstelle *Testseries* wurde neu eingeführt, da DESMO-J bisher kein vergleichbares Konzept zur Wiederholung von Testläufen bereitstellt. Eine Testreihe kann das ihr übergebene Modell vervielfältigen und wiederholt mit unterschiedlichen Zufallszahlen ausführen. Sie verwendet hierfür mehrere Techniken, die in den vorangehenden Abschnitten beschrieben wurden, u. a. die Serialisierung (vgl. Abschnitt 4.3 auf Seite 85), einen Zufallszahlengenerator mit Teilströmen (vgl. Abschnitt 2.3) und den *DistributionManager* (vgl. Abschnitt 4.4.1 auf Seite 107). Als Referenzimplementierung wurde eine abstrakte Klasse *AbstractTestseries* geschaffen, deren wesentliche Eigenschaften im Folgenden beschrieben werden.

Die Testreihe wird durch einen Builder erzeugt, der als zwingende Argumente die Anzahl an Wiederholungen, eine Referenz auf ein Modell, sowie die Simulationsdauer übernimmt. Das übergebene Modell darf an kein Experiment gebunden sein, wenn es der Testreihe übergeben wird (Zustand NOTINITIALISED). Die Testreihe benutzt dieses Modell als Vorlage für die Kopien und serialisiert es. Die so erhaltene Vorlage wird anschließend deserialisiert für einen ersten Durchlauf. Jeder Durchlauf besitzt eine eindeutige Nummer, die mit jeder Wiederholung um eins erhöht wird. Der Startwert kann von außen vorgegeben werden, so daß gezielt einzelne Läufe wiederholt werden

können. Nachdem die Kopie erzeugt wurde, teilt die Testreihe dem *DistributionManager* die Nummer des Laufs mit, so daß die stochastischen Verteilungen auf die entsprechenden Teilströme ausgerichtet werden können. Anschließend wird das Modell an ein Experiment gebunden und ausgeführt. Nach Beendigung wird die Nummer des Laufs erhöht und die Schleife beginnt von vorn. Die Schleife wird so oft durchlaufen, bis die gewünschte Anzahl an Wiederholungen erreicht ist.

Der Vorteil dieser Konstruktion ist die Verwendung von Standardkomponenten und Schnittstellen (Serialisierung), die von jeder Modellkomponente im System per Definition implementiert werden müssen. Im Gegensatz zu DESMO-J ist es hier nicht nötig, ein Modell auf den Ursprungszustand zurückzusetzen, um es mit einer anderen Konfiguration zu starten. Als Nachteil ist zu erwähnen, daß die Deserialisierung im Vergleich zu anderen Konzepten der Vervielfältigung nicht sehr schnell ist. Bei den hier untersuchten Modellen, die sehr umfangreiche Daten verwalteten, wurden jedoch keine signifikanten Laufzeiten gemessen. Die Deserialisierung erfolgt in Sekundenbruchteilen, was im Vergleich zur Gesamtlaufzeit zu vernachlässigen ist.

Zur weiteren Verarbeitung von Ergebnissen und Daten können andere Klassen vor Beginn und nach Ende eines Experimentes informiert werden. Dies wird durch den Einsatz eines speziellen Observer (Gamma u. a. 1995, S. 293) implementiert. Auf Basis der Referenzimplementierung bestehen so unterschiedliche Möglichkeiten zur Erweiterung einer Testreihe. Diese werden z. B. genutzt, um Informationen über Testläufe in Datenbanken abzulegen oder vor einem Durchlauf Observer an Komponenten eines Modells (der aktuellen Kopie) zu binden, die statistische Daten sammeln. Nach Abschluß eines Durchlaufs wird eine Methode des Experiment-Observers aufgerufen, der wiederum die Observer der einzelnen Komponenten ansteuert, um die Daten zu übertragen. Die Daten aller Durchläufe eines Experimentes können somit gemeinsam gespeichert werden, was die spätere Auswertung und Aggregation von Daten eines Experimentes erleichtert.

SimTime

Die *SimTime* Klasse repräsentiert einen Zeitpunkt innerhalb der Simulation. Dieser Zeitpunkt wird durch eine Fließkommazahl ausgedrückt, deren Wert bei der Initialisierung dem Konstruktor übergeben wird. Die bisherige Implementierung übernahm diesen Datentyp in ein Feld und führte sämtliche Berechnungen und Vergleiche auf Basis der Fließkommazahl durch. Probleme entstehen durch die bei den Rechenoperationen auftretenden Rundungsfehlern, die ihre Ursachen in der internen Repräsentation haben.

Eine Fließkommazahl hat eine fest eingestellte Genauigkeit, die von der internen Darstellung abhängig ist. In Java wird für den Datentyp *double* die 64 bit IEEE 754 Darstellung verwendet (Sun 2005, Kap. 4.2.3), welcher eine Mantisse der Länge 52 bit und einen Exponenten von 11 bit besitzt (IEEE 1985, S. 3). Der Diskretisierungsfehler beträgt somit $\frac{1}{2^{52}} \approx 2 \cdot 10^{-16}$, was für die meisten Anwendungen ausreichend gering ist. Diese hohe Genauigkeit wird begleitet von den Rundungsfehlern, die bei Durchführung von elementaren Grundoperationen (Multiplikation und Division) auf den letzten Bits auftreten. Diese Rundungsfehler machen sich bemerkbar, wenn Zeitpunkte, die sich durch externe Berechnungen ergeben, mit vorhandenen Zeiten verglichen werden sollen. Diese Fehler lassen sich vermeiden, wenn die Simulation mit einer einstellbaren Genauigkeit arbeitet, die geringer ist als die vom System vorgegebene. In DESMO-J wird dies über einen Parameter ϵ erreicht, der in der Klasse ***SimClock*** definiert ist und auf den über das Experiment zugegriffen werden kann. Diese Genauigkeit wird jedoch nicht innerhalb des Schedulers berücksichtigt, so daß sie im Moment nur für die Auswertung von statistischen Daten eingesetzt wird.

Durch eine Änderung des Designs kann dieser Punkt berücksichtigt werden. Das UML-Diagramm auf der gegenüberliegenden Seite zeigt, daß die äußere Darstellung identisch mit der bisherigen Implementierung ist. Die interne Darstellung hingegen wurde vollständig überarbeitet. Alle Zeitangaben werden in diskrete Zeiteinheiten umgewandelt, die in einer Variablen vom Typ *long* gespeichert sind. Während

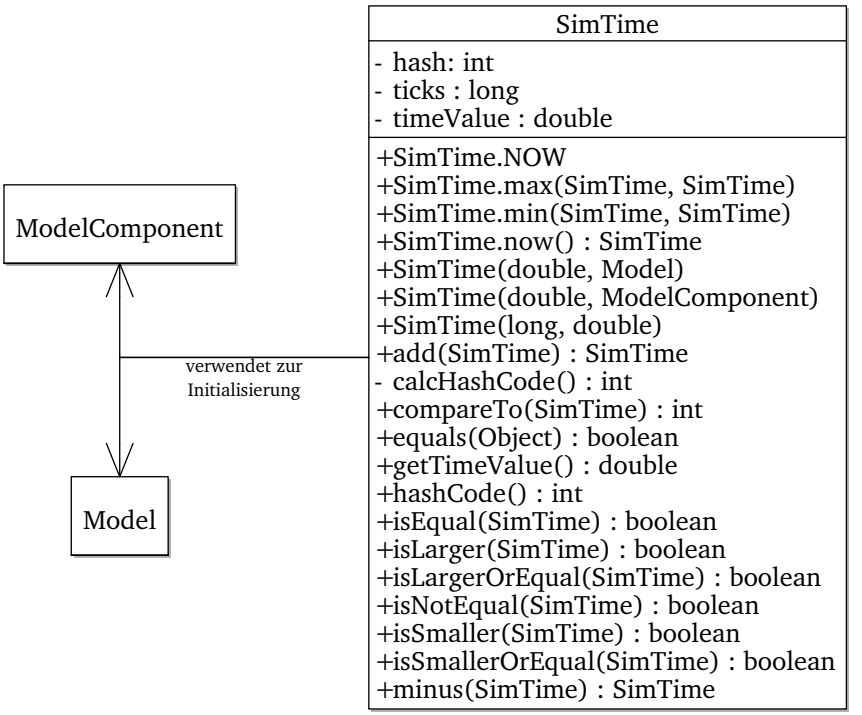


Abbildung 4.4: UML-Klassendiagramm *SimTime*

der Umwandlung wird die Diskretisierung der Fließkommazahl mit einer im Simulationsmodell vorgegebenen Genauigkeit durchgeführt. Die Konstruktoren wurden so verändert, daß immer die Angabe eines Modells oder einer Modellkomponente notwendig ist, um ein Objekt zu erzeugen. Ein *SimTime* Objekt wird immer unter Berücksichtigung der im Modell eingestellten Genauigkeit erzeugt. Die Genauigkeit des Simulationsmodells ist über die Anzahl an diskreten Zeiteinheiten je (kontinuierlicher) Simulationszeit definiert. Die interne Zeit entspricht der Anzahl an diskreten Zeiteinheiten bezogen auf den Zeitpunkt 0, an dem die Simulation beginnt.

Die Umwandlung wird vom Konstruktor vorgenommen, so daß im weiteren Gebrauch dieser Klasse nur noch mit der internen Darstellung

gearbeitet werden kann. Rundungsfehler durch nachfolgende Rechenoperationen können nicht mehr auftreten. Durch die Diskretisierung wird auch gleichzeitig die Genauigkeit für spätere Vergleiche zwischen zwei Zeitpunkten festgelegt. Bei einer Genauigkeit von δ besitzen alle Zustände in einem Intervall $[a, a+\delta[$ die gleiche interne Darstellung und damit einen identischen Inhalt.

Der Vorteil dieser Umwandlung liegt in der steuerbaren Genauigkeit. Der Benutzer trifft eine Abwägung zwischen der Genauigkeit der Zeit und der maximal darstellbaren Länge der Simulationsdauer. Beide sind beschränkt durch die Verwendung des *long* Datentyps, der eine Länge von 64 bit aufweist (von diesen 64 Bits sind nur 63 Bits einsetzbar, da ein Bit für das Vorzeichen benötigt wird, vgl. Sun 2005, Kap. 4.2.3). Diese sind auf die Anzahl an diskreten Zeiteinheiten und die maximale Länge der Simulation aufzuteilen. Der bisher verwendete Datentyp *double* nimmt diese Aufteilung implizit durch die interne Repräsentation vor. Eine Fließkommazahl besitzt eine Mantisse mit 53 bit, wovon ein Bit für das Vorzeichen verwendet wird, der Exponent umfaßt die restlichen 11 bit. Da die Länge der Mantisse konstant ist, steigt der Diskretisierungsfehler mit steigendem Exponenten. Die Genauigkeit, mit der sich eine Fließkommazahl bei einem Exponenten von 0 darstellt, läßt sich als absolute Größe nicht über den gesamten Bereich beibehalten, sondern sinkt mit steigendem Exponenten.

Der Wechsel der internen Darstellung auf ganzzahlige Größen ändert auch dieses Verhalten. Das Modell legt die Genauigkeit bzw. die maximale Auflösung fest. Somit steht auch die Anzahl an Bits fest, die benötigt wird, um eine Simulationseinheit darzustellen. Die restlichen Bits werden als Multiplikator verwendet und nicht als Exponent. Wenn z. B. je Simulationseinheit eine maximale Auflösung von 2^{40} benötigt wird, stehen $63 - 40 = 23$ bit als Multiplikator zur Verfügung. Die Simulationsdauer ist damit auf $2^{23} \approx 8,3 \cdot 10^6$ beschränkt. Der Benutzer kann so allein durch Kenntnis der maximalen Simulationsdauer festlegen, mit welcher Genauigkeit die Simulationszeit aufgelöst wird.

Die Auswirkungen der Diskretisierung auf die Vergleichsoperationen folgen unmittelbar aus der Anwendung im Design. Zwei Zeitpunkte werden jetzt als identisch erachtet, wenn die interne Zeit übereinstimmt.

Diese stimmt genau dann überein, wenn beide Zeitpunkte bei der Erzeugung innerhalb der vom Modell (vom Benutzer) vorgegebenen Genauigkeit liegen. Somit ist sichergestellt, daß während der gesamten Simulationszeit bei allen Vergleichen zwischen zwei Zeitpunkten mit der gleichen Genauigkeit gearbeitet wird. Fehler, die sich durch die Diskretisierung bei Fließkommazahlen ergeben, sind ausgeschlossen.

Zusammen mit den internen Änderungen wurde das Design dieser Klasse hin zu einem Value-Objekt entwickelt (vgl. Abschnitt 4.3 auf Seite 84) - die Klasse wurde in eine *immutable* Klasse verwandelt (Bloch 2008, S. 34). Dieses Refactoring ergab sich aus der Klassenstruktur und der Semantik einer Simulation. Die Zeit stellt eine eigene Größe dar, auf der Rechen- und Vergleichsoperationen durchgeführt werden. In der Klasse sind diese Methoden so implementiert, daß ein Aufruf eine neue Instanz zurückgibt ohne den Originalwert zu verändern (Bloch 2008, S. 73). Eine weitere nach außen sichtbare Änderung ist die Implementierung des *Comparable* Interface. Die natürliche Ordnung der Simulationszeit wird in aufsteigender Reihenfolge der Zeitpunkte definiert.

EventNote

EventNote Instanzen werden innerhalb des Kerns zur Bearbeitung von Ereignis-Entität Beziehungen verwendet. Der Scheduler erzeugt bei jeder Terminierung einer Entität automatisch eine *EventNote*, die in die Liste mit Ereignissen einsortiert wird. Die Sortierung erfolgt unter Berücksichtigung der Zeit und bei gleichen Zeitpunkten auch der Priorität des Ereignisses. Zur einheitlichen Verarbeitung wurde ein *Comparator* geschaffen, der in zwei Stufen die Beziehung zwischen zwei Ereignissen ermittelt (Bloch 2008, S. 62). Somit ist sichergestellt, daß jede Methode, die eine Ordnung zwischen zwei oder mehreren Ereignissen herstellt, auf dem gleichen Code basiert und die Relationen zwischen den Ereignissen im gesamten Programm konsistent bleiben.

Ein weiteres Refactoring betrifft die Sichtbarkeit und das Verhalten einer *EventNote* (vgl. Abbildung 4.5 auf der folgenden Seite). In der Originalimplementierung ist eine *EventNote* veränderlich und besitzt

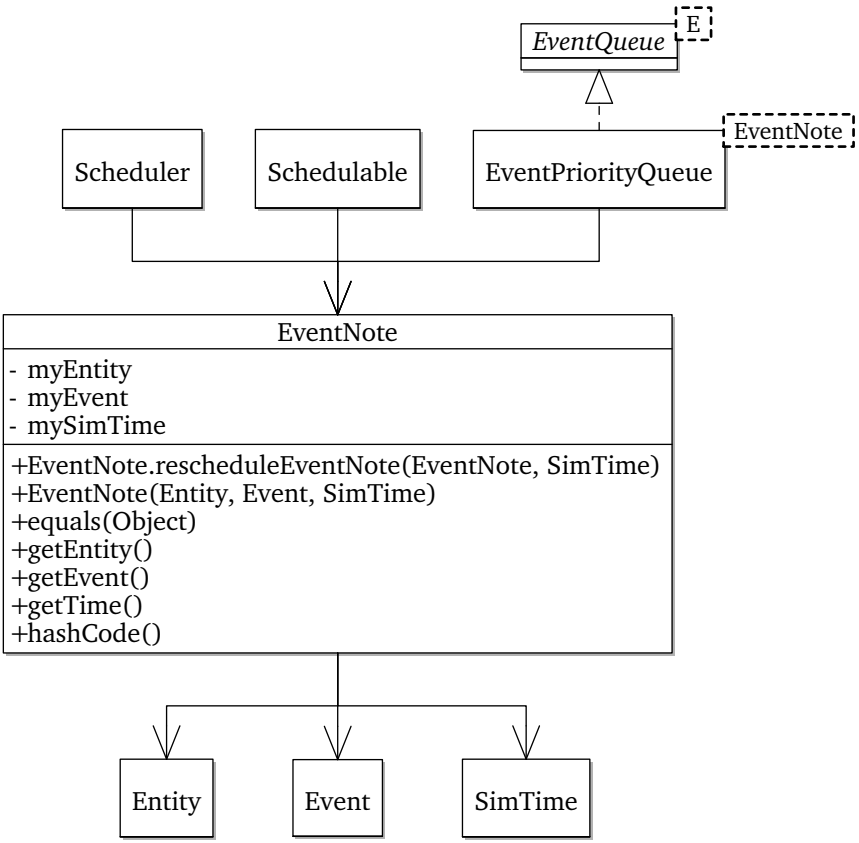


Abbildung 4.5: UML-Klassendiagramm *EventNote*

get- und set-Methoden (Bloch 2008, S. 71). Die set-Methoden werden eingesetzt, wenn ein geplantes Ereignis zu einem anderen Zeitpunkt ausgeführt werden soll. Die bestehende Instanz wird weiterverwendet und muß nicht neu erzeugt werden. Treten solche Verschiebungen häufig auf, wird der Garbage-Collector entlastet, da keine neuen Instanzen erzeugt wurden.

Da aufgrund der Struktur der zweiten Ebene die Verschiebung von Ereignissen nicht eingesetzt wird, wurde ein anderes Design umgesetzt. Die Klasse *EventNote* wird zu einem Value-Objekt weiterentwickelt, dessen Design auch gleichzeitig einer immutable-Klasse entspricht (vgl. Abschnitt 4.3 auf Seite 84 und Bloch 2008, S. 73). Durch den Verzicht auf die set-Methoden wurde auch gleichzeitig eine potentielle Fehlerquelle beseitigt, die durch unvorsichtigen Gebrauch die Integrität der gesamten Ereignisverwaltung stört. Durch dieses Redesign konnte die gesamte Logik der *Scheduler* Klasse stark vereinfacht werden. Jede Änderung an einer Entität-Ereignis Zuweisung ist jetzt automatisch mit der Erzeugung einer neuen Instanz und dem Löschen einer alten Instanz verbunden. Da dieses Prinzip durch die Konstruktion der Klasse nicht durchbrochen werden kann, ist die Konsistenz der Zuordnung Ereignis-Entität gewährleistet. Der Nachteil an diesem Design ist die Erzeugung einer zusätzlichen Instanz von *EventNote*, wenn ein Ereignis verschoben wird. Implementiert wird dieses Verhalten über eine statische Factory-Methode der Klasse. Im Vergleich zur Originalimplementierung wird durch die neue Instanz hier der Garbage-Collector zusätzlich belastet, was bei einer hohen Anzahl an Verschiebungen zu einer Geschwindigkeitseinbuße gegenüber der bisherigen Implementierung führt.

Scheduler

Der *Scheduler* ist das zentrale Element zur Verwaltung aller *EventNote* Instanzen in einem Modell. Jedes Modell besitzt genau eine Instanz des *Scheduler*, der alle Instanzen von *EventNote* verwaltet, die dem Modell zugeordnet sind (vgl. Abbildung 4.6 auf der nächsten Seite). Die Methoden, die der Scheduler bereitstellt, werden von *Entity* oder *Event* Objekten verwendet, um die Ereignisse zu terminieren.

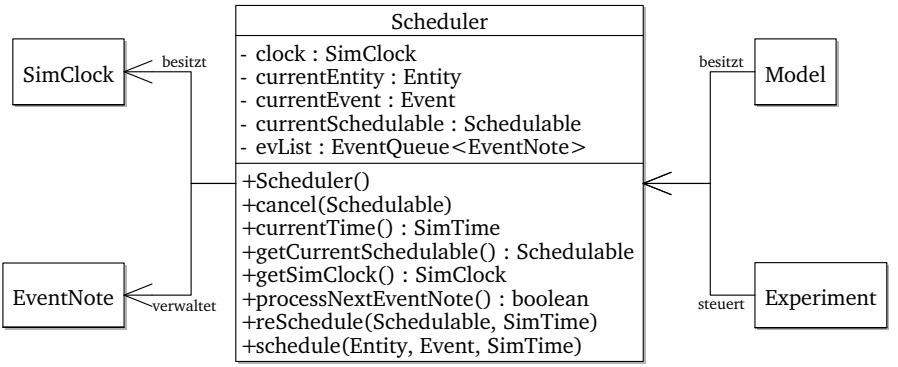


Abbildung 4.6: UML-Klassendiagramm *Scheduler*

Während das Modell die Infrastruktur enthält, übernimmt das zugehörige *Experiment* die Ablaufsteuerung, indem es *processNextEventNote* aufruft, bis alle Ereignisse abgearbeitet sind, oder der Zustand des Modells auf *STOPPED* gesetzt wird.

Eine weitere wesentliche Veränderung betrifft das interne Design und die Veränderungen der Schnittstelle aufgrund des Wegfalls der prozeßorientierten Simulation. Zur Verwaltung von Ereignissen und Prozessen ist es nötig, daß Ereignisse gezielt vor und nach einem anderen Ereignis oder einer Entität eingeplant werden. Dadurch entsteht die Notwendigkeit, daß die Ereignisliste die Reihenfolge beibehält, auch wenn andere Ereignisse eingeplant, verschoben oder entfernt werden. Zur Anwendung kommen hier Sortierverfahren, die stabil sein müssen, wie z. B. Mergesort oder in der aktuellen Implementierung von DESMO-J ein entsprechend angepaßter balancierter binärer Baum. Die Anforderung der Stabilität entfällt im Minimalsystem, da keine Reihenfolgebeziehungen zwischen Ereignissen eingehalten werden müssen. Sowohl der *Scheduler*, als auch alle Ereignisse und Entitäten enthalten keine Methoden, in denen die Reihenfolge bei der Terminierung von Ereignissen gefordert wird. Somit können auch Sortierverfahren eingesetzt werden, die instabil sind. Zur Auswahl stehen z. B. HeapSort (binärer Heap) oder ein angepaßter Fibonacci-Heap. Der Unter-

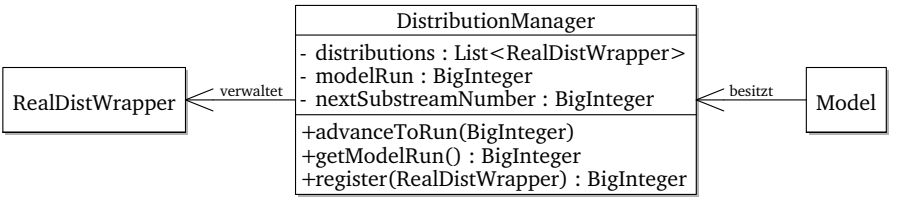


Abbildung 4.7: UML-Klassendiagramm *DistributionManager*

schied zwischen beiden liegt darin, daß HeapSort in-place arbeitet, während ein Fibonacci-Heap durch die zusätzlichen Datenstrukturen out-of-place arbeitet (Cormen u. a. 2007, S. 461,481ff). Eingesetzt wird ein HeapSort, der in der Java-Klassenbibliothek bereits in der Klasse *PriorityQueue* implementiert ist, so daß nur noch eine Adapter-Klasse zur Schnittstelle *EventQueue* erstellt wurde.

DistributionManager

Der *DistributionManager* verwaltet alle stochastischen Verteilungen in einem Modell. Er ist fester Bestandteil jedes Modells und wird von allen stochastischen Verteilungen benutzt, die in einem Modell erzeugt werden. Erzeugt wird er im Konstruktor des Modells, so daß er bereits von Anfang an zur Verfügung steht, auch wenn das Modell noch nicht an ein Experiment gebunden ist. Die Abbildung auf dieser Seite zeigt seine Einbettung innerhalb der Infrastruktur-Klassen.

Der *DistributionManager* vereinheitlicht die Verwaltung der Verteilungen in Zusammenarbeit mit einem Zufallszahlengenerator und einem umgebenden *Testrunner*. Als Zufallszahlengenerator wird der in Abschnitt 2.3 vorgestellte Generator von L'Ecuyer benutzt. Die Einteilung der Teilströme in drei Ebenen wurde übernommen. Der *DistributionManager* geht davon aus, daß der Zufallszahlengenerator mindestens zwei Ebenen an Teilströmen zur Verfügung stellt, was hier gegeben ist. Die erste Ebene ist für Modelle reserviert, die Zweite für die Einordnung der Verteilungen innerhalb eines Modells. Sobald eine Verteilung in einem Modell erzeugt wird, registriert sich diese automatisch über das

Modell im *DistributionManager*. Dieser teilt dem Modell eine eindeutige ID zu, die der Nummer des Teilstroms in der zweiten Ebenen bezogen auf den übergeordneten Strom der ersten Ebene entspricht.

Mit dem Start des Modells setzt der *DistributionManager* die Anfangsposition der ersten Ebene. Die Position entspricht der Nummer der Wiederholung, dem ersten Lauf wird Teilstrom 0 zugeordnet. Zu Beginn einer Wiederholung nehmen alle Verteilungen ihre Position relativ zum Teilstrom der ersten Ebene ein. Da dieser jetzt auf einer anderen Position steht als in allen anderen Läufen, verändern sich auch die Zufallszahlen, die von den Verteilungen benutzt werden. Auf diese Art können Wiederholungsläufe eines Modells durchgeführt werden, die jeweils unterschiedliche Ströme von Zufallszahlen benutzen. Da die Teilströme getrennt ansteuerbar sind, lassen sich auch Läufe gezielt wiederholen, bei denen z. B. Fehler aufgetreten sind. Auch die parallele Durchführung von Wiederholungsläufen ist möglich, da die Teilströme sich nicht überlappen und gezielt ansteuerbar sind. Entsprechende Methoden zum gleichzeitigen Zurücksetzen und zur Neupositionierung der Verteilungen aller Teilströme der ersten Ebene werden angeboten.

4.4.2 Modellkomponenten

ModelComponent und Schedulable

ModelComponent und *Schedulable* stellen den Ursprung der Klassenhierarchie für alle Modellkomponenten dar (vgl. Abbildung 4.1 auf Seite 93). *ModelComponent* ist eine allgemeine Komponente, die als abstrakte Klasse implementiert ist und die nur eine Referenz auf das mit ihr verbundene Modell enthält. Damit stellt sie auch gleichzeitig die Verbindung zur Infrastruktur des Modells her, wie z. B. den *Scheduler* oder *DistributionManager* (vgl. Abbildung 4.8 auf der gegenüberliegenden Seite). Direkte Nachkommen, die nicht zu *Schedulable* gehören, sind Bestandteile eines Modells, die keine Ereignisse verarbeiten und eine Verbindung zur Modell-Infrastruktur benötigen. Beispiele hierfür sind die Wrapper-Klassen um die stochastischen Verteilungen, da diese eine Verbindung zum *DistributionManager* verwenden.

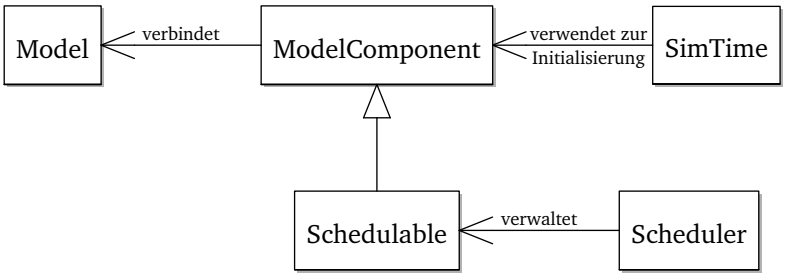


Abbildung 4.8: Einbindung von *ModelComponent* und *Schedulable* in die Infrastruktur

Schedulable stellt die Oberklasse aller Ereignisse und Entitäten in einem Modell dar. *Schedulable* ist eine abstrakte Klasse, die verschiedene Methoden zur Kommunikation mit der Infrastruktur, speziell mit dem *Scheduler* bereitstellt. Als einzige Klasse besitzt sie auch die Referenz auf eine Instanz von *EventNote*, so daß sie direkt vom *Scheduler* angesprochen werden kann. Der *Scheduler* muß daher keine Unterscheidung zwischen einem Ereignis oder einer Entität treffen, wenn ein Ereignis terminiert werden soll.

Die Struktur für beide Klassen wurde im Wesentlichen übernommen. Für beide wurde jedoch die Serialisierung eingeführt, die später zur Vervielfältigung von Modellen und Testläufen dient.

Entity

Ein *Entity* Objekt entspricht einer Entität innerhalb einer Simulation, wie sie von Page u. Kreutzer (2005, S. 24), Banks u. a. (2000, S. 68) und Law u. Kelton (2000, S. 11,207) definiert wurde. Eine Entität ist definiert als ein Objekt, welches eine explizite Repräsentation innerhalb des Simulationsmodells erfordert. Eine Entität besitzt einen internen Zustand, der durch Ereignisse verändert wird. Sie kann entweder das Ziel der Ereignisse sein, oder sie auslösen. Innerhalb von DESMO-J gilt folgende Einschränkung: Je Entität darf nur ein Ereignis in der Ereignisliste vorhanden sein (Page u. Kreutzer 2005, S. 268).

Event

Event ist eine abstrakte Oberklasse, von der konkrete Unterklassen abgeleitet werden können, die Ereignistypen repräsentieren. Jedes Ereignis muß an eine einzige Entität gebunden sein. In der Originalimplementierung ist keine Aussage darüber getroffen worden, wie sich das System verhalten soll, wenn zwei Ereignisse zum gleichen Zeitpunkt terminiert worden sind. Das Verhalten ist nicht exakt definiert, sondern hängt von der Sortierreihenfolge in der Ereignisliste des Schedulers ab.

Dieser nicht definierte Zustand ist problematisch, wenn Materialflüsse betrachtet werden sollen. Treffen z. B. in einem Lager zum gleichen Zeitpunkt eine Lieferung und eine Nachfrage in gleicher Höhe ein, so hängt es von der Reihenfolge der Bearbeitung ab, ob die Nachfrage erfüllt werden kann. Wenn die Nachfrage von einer Maschine kommt und nicht erfüllt werden kann, so kann die Produktion nicht starten. Die Simulation muß entweder abgebrochen werden oder die Produktion auf der Maschine wird verschoben, bis die Lieferung verbucht wurde. Die erste Alternative ist unerwünscht, die Zweite erfordert eine zusätzliche Logik, die noch weitere Grenzfälle verarbeiten muß. Um diese Grenzfälle bereits in der Simulation zu vermeiden, wurde eine zusätzliche Eigenschaft der Event-Klasse hinzugefügt.

In DESMO-J gibt es die Möglichkeit, Entitäten eine Priorität zuzuweisen, um die Sortierreihenfolge zu beeinflussen. Bei gleicher Simulationszeit werden Entitäten mit einer höheren Priorität vorgezogen. Durch ein Refactoring wurde diese Eigenschaft in die Ereignisse verlagert. Beide Designs sind äquivalent in den Auswirkungen, solange gewährleistet ist, daß zwischen Ereignis und Entität eine 1:1-Beziehung besteht. Die Übertragung der Priorität in die Ereignisse betont, daß die Handlungen (Ereignisse) in einer bestimmten Reihenfolge bearbeitet werden und nicht die Objekte (Entitäten).

Jedes Ereignis besitzt daher eine Priorität als zweites, nachgeordnetes, Sortierkriterium. Die Ereignisse werden vom *Scheduler* zuerst nach Zeitpunkten und bei identischen Zeitpunkten mit absteigender Priorität sortiert. Bei identischer Priorität und identischen Zeitpunkten ist die

Reihenfolge nicht definiert, d. h. sie hängt von der Implementierung des *Scheduler* ab und kann nicht garantiert werden.

Die Priorität kann entweder für jedes Ereignis oder für einen Ereignistyp innerhalb der Klasse definiert werden. Dieses Verfahren wird unter anderem im Rahmen der Materialflußsimulation eingesetzt, um an Knotenpunkten Lieferungen vor Nachfragen zu bearbeiten. Die Priorität ist eine ganze Zahl vom Typ *int* und besitzt einen Wertebereich von $[-2^{31}, 2^{31}-1]$ (Sun 2005, Kap. 4.2.1). Ohne Angabe einer Priorität wird eine neutrale Priorität von 0 benutzt.

ExternalEvent

In der Klasse *Event* wurde vereinbart, daß jedem Ereignis eine Entität zugeordnet sein muß. Die Klasse *ExternalEvent* ist in der Originalimplementierung davon ausgenommen, da sie ein vom Benutzer generiertes, von außen in das System eingeleitetes Ereignis darstellt. Da dieses nicht an eine Entität gekoppelt sein kann, sondern das gesamte Modell betrifft, wurde es auch nicht an eine Entität gekoppelt, sondern stattdessen ein *null* Wert verwendet. Dies hat zur Folge, daß bei allen Objekten, die mit Events arbeiten immer eine logische Abfrage durchgeführt werden muß, ob die Entität den Wert *null* hat. Dieses Verhalten führt in vielen Teilen zu unnötigen Verzweigungen und einem insgesamt komplizierteren Programmcode.

Durch Einsatz des Null-Objekt-Pattern (Kerievsky 2005, S. 327) vereinfacht sich die Logik erheblich. Die *NullEntity* Klasse ist ein Nachkomme von *Entity*, der als *final* deklariert wurde (vgl. Abbildung 4.1 auf Seite 93). Alle Methoden, welche aktiv den *Scheduler* für dieses Objekt aufrufen, wurden so überschrieben, daß sie eine Exception auslösen. Eine *NullEntity* ist auch nicht direkt instanziierbar, sondern muß über einen Aufruf der Factory-Methode *Entity.createNullEntity* erzeugt werden. Der Typ des Rückgabewertes der Factory wurde auf *Entity* festgelegt, so daß nach der Erzeugung einer Instanz von *NullEntity* dieses transparent als Ersatz für jedes *Entity* Objekt benutzt werden kann.

Durch die Unterbindung der aktiven Aufrufe wird es zu einem passiven Objekt, welches nur über ein korrespondierendes Event geplant

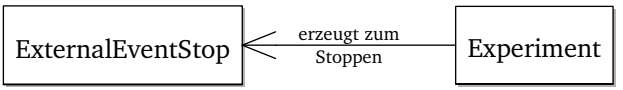


Abbildung 4.9: Einbindung von *ExternalEventStop*

werden kann. Für jedes externe Ereignis wird eine eigene Instanz von *NullEntity* erzeugt, so daß die Anforderung der Klasse *Event* (eine Entität je Ereignis) erfüllt wird. Da keine Ausnahmen für *null* Werte mehr berücksichtigt werden, vereinfacht sich die gesamte Bearbeitungslogik von *Scheduler*.

Eine weitere wichtige Modifikation betrifft das von der Klasse *ExternalEvent* abgeleitete Objekt ***ExternalEventStop***. Dieses Event gibt an, wann die Simulation gestoppt werden soll und wird vom Scheduler wie ein normales Ereignis behandelt. Probleme entstehen, wenn die Simulation zu einem Zeitpunkt stoppt, an dem noch andere zeitgleiche Ereignisse vorliegen, die abgearbeitet werden sollen. Das Verhalten der Simulation am Ende des Zeitraums ist in DESMO-J nicht exakt definiert.

Folgendes Verhalten für die Klasse *ExternalEventStop* wird neu definiert: Das Ereignis zum Beenden einer Simulation (*ExternalEventStop*) bekommt die niedrigste Priorität zugewiesen (-2^{31}). Somit ist sichergestellt, daß andere zeitgleiche Ereignisse mit einer höheren Priorität in jedem Fall ausgeführt werden, bevor die Simulation beendet wird.

Kapitel 5

Ebene 1 - Materialfluß

Aufbauend auf Ebene 0, die eine ereignisorientierte Basis zur Simulation zur Verfügung stellt, wurde Ebene 1 zur Simulation von Materialflüssen entwickelt (vgl. Tabelle 4.1 auf Seite 75). Das Ziel dieser Ebene ist die Bereitstellung von Elementen und Strukturen zur Simulation der Ergebnisse von Planungsprozessen im Bereich der Prozeßindustrie, nicht jedoch der Planungsprozesse selbst.

Durch den Einsatz in der Prozeßindustrie müssen im Gegensatz zur Stückgutproduktion kontinuierliche Materialflüsse berücksichtigt werden. Von den in Abschnitt 4.1 aufgeführten Simulationsbibliotheken können einige auch kontinuierliche Prozesse simulieren. Der klassische Ansatz zur Simulation von kontinuierlichen Elementen ist die Modellierung mit Hilfe von Differentialgleichungen (Murray-Smith 1995). Die Gleichungen beschreiben dazu den Zustand des Systems in Abhängigkeit von der Zeit und anderen Systemgrößen. Beispiele hierfür sind die Simulation von elektrischen Schaltungen. Der Nachteil von diesem Ansatz ist die benötigte Rechenzeit. Die Simulation wird durchgeführt, indem z. B. alle $\delta = 10^{-6}$ Zeiteinheiten die Simulation angehalten wird und der Zustand des Systems mit Hilfe der Differentialgleichungen neu berechnet wird. Die Simulation längerer Zeiträume erfordert daher eine hohe Anzahl an Neuberechnungen, was für die Simulation von mittelfristigen Planungsprozessen ungeeignet ist. Das Ziel war daher

die Entwicklung eines Algorithmus zur effizienten Berechnung von kontinuierlichen Produktionsprozessen, ohne den Einsatz von Differentialgleichungen.

Über das gesamte Kapitel werden Definitionen vorgestellt und erläutert, die zusammen das Verhalten der Simulation im Bereich der kontinuierlichen Materialflüsse bestimmen. In Abschnitt 5.5 sowie den folgenden Abschnitten werden die Unterschiede zur Simulation mit Differentialgleichungen vorgestellt. Es wird gezeigt, daß dieser neue Ansatz mit einer wesentlich geringeren Anzahl an Berechnungen auskommt, ohne dabei an Genauigkeit zu verlieren.

Das Kapitel gliedert sich in zwei Teile: In den Abschnitten 5.1 - 5.7 werden die allgemeinen Konzepte vorgestellt, um kontinuierliche Prozesse abzubilden. Dies erfolgt ohne auf eine spezielle Implementierung Bezug zu nehmen, sondern abstrakt für jede diskrete ereignisorientierte Simulationsbibliothek. Erst der letzte Abschnitt 5.8 enthält eine detaillierte Beschreibung der neu entwickelten Bausteine auf Basis von DESMO-J, der in Kapitel 4 ausgewählten Bibliothek für die Referenzimplementierung. Die Bausteine setzen die vorher entwickelten Definition und Verhaltensweisen um und dienen so als Referenzimplementierung für die Analyse einer hierarchischen Produktionsplanung in Kapitel 8.

5.1 Elementare Bausteine und Informationsstrukturen von Produktionssystemen

Die wesentlichen Elemente eines Produktionssystems sind das Material, die Lagerhaltung, die Ressourcen, die Erzeugnisstruktur und die Produktionsprozesse (Loos 1997, S. 87 ff.).

Das Material wird durch die diskreten oder kontinuierlichen Materialflüsse repräsentiert. Die von Loos (1997, S. 87) aufgeführten Eigenschaften des Materials (z. B. besondere Anforderungen an die Lagerung von Gefahrgütern) und Informationsstrukturen werden innerhalb dieser Ebene nicht berücksichtigt, da sie nur die Realisierung der Produktionspläne beinhaltet. Die Berücksichtigung der spezifischen Eigenschaften

des Materials ist eine Aufgabe der Planung und somit einer höheren Ebene zugehörig.

Die Lagerhaltung besitzt die Aufgabe, die Zeit zwischen Produktion, Beschaffung und Vertrieb zu überbrücken (Loos 1997, S. 113). Anhand der Eigenschaften des Materials werden die Lagerplätze aufgrund ihrer Eignung und Kapazitätsbeschränkungen (Volumen, Anzahl der Stellplätze) zugeordnet. Da diese Zuordnung einen dispositiven Charakter hat und der Planungsebene zuzuordnen ist, wird in der Simulation ein allgemeiner Lagerbaustein zur Verfügung gestellt, der als Grundlage für spätere Erweiterungen dienen kann (vgl. Abschnitt 5.8.4 auf Seite 175).

Loos (1997, S. 122) faßt unter dem Begriff „Ressource“ Anlagen aller Art zusammen. Es wird z. B. anhand der Verkettung zwischen Einzelstrang-, Mehrstrang- und Multipfadanlagen unterschieden (Loos 1997, S. 135). Diese Unterscheidung ist für die Simulation nicht wichtig, da Anlagenkomplexe aus einzelnen Anlagen aufgebaut werden können. Die Verrohrung zwischen den Anlagenteilen ist ebenfalls ein wesentliches Charakterelement einer Anlage. Diese kann entweder fest sein oder flexibel entsprechend der Planung neu festgelegt werden. Innerhalb der Simulation gibt es für die Verrohrung kein separates Element, sondern nur den Materialfluß in Form einer Kante zwischen zwei Knoten (Bausteinen). Jeder Knoten bietet daher die Möglichkeit, daß die Kanten auch während einer Simulation neu verbunden werden. Weitere Unterscheidungen im Bezug auf die Struktur, wie sie von Loos (1997, S. 138) getroffen werden, sind für die Simulation nicht nötig, da es sich um organisatorische Zuordnungen von Ressourcen zu Ressourcengruppen handelt.

Ein weiteres wesentliches Merkmal einer Ressource ist die Kapazitätsbeschränkung. Diese umfaßt sowohl die Leistungsfähigkeit der Ressource, als auch die zeitliche Verfügbarkeit aufgrund der Betriebszeiten, sowie geplanten und ungeplanten Unterbrechungen. Die Simulation bildet die Leistungsfähigkeit nicht direkt ab, sondern indirekt mit den Eigenschaften der auf der Ressource laufenden Rezepte (der Begriff der Rezepte wird in Abschnitt 5.8.1 auf Seite 143 näher erläutert). Die zeitliche Verfügbarkeit einer Ressource ist ebenfalls nicht direkt in der

Ressource implementiert, sondern kann nur auf der Planungsebene abgebildet werden.

Die Erzeugnisstruktur gibt an, aus welchen Bestandteilen ein Produkt besteht. Die Abbildung der Erzeugnisstruktur kann in der Prozeßindustrie auf unterschiedliche Arten erfolgen (Loos 1997, S. 149). Beispiele hierfür sind Stücklisten, Teileverwendungsnachweise, Erzeugnisstrukturbäume und Gozintographen. Alle aufgeführten Strukturen können sowohl ein- ,als auch mehrstufige Erzeugnisstrukturen abbilden. Die Abbildung einer mehrstufigen Produktion beinhaltet jedoch eine Koordination zwischen den einzelnen Fertigungsstufen, was einer Planungsaufgabe entspricht und somit nicht Bestandteil dieser Ebene sein kann. Die Informationsstrukturen zur Herstellung eines Produktes müssen sich auf dieser Ebene somit auf einstufige Strukturen beschränken. Diese werden zur Steuerung der Ressourcen eingesetzt. Die Steuerung erfolgt durch Angabe der Input- und Outputfaktoren, die eine Ressource während der Produktion eines Loses von ihren Vorgängerknoten und Nachfolgerknoten bezieht. Diese Darstellung entspricht einer Baukastenstückliste. Entsprechende Darstellungen der Stückliste und des damit verbundenen Gozintographen finden sich in Schneeweiß (2002, S. 206). Durch Kombination dieser einstufigen Beziehungen lassen sich auch mehrstufige und zyklische Erzeugnisstrukturen darstellen. Die Verwaltung dieser zusammengesetzten einstufigen Stücklisten hat auf einer höheren Ebenen zu erfolgen.

Durch die Beschränkung der Darstellungsformen der Simulationsbibliothek auf elementare Grundelemente wird eine hohe Flexibilität erreicht. Weitere Unterscheidungen, wie sie von Loos (1997, S. 156) getroffen werden, sind nicht notwendig. Aus den vorgestellten Basiselementen lassen sich auch kompliziertere Erzeugnisstrukturen wie z. B. Kuppelproduktion oder zyklische Strukturen erstellen. Weiterführende Erzeugnisstrukturen, wie z. B. die Abbildung von logischen Verknüpfungen Loos (1997, S. 160), lassen sich ebenfalls hiermit erstellen. Die dafür notwendige Steuerungslogik muß jedoch auf einer höheren Ebene implementiert werden.

5.2 Abbildung der Produktionsplanungsmodelle in der Simulation

Der Einsatzbereich der Simulationsbibliothek umfaßt die Simulation der Entscheidungen von Modellen zur Produktionsplanung. Diese Modelle liegen im Bereich der kurzfristigen Produktionsplanung, der Ablaufplanung.

Modelle und Heuristiken vom Typ Jobshop, Flowshop und Openshop werden ausführlich in Pinedo (2002, S. 33 ff) vorgestellt. Bei dieser Art von Modellen ist die Losgröße nicht Gegenstand der Entscheidungsvariablen, sondern gehört zur Menge der Eingabedaten. Andere Modelltypen planen gleichzeitig die Losgröße und die Reihenfolge, in der ein Auftrag bearbeitet werden soll. Als Grundmodelle dieses Typs gelten das Proportional Lotsizing and Scheduling Problem (PLSP) (Drexl u. Haase 1995) sowie das General Lotsizing and Scheduling Problem (GLSP), vgl. Fleischmann u. Meyr (1997). Weitere Modelle, die jedoch älter sind, und Vorläufer der bisher genannten Modelle sind das Discrete Lotsizing and Scheduling Problem (DLSP), vgl. Fleischmann (1990) sowie das Continuous Setup and Lotsizing Problem (CSLP) vgl. Karmarkar u. Schrage (1985, S. 328). Außer diesen einstufigen Grundmodellen gibt es Erweiterungen, die mehrstufige oder parallele Maschinenanordnungen und Erzeugnisstrukturen nachbilden. Andere Erweiterungen der Modelle wie periodenübergreifende Rüstzeiten (Suerie u. Stadtler 2003) oder Nachlieferungen (Pochet u. Wolsey 1988) betreffen die Planungsebenen und nicht die Simulationsebene, so daß sie in der Umsetzung keine Probleme bereiten.

Modelle und Heuristiken der Ablaufplanung haben zum Ziel, Aufträge an begrenzte Ressourcen im Zeitablauf zuzuweisen (Pinedo 2002, S. 1). Ergebnis einer Ablaufplanung ist eine Zuweisung von Aufträgen an eine oder mehrere Maschinen, sowie der Zeitpunkte, an denen die Aufträge an den Maschinen bearbeitet werden sollen. Die Planungsalgorithmen müssen hierzu die Erzeugnisstruktur, Lieferzeitpunkte und Kapazitäten berücksichtigen (für weitere Restriktionen siehe Pinedo 2002, S. 14). Das Ergebnis der Planung muß anschließend in der Anlage

umgesetzt werden. Dieser Prozeß ist eingebunden in eine rollierende Planung, die regelmäßig anhand der Umweltsituation überprüft, ob der Plan weiterhin so durchgeführt werden kann, oder ob ein neuer Plan erstellt werden muß. Mit Freistellung des Auftrags beginnt die Fertigung, den Auftrag zu verarbeiten. So ist sichergestellt, daß alle Kapazitätsbeschränkungen eingehalten werden und ausreichend Material vorhanden ist.

Die Simulationsbibliothek muß somit auf der Ebene der Materialflüsse nur freigegebene Aufträge bearbeiten können. Weiterhin wird angenommen, daß ein freigegebener Auftrag automatisch an die richtige Maschine weitergeleitet wird mit allen benötigten Informationen. Ein freigegebener Auftrag wird, sobald er an der Maschine eintrifft, unmittelbar produziert. Sind zwischen zwei Aufträgen Rüstvorgänge durchzuführen, so werden diese selbstständig von der Maschine ausgelöst. Es wird angenommen, daß die Planung die Zeit-/Kapazitätsverluste, die durch das Umrüsten entstehen, im Produktionsplan berücksichtigt hat.

Das mit der Fertigung verbundene Informationsobjekt ist das Produktionslos (oder kurz Los). Ein Produktionslos besteht aus einer Produktionsvorschrift, der zugewiesenen Maschine, der Produktionsmenge sowie der geplanten Startzeit. Das Produktionslos ist ein Ergebnis der Ablaufplanung und dient als Eingabe für die Ressourcen auf dieser Ebene. Im Unterschied zur Stückgutproduktion werden im Bereich der Prozeßindustrie keine einzelnen Stücke der gefertigten Produkte betrachtet, sondern nur ganze Produktionslose. Ein Produktionslos kann entweder eine beliebige Größe oder ein Vielfaches einer festen Losgröße annehmen, was als Batchproduktion bezeichnet wird. Entsprechend werden auch keine individuellen Einheiten eines Produktes durch das Produktionsnetzwerk geführt, sondern Produktionslose. Für die Simulation ist es dabei unerheblich, ob es sich um eine offene oder geschlossene Produktion handelt, da beide Fälle simuliert werden können. Weitere Informationen zu Aspekten der Modellierung sowie eine Klassifikation von Batchprozessen in der chemischen Industrie finden sich in Méndez u. a. (2006, S. 913).

Auf der Ebene der Materialflüsse werden nur die Aktionen umgesetzt, die sich aus der direkten Zuweisung der Lose zu den Ressourcen

ergeben. Die zeitliche Einordnung der Lose und deren rechtzeitige Weiterleitung an die Ressourcen ist eine Aufgabe, die Planungscharakter besitzt und damit nicht mehr auf dieser Ebene durchgeführt wird.

5.3 Erzeugnisstrukturen

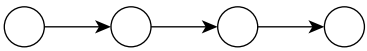
Zur Beschreibung der Erzeugnisstruktur wird ein gerichteter Graph verwendet, der das Verhältnis zwischen Input- und Outputfaktoren definiert. Im Bereich der ein- und mehrstufigen Fertigung wird zwischen folgenden elementaren Strukturtypen unterschieden (Dyckhoff 2006, S. 93):

- lineare, glatte oder durchgängige Produktion
- konvergierende oder synthetische Produktion
- divergierende oder analytische Produktion
- umgruppierende oder austauschende Produktion
- einstufige zyklische Produktion
- mehrstufige zyklische Produktion (Dyckhoff 2006, S. 104)

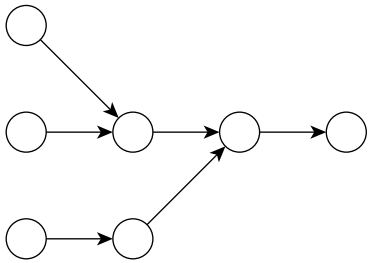
Beispiele für die einzelnen Strukturen zeigen die Abbildungen auf der nächsten Seite. Alle gezeigten und hieraus abgeleiteten Strukturen sollen simuliert werden können, auch wenn alle Kanten aus kontinuierlichen Materialflüssen bestehen.

5.4 Prozeßführung

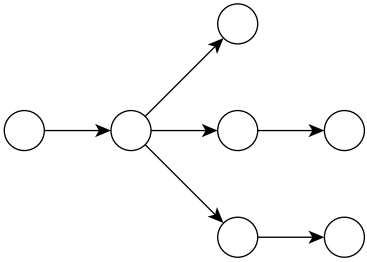
Das Verhalten eines Prozesses kann auch anhand der zugrundeliegenden Materialflüsse charakterisiert werden. Blömer (1999, S. 10) unterscheidet hier zwischen kontinuierlicher, semikontinuierlicher und diskontinuierlicher Prozeßführung bzw. Chargenproduktion (Batchproduktion).



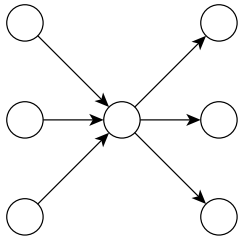
(a) Lineare Produktion



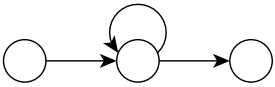
(b) Konvergierende Produktion



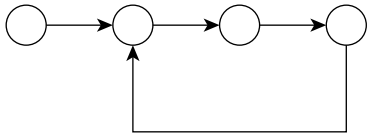
(c) Divergierende Produktion



(d) Umgruppierende Produktion



(e) Einstufige zyklische Produktion



(f) Mehrstufige zyklische Produktion

Abbildung 5.1: Erzeugnisstrukturen

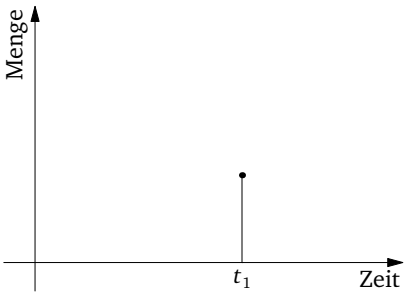
Die kontinuierliche Produktion ist dadurch gekennzeichnet, daß der Anlage das Material stetig zugeführt wird und die Anlage durchläuft, um schließlich einen kontinuierlichen Abfluß zu erzeugen. Im Gegensatz hierzu wird bei der Chargenproduktion das Material zum Beginn auf einmal in die Anlage (den sog. Reaktor) gefüllt, und am Ende der Produktion (der Reaktion) auch auf einmal abgeführt (Blömer 1999, S. 12). Zwischen beiden Formen liegt die semikontinuierliche Prozeßführung, bei der entweder zum Beginn oder am Ende die Bestandteile gemeinsam zugeführt oder die Produkte abgeführt werden.

Der Materialfluß dieser drei Grundformen der Prozeßführung läßt sich durch zwei elementare Bestandteile charakterisieren - diskrete und kontinuierliche Materialflüsse. Durch Kombination dieser beiden Bestandteile als Input- oder Outputfaktoren lassen sich alle bisher genannten Formen herleiten.

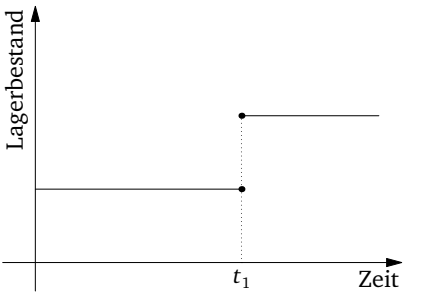
Eine kontinuierliche Produktion entspricht einer Produktion mit kontinuierlichen ein- und ausgehenden Materialflüssen. Die Chargenproduktion läßt sich durch diskrete ein- und ausgehende Materialflüsse abbilden. Die Modellierung von semikontinuierlichen Prozessen verläuft entsprechend. Das in dieser Simulationsbibliothek verwendete Modell der Rezepte (vgl. Abschnitt 5.8.1 auf Seite 143) erlaubt jedoch noch wesentlich flexiblere Modellierungen. So ist es möglich, daß einzelne Bestandteile, unabhängig davon ob sie als Input- oder Outputfaktoren dienen, kontinuierlich oder diskret sind. Damit kann bei einer Chargenproduktion gleichzeitig auch ein weiterer kontinuierlicher Inputfaktor simuliert werden, oder ein diskreter Outputfaktor bei einer kontinuierlichen Produktion (z. B. ein Katalysator, der nach der Produktion wieder zur Verfügung steht).

5.4.1 Diskrete Materialflüsse

Ein diskreter Materialfluß ist dadurch charakterisiert, daß er das Material zu einem bestimmten Zeitpunkt bereitstellt bzw. abruft. Die Materialbewegung findet unmittelbar, in einem einzigen Zeitpunkt statt. Die Menge, die bei einem diskreten Materialfluß bewegt wird, kann jeden beliebigen Wert aus dem Bereich der positiven reellen Zahlen



(a) Diskreter Materialfluß (Output)



(b) Bestandsverlauf bei einem eingehenden diskreten Materialfluß

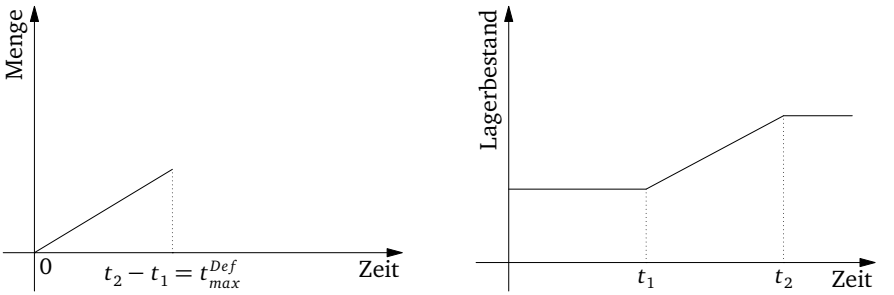
Abbildung 5.2: Beispiel für einen diskreten Materialfluß

annehmen. Ein diskreter Materialfluß kann überall dort eingesetzt werden, wo zum Beginn oder am Ende der Produktion eine feste Menge bewegt wird. Die entsprechende Fertigung wird auch als geschlossene Produktion klassifiziert (Bloech u. a. 2007, S. 266).

Ein Beispiel hierfür zeigt die Abbildung auf dieser Seite. Der diskrete Materialfluß findet ausschließlich zum Zeitpunkt t_1 statt. Der Lagerbestand erhöht sich im Zeitpunkt t_1 um den Betrag des diskreten Materialflusses. Dieses Verhalten ist nur zum Teil mit dem von Blömer (1999, S. 11) vorgestellten diskontinuierlichen Prozeßablauf vergleichbar. Dieser hat bei Blömer eine zeitliche Ausdehnung und ist nicht zeitpunktgeballt. Um den bei Blömer dargestellten Prozeßablauf zu modellieren, müssen stückweise definierte kontinuierliche Materialflußfunktionen verwendet werden. Der hier verwendete Ansatz stellt eine idealisierte Darstellung der in der Praxis auftretenden Materialflüsse dar.

5.4.2 Kontinuierliche Materialflüsse

Definition 1 (Materialflußfunktion)**.** *Eine Materialflußfunktion ist eine auf dem Intervall* $[0, t^{Def}_{max}], t^{Def}_{max} \in \mathbb{R}^+$ *explizit definierte monoton steigende Funktion* $f(t)$ *mit* $f(0) = 0$*. Der Funktionswert* $f(t)$ *entspricht der bis zu*



(a) Kontinuierlicher Materialfluß

(b) Bestandsverlauf bei einem kontinuierlichen Materialfluß

Abbildung 5.3: Beispiel für einen kontinuierlichen Materialfluß

diesem Zeitpunkt t auf der Kante transportierten Menge. Die Richtung des Transports stimmt mit der Richtung der Kante überein.

Die Abbildung auf dieser Seite zeigt ein Beispiel für einen kontinuierlichen Materialfluß. Der kontinuierliche Materialfluß wird über das Intervall $[0, t_{max}^{Def}], t_{max}^{Def} = t_2 - t_1$ berechnet, der Lagerbestand verändert sich entsprechend im Intervall $[t_1, t_2]$. Der wesentliche Unterschied - die zeitliche Dimension - zeigt sich im direkten Vergleich zu diskreten Materialflüssen. Die Veränderungen im Lagerbestand geschehen nicht mehr zeitpunktgeballt, sondern erstrecken sich über ein Zeitintervall. Der Startzeitpunkt t_1 wird auf den Punkt $t = 0$ der Materialflußfunktion abgebildet, so daß der Endpunkt bei $t_2 - t_1$ liegt. Die zeitlichen Auswirkungen der kontinuierlichen Materialflüsse entsprechen der von Blömer (1999, S. 11) definierten kontinuierlichen Prozeßführung. Die entsprechende Fertigung wird auch als offene Produktion klassifiziert (Bloech u. a. 2007, S. 266).

Die Darstellung der Materialflußfunktion muß explizit (d. h. in der Form $y = f(x)$) sein, da eine implizite Formulierung (in der Form $F(x,y) = 0$, vgl. Bronstein u. a. 1999, S,48) von der Simulationsbibliothek nicht unterstützt wird. An die Materialflußfunktion wird nur

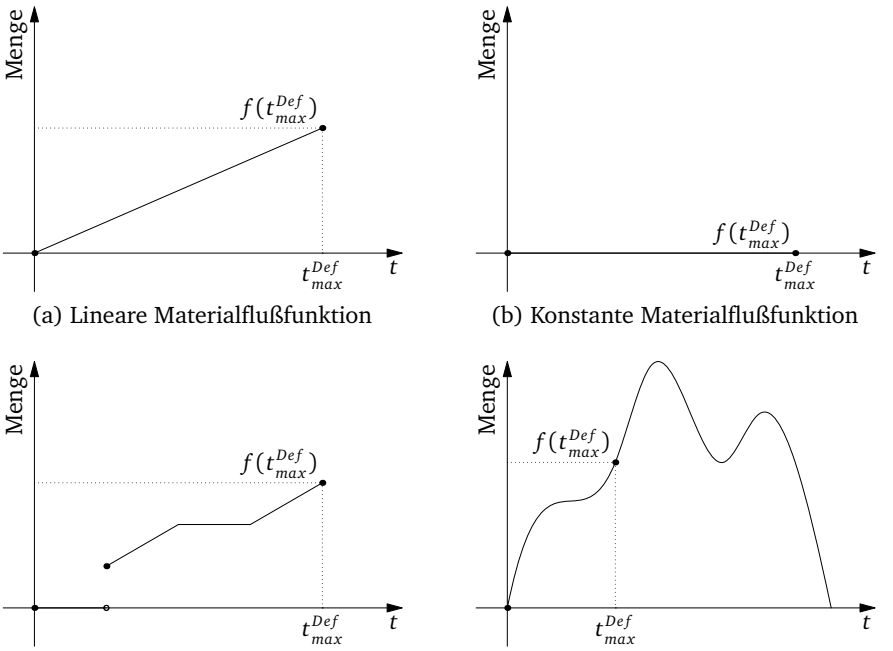
die Anforderung der Monotonie gestellt, sie muß nicht streng monoton sein und auch die Stetigkeit ist nicht erforderlich.

Die Abbildungen auf der nächsten Seite zeigen Beispiele für Materialflußfunktionen. Die erste Abbildung enthält eine klassische lineare Materialflußfunktion, wie sie in Losgrößenmodellen eingesetzt wird. Die zweite Abbildung zeigt eine konstante Materialflußfunktion, die mit der x-Achse zusammenfällt. In diesem Fall wird kein Material bewegt. Konstante Materialflußfunktionen treten z. B. bei der Berechnung von Netto-Materialflüssen auf (vgl. Abschnitt 5.7.2 auf Seite 140).

Einen weiteren Sonderfall zeigt Abbildung 5.4c, eine abschnittsweise definierte Funktion. Im ersten Abschnitt fällt sie mit der x-Achse zusammen, in den folgenden Abschnitten entspricht sie jeweils einer linearen Funktion. Die Funktion ist unstetig, aber monoton und somit eine zulässige Materialflußfunktion. Implementierungen, die abschnittsweise definierte Funktionen erlauben, müssen dies auch bei den in den folgenden Abschnitten beschriebenen Berechnungen der Grenzzustände berücksichtigen.

Das letzte Beispiel zeigt eine innerhalb des Definitionsbereiches monoton steigende Funktion. Außerhalb des Bereiches verliert sie diese Eigenschaft. Funktionen dieser Art können auftreten, wenn nichtlineare Funktionen durch Taylor-Reihen approximiert werden. Durch den Einsatz von Polynomen höheren Grades ergeben sich diese Verläufe, die jedoch innerhalb des Definitionsbereiches eine zulässige Materialflußfunktion darstellen.

Die Definition der Materialflußfunktion wurde anhand der in Abschnitt 5.2 vorgestellten Modelle entwickelt. Eine gemeinsame Eigenschaft der dort vorgestellten Losgrößenmodelle ist ihr Modelltyp. Es handelt sich um gemischt-ganzzahlige Modelle der linearen Programmierung. Alle Modelle basieren auf der Eigenschaft, daß die Produktionsmenge proportional zur Produktionszeit ist. Die Produktionsmenge ist von keiner anderen Größe im Modell abhängig, so daß sie sich immer als lineare Funktion der Zeit darstellen läßt. Dieser funktionale Zusammenhang läßt sich aus den Losgrößenmodellen nur indirekt ablesen. Fleischmann (1990, S. 339) definiert eine produktspezifische konstante Produktionsrate, welche mit der Produktionsmenge multipliziert wird



(a) Lineare Materialflußfunktion

(b) Konstante Materialflußfunktion

(c) Abschnittsweise definierte Materialflußfunktion

(d) Im Definitionsbereich monoton steigende Materialflußfunktion

Abbildung 5.4: Beispiele für Materialflußfunktionen

und geringer sein muß als die verfügbare Kapazität. Die Kapazität ist durch die verfügbare Anzahl an Maschinenstunden je Periode definiert. Die gleiche Formulierung findet sich auch in Drexl u. Haase (1995, S. 75), Fleischmann u. Meyr (1997, S. 12). Diese Verbindung zwischen der verbrauchten und der verfügbaren Kapazität ist eine Standardformulierung im Bereich der Losgrößenmodelle, die implizit annimmt, daß die Produktionsgeschwindigkeit innerhalb des Planungszeitraums konstant bleibt und sich nur zwischen den Produkten und Maschinen unterscheidet.

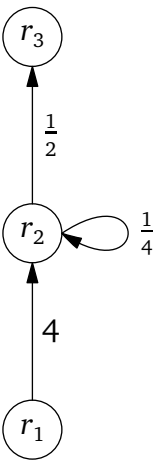
Der Einsatz einer linearen Funktion ist für die Standard-Losgrößenmodelle ausreichend, würde den zukünftigen Anwendungsbereich jedoch zu sehr einschränken. Daher wurde die Definition auf die Klasse der monoton steigenden Funktionen erweitert. Anwendungen von nichtlinearen Materialflüssen stellt Chubanov u. a. (2008, S. 878) vor, durch Erweiterung des Standard-Losgrößenmodells auf eine nicht gleichförmige Auslastung der Maschine.

5.5 Auflösung von Zyklen

Es wurde bereits erwähnt, daß die Erzeugnisstrukturen auch ein- und mehrstufige Zyklen enthalten können. Die Berechnung der Produktionsmengen in einem solchen Zyklus ist umfangreicher als bei zyklenfreien Strukturen (Dyckhoff 2006, S. 104). Entsprechend ist auch die Berechnung der Materialbestände komplizierter, wenn kontinuierliche Materialflüsse vorhanden sind.

Das Problem der algebraischen Zyklen

Das Problem der Zyklen in Erzeugnisstrukturen ist vergleichbar mit dem Problem der algebraischen Zyklen in der kontinuierlichen Simulation (Cellier 1991, S. 25). Das Problem der algebraischen Zyklen tritt auf, wenn zwei Variablen voneinander abhängig sind und gemeinsam berechnet werden müssen. Eine Methode diese Zyklen aufzulösen ist die Umstellung der Gleichungen, so daß sie sequentiell bearbeitet werden



Berechnung der Sekundärbedarfe bei einem Primärbedarf von $r_3 = 30$

$$r_1 = 4r_2$$
$$r_2 = \frac{1}{2}r_3 + \frac{1}{4}r_2 \Rightarrow \overrightarrow{r} = \begin{pmatrix} 0 & 4 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} \\ 0 & 0 & 0 \end{pmatrix} \overrightarrow{r} + \begin{pmatrix} 0 \\ 0 \\ 30 \end{pmatrix}$$
$$r_3 = 30$$

$$\overrightarrow{r} = \begin{pmatrix} 1 & \frac{16}{3} & \frac{8}{3} \\ 0 & \frac{4}{3} & \frac{2}{3} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 30 \end{pmatrix} = \begin{pmatrix} 80 \\ 20 \\ 30 \end{pmatrix}$$

Abbildung 5.5: Beispiel für die Berechnung der Produktionsmengen bei einem Zyklus in der Erzeugnisstruktur

können. Dies ist jedoch nicht für alle Gleichungssysteme möglich, so daß diese Alternative im allgemeinen Fall nicht anwendbar ist.

Die Abhängigkeit kann anhand des Beispiels in Abbildung 5.5 gezeigt werden. Die Bedarfsauflösung von $r_3 = 30$ führt zunächst zu einem direkten Bedarf von $r_2 = 15$. Damit ist jedoch nicht der gesamte Bedarf an r_2 gedeckt, da zur Herstellung von r_2 noch einen Anteil an r_2 notwendig ist. Die Bedarfsmengen in diesem einstufigen Zyklus lassen sich durch Auflösung der Produktionsgleichung mit Hilfe der Direktbedarfsmatrix lösen (Schneeweiß 2002, S. 51). Das Beispiel zeigt, daß bereits die Berechnung von Produktionsmengen im statischen Fall durch die Existenz von Zyklen wesentlich aufwendiger ist, als eine stufenweise Bedarfsauflösung.

Eine weitere Methode zur Lösung von algebraischen Zyklen ist das Aufbrechen der Zyklen. Ausgehend von einem Startpunkt in der Berechnung wird am letzten Punkt vor dem Schließen des Zyklus eine Hilfsvariable hinzugefügt. Die Berechnung wird anschließend sequentiell durchgeführt und der Wert in der Hilfsvariablen nach jeder Berechnung angepaßt. Durch mehrfache Wiederholung wird der berechnete

Wert an den tatsächlichen Wert mit einer vorgegebenen Genauigkeit approximiert (auch als „Implicit Loop Solver“ bezeichnet, vgl. Cellier 1991, S. 30). Dieser iterative Prozeß muß bei jeder Zustandsänderung durchgeführt werden, was entsprechend rechenintensiv ist.

Berechnung von Materialflüssen

Durch die Beschränkung der Materialflußfunktionen auf eine zeitliche Abhängigkeit kann jedoch ein anderes Verfahren angewendet werden, welches ähnlich arbeitet wie ein „Implicit Loop Solver“, jedoch eine wesentlich geringere Rechenzeit benötigt. Der Algorithmus ist in den Abbildungen 5.6 bis 5.7 auf den Seiten 130–131 dargestellt.

Gegeben sei eine Erzeugnisstruktur, die durch einen gerichteten Graphen dargestellt wird. Der Graph kann sowohl ein- ,als auch mehrstufige Zyklen enthalten. An einem Knoten tritt eine Zustandsänderung ein, oder der Zustand wird abgefragt (vgl. Definition 2). Ausgehend von diesem Startknoten k werden rekursiv alle Vorgänger $V(k)$ und Nachfolger $N(k)$ besucht, bis alle Kanten im Graphen besucht wurden. Knoten, die bereits besucht worden sind, werden nicht noch einmal besucht. Bei jedem Besuch einer Kante wird der Status aktualisiert, d. h. der Materialfluß zwischen der letzten Aktualisierung und dem aktuellen Zeitpunkt wird berechnet und unmittelbar unter Umgehung der ereignisbasierten Methoden ausgeliefert bzw. angefordert. Der Zeitpunkt der letzten Aktualisierung definiert den internen Zustand eines kontinuierlichen Materialflusses. Dieser wird mit jedem Durchlauf aktualisiert. So ist sichergestellt, daß nur die Zeitdifferenz zwischen dem letzten Durchlauf und dem aktuellen Zeitpunkt in die Berechnung der Menge einbezogen wird. Ohne diese Zustandsfortschreibung würden die Mengen bei nachfolgenden Aufrufen mehrfach berechnet werden.

Enthält eine Kante keinen kontinuierlichen, sondern einen diskreten Materialfluß, so wird sie nicht berücksichtigt. Befindet sich der Algorithmus am Zielknoten des diskreten Materialflusses, so wird der Vorgängerknoten nicht besucht. Gleiches gilt für den symmetrischen Fall des Startknotens. Der Grund hierfür ist die Eigenschaft eines diskreten

Materialflusses, daß er zustandslos ist. Der Materialfluß findet ohne Zeitverzögerung zum Beginn oder am Ende der Produktion statt.

Das frühzeitige Abbrechen der Rekursion bei diskreten Materialflüssen kann bei entsprechenden Erzeugnisstrukturen auch die Anzahl an Kanten verringern, die für eine Neuberechnung durchlaufen werden müssen. Abschnitte der Struktur, die durch keine aktiven Materialflüsse verbunden sind oder nur durch diskrete Materialflüsse werden automatisch isoliert und für die Neuberechnung ignoriert.

Das Verhalten der Knoten im Bezug auf die unmittelbar eingehenden Materiallieferungen/-anforderungen hängt von der Art des Knotens ab. Quellen und Senken können aufgrund ihrer unbegrenzt vorhandenen Kapazität jede Anforderung und Lieferung sofort annehmen. Ressourcen hingegen benötigen Inputfaktoren zur Herstellung der Outputfaktoren. Da sie keine Speicherfunktion besitzen, führen sie nur eine Aktualisierung aller kontinuierlichen ein- und ausgehenden Ströme durch.

Das Verhalten der Lagerknoten ist entscheidend für das Aufbrechen der Zyklen. Lagerknoten besitzen die Eigenschaft, daß sie Material speichern können. Die Speicherfunktion ist durch Ober- und Untergrenzen beschränkt, die während einer Simulation nicht verletzt werden dürfen. Die ereignisorientierten Methoden zur Materiallieferung und -anforderung berücksichtigen diese Beschränkungen und reagieren auf Verletzungen. Während einer Aktualisierung der Materialflüsse hingegen ist dieses Verhalten nicht erwünscht, da innerhalb der Rekursion Materialüberschüsse oder -fehlmengen entstehen können. Diese Differenzen entstehen aufgrund der Bearbeitungsreihenfolge und sind temporärer Natur, da nach Abschluß aller Berechnungen ein Saldo erstellt wird und dieser Saldo anschließend mit dem Lagerbestand endgültig verrechnet wird. Erst bei diesem letzten Berechnungsschritt werden die oberen und unteren Grenzen überprüft und bei einer Über- oder Unterschreitung entsprechende Aktionen ausgelöst. Die Reihenfolge der Schritte Materialfluß-Auslieferung und Materialfluß-Anfrage ist willkürlich gewählt worden und kann auch in umgekehrter Folge ausgeführt werden.

Mit Hilfe dieses Algorithmus lassen sich alle Erzeugnisstrukturen,

Zustandsaktualisierung

	Parameter: k {Startknoten}	
	Rekursiver Durchlauf mit Knoten k - Durchführung aller Materialfluß-Anfragen	
	Rekursiver Durchlauf mit Knoten k - Durchführung aller Materialfluß-Auslieferungen	
	Rekursiver Durchlauf mit Knoten k - Saldierung aller Bedarfe mit den Lagerbeständen	

Abbildung 5.6: Algorithmus „Aktualisierung der Zustände“

die in Abschnitt 5.3 vorgestellt worden sind, simulieren, auch wenn alle Kanten aus kontinuierlichen Materialflüssen bestehen. Somit gibt es für die Simulation keinerlei Einschränkungen bezüglich der Struktur des Produktionsnetzwerkes. Im Gegensatz zu einer Berechnung mit Differentialgleichungen lassen sich die Materialbewegungen zu einem beliebigen Zeitpunkt berechnen, ohne daß weitere Berechnungen seit dem Zeitpunkt der letzten Zustandsänderung notwendig sind.

Ein numerisches Beispiel ist in den Abbildungen auf Seite 134 dargestellt. Die Produktionsstruktur entspricht dem Beispiel aus Abbildung 5.5 auf Seite 127, ergänzt um Zwischenlager zwischen den Knoten. Es wird angenommen, daß die Lager auf den Kanten (r_1, r_2) und (r_2, r_3) auch in der Produktion vorhanden sind. Der Zyklus in der Produktionsstruktur wurde ebenfalls um eine Lager ergänzt, daß in der Realität nicht existiert, aufgrund der Anforderungen an die Kombination von Knoten aber eingefügt werden mußte (vgl. Abschnitt 5.8.4 auf Seite 161). Die obere und untere Grenze dieses Zwischenlagers liegt bei Null, d. h. es darf zu keinem Zeitpunkt einen Lagerbestand enthalten.

Abbildung 5.8a auf Seite 133 zeigt die gesamte Produktionsstruktur, Abbildung 5.8b den Ausschnitt, der im folgenden Beispiel betrachtet wird. Es wird angenommen, daß auf allen Kanten ein kontinuierlicher Materialfluß aktiv ist. Die Materialflußfunktionen wurden so gewählt,

Rekursiver Durchlauf

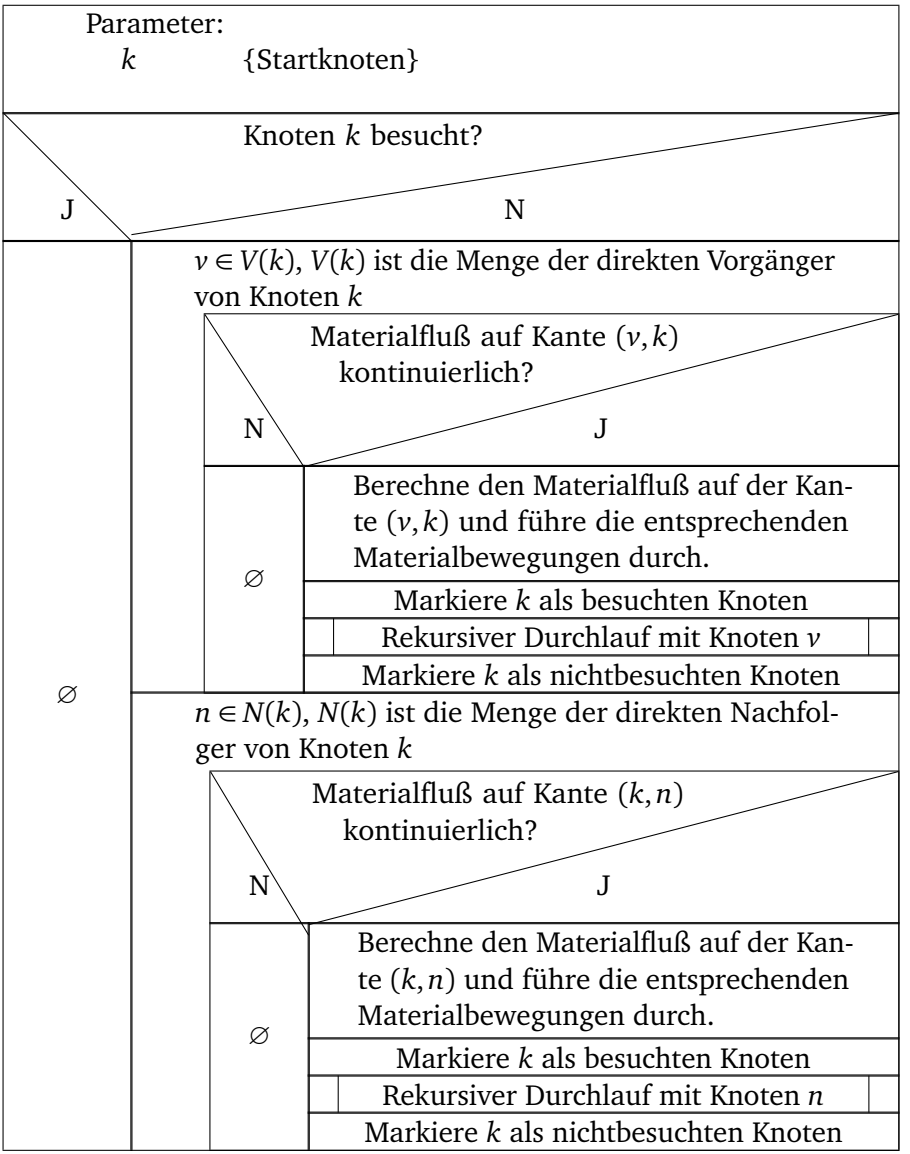


Abbildung 5.7: Algorithmus „Berechnung der Materialflüsse in Erzeugnisstrukturen mit Zyklen“

daß die in Abbildung 5.5 auf Seite 127 berechneten Mengen nach einer Produktionsdauer von $t = 5$ erreicht werden. Die Lageranfangsbestände in allen Bauteilen sollen nicht berücksichtigt werden, da nur eine Betrachtung der Materialflüsse erfolgt. Die Abbildungen enthalten nur die wichtigsten Schritte des Algorithmus. Die Teilschritte, in denen das Netzwerk durchlaufen wird und keine Berechnung von Materialflüssen erfolgt (z. B. weil ein Knoten bereits besucht wurde), sind nicht aufgeführt.

Das Beispiel in Abbildung 5.9 zeigt den Zustand des Systems im Zeitpunkt $t = 2$. Die Aktualisierung erfolgt im Lager am Ende der Produktionsstruktur. Über die aktive Kante wird der Vorgängerknoten, die Ressource erreicht. Die Ressource aktualisiert alle eingehenden Materialflüsse und erreicht das Zwischenlager, welches in die zyklische Verbindung eingefügt wurde (vgl. Abbildung 5.9a). Der davon in die Ressource eingehende Materialfluß wird berechnet und mit den temporären Lagerbeständen verrechnet.

Im zweiten Schritt (vgl. Abbildung 5.9b) erreicht der Algorithmus vom Lager aus wieder die Ressource und beendet den Teil der Verzweigung, da die Ressource bereits besucht wurde. Ausgehend von der Ressource wird der anderen Vorgängerknoten - das vorgelagerte Lager - erreicht und auch hier der Materialfluß berechnet und mit den temporären Lagerbeständen saldiert. Damit ist der rekursive Durchlauf in diesem Teil der Anlage beendet und die Materialauslieferungen können erfolgen.

Der dritte Schritt durchläuft wieder ausgehend vom ersten Lager das Netzwerk und erreicht die Ressource. Der ausgehende Materialfluß wird berechnet und mit dem temporären Lagerbestand verrechnet (vgl. Abbildung 5.9c). Der zweite ausgehende Materialfluß - zum Zwischenlager innerhalb des Zyklus - wird im vierten Schritt in Abbildung 5.9d berechnet. Der Lagerbestand des Zwischenlagers innerhalb des Zyklus entspricht jetzt genau 0, so daß die Annahmen über das Zwischenlager erfüllt sind. Die anderen Lagerbestände in Höhe von 12 und -32 entsprechen genau 40% ($t = 2$ bei einer Produktionsdauer von $t = 5$) der im statischen Fall berechneten Produktionsmengen (vgl. Abbildung 5.5 auf Seite 127). Sie werden mit den bestehenden

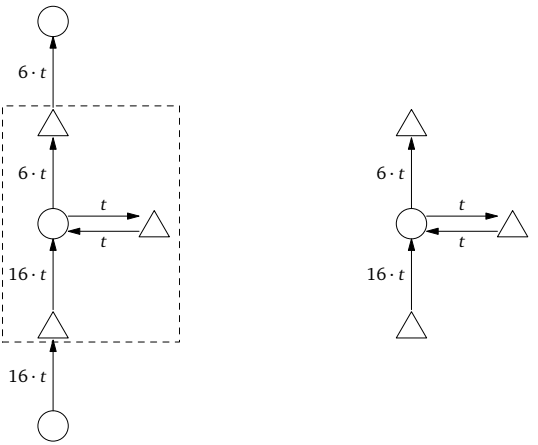


Abbildung 5.8: Produktionsstruktur des Beispiels zur Berechnung der Materialflüsse in zyklischen Strukturen

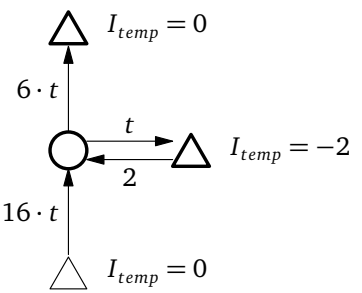
Lagerbeständen in einem letzten Schritt verrechnet. Die dabei zu berücksichtigenden Grenzzustände aufgrund der Ober-/Untergrenzen der Lager werden in Abschnitt 5.7 auf Seite 137 beschrieben.

5.6 Zustandsaktualisierung

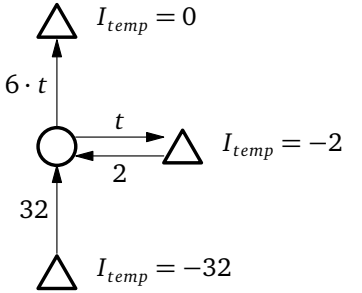
Bei den in Abschnitt 5.5 vorgestellten Konzepten und Algorithmen wurde gezeigt, daß die Berechnung der Materialflüsse zu jedem Zeitpunkt durchgeführt werden kann, ohne daß iterative Berechnungen mit Differentialgleichungen oder anderen Methoden notwendig sind. Bisher wurde jedoch noch nicht definiert, wann diese Berechnungen durchgeführt werden müssen.

Definition 2 (Aktualisierung von Materialflüssen)**.** *Die Neuberechnung der Materialflüsse muß zu jedem Zeitpunkt durchgeführt werden, an dem Informationen über den Zustand eines Knotens abgefragt werden,*

Ressourcen aktualisieren eingehende Materialflüsse

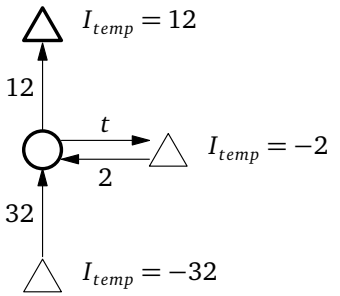


(a)

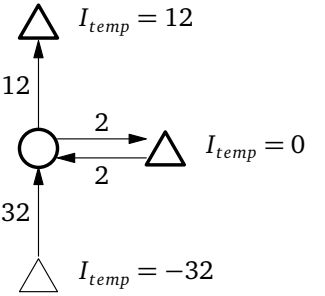


(b)

Ressourcen aktualisieren ausgehende Materialflüsse



(c)



(d)

Abbildung 5.9: Beispiel zur Berechnung der Materialflüsse in zyklischen Strukturen

der Quelle oder Ziel eines kontinuierlichen Materialflusses ist und dessen interner Zustand von der Materialflußfunktion abhängig ist.

Daraus lassen sich folgende zwei Gruppen von Zeitpunkten ableiten, an denen die Zustände neu berechnet werden müssen:

1. Zustandsverändernde Ereignisse
    - bei Beginn einer Produktion
    - am Ende der Produktion
    - nachdem ein Bedarf für ein Produkt auftritt
    - vor der Auslieferung von Produkten, wenn diese durch ein anderes Ereignis ausgelöst wurde
2. Abfrage statistischer Informationen
    - Abfrage eines Lagerbestandes
    - Abfrage der ausgelieferten Mengen einer Quelle
    - Abfrage der angenommenen Mengen einer Senke

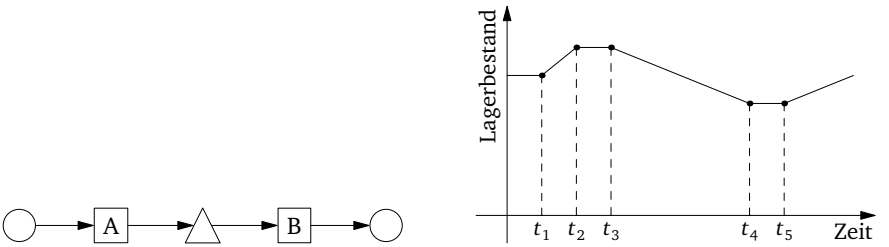
Die Ursache der Neuberechnung in der ersten Gruppe liegt in einer Änderung des internen Zustands eines Bausteins, der Auswirkungen auf Materialflüsse hat, die mit dem Baustein verbunden sind. Hierzu gehören der Beginn und das Ende einer Produktion, da zu diesen Zeitpunkten ein Materialfluß beginnt oder endet. Nachdem ein Bedarf für ein Produkt an einen Lagerbaustein auftritt, wird dessen interner Lagerbestand oder Auftragsbestand verändert, so daß die Grenzzustände neu berechnet werden müssen (vgl. Abschnitt 5.7 auf Seite 137). Der letzte Fall tritt ein, wenn ein Produkt zu einem späteren Zeitpunkt nachgeliefert werden soll und ein kontinuierlicher Materialfluß in ein Lager hineingeht. Zu dem Zeitpunkt, an dem der Lagerbestand die nachzuliefernde Menge erreicht, muß die Auslieferung erfolgen. Hierzu ist jedoch zunächst der Lagerbestand zu aktualisieren, was wiederum die Grenzzustände beeinflußt.

Neuberechnungen bei Ereignissen aus der zweiten Gruppe sind notwendig, da zu diesen Zeitpunkten der aktuelle Zustand des Systems

benötigt wird. Bei Lagerbausteinen betrifft dies alle Anfragen zu dem Lagerbestand eines Produktes. Anders verhält es sich bei Quellen und Senken. Da diese Ausgangs- oder Endpunkt von Materialflüssen sind und statistische Informationen sammeln, muß eine Neuberechnung durchgeführt werden, sobald Informationen an den Bausteinen abgefragt werden.

Die Abbildung auf der gegenüberliegenden Seite zeigt ein Beispiel für die Aktualisierung durch Zustandsänderungen. Die Erzeugnisstruktur ist linear, die Produktion besteht aus einer Quelle, einer Senke, dazwischen die Ressourcen A und B, sowie ein Lager. Bis zum Zeitpunkt t_1 ist keine Maschine aktiv und der Lagerbestand bleibt konstant. Im Zeitpunkt t_1 wird die Produktion auf Maschine A gestartet, so daß der Lagerbestand kontinuierlich steigt. Zwischen t_2 und t_3 findet keine Produktion statt, so daß der Lagerbestand konstant bleibt. Ab t_3 beginnt auf Maschine B die Produktion, so daß der Lagerbestand sinkt bis zum Zeitpunkt t_4. Ab Zeitpunkt t_5 sind beide Maschinen aktiv, so daß hier ein Nettozufluß erzeugt wird.

Eine Neuberechnung der Zustände ist nur an den Zeitpunkten t_1 bis t_5 notwendig, unabhängig von der Länge der Intervalle zwischen den Zeitpunkten. Der Zustand des Systems zwischen diesen Zeitpunkten ist durch die Materialflußfunktionen definiert, so daß bis zur nächsten Zustandsänderung der Bisherige fortgeschrieben werden kann. Erst zum Zeitpunkt einer Änderung hat eine Neuberechnung zu erfolgen. Weitere Berechnungen außerhalb der angegebenen Zeitpunkte sind nötig, wenn zu einem Zeitpunkt zwischen den Intervallgrenzen Informationen über Lagerbestände oder allgemein über Materialbewegungen abgefragt werden. Im Gegensatz hierzu ist die Anzahl an Neuberechnungen bei Verwendung von Differentialgleichungen von der gewählten Auflösung und dem betrachteten Zeitrahmen abhängig. In diesem Bereich der Simulation ist das Verfahren den alternativen Ansätzen auf Basis von Differentialgleichungen deutlich überlegen. Die Anzahl an Neuberechnungen wird auf die notwendige Anzahl reduziert, während die Genauigkeit erhalten bleibt.



(a) Maschinenanordnung (b) Lagerbestandsverlauf

Abbildung 5.10: Lagerbestandsverlauf bei kontinuierlicher Produktion

5.7 Grenzzustände

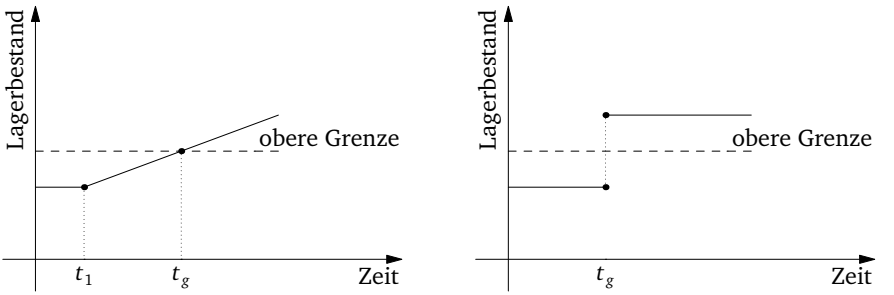
5.7.1 Definition

Bei den Berechnungen der Zustände und den Zustandsfortschreibungen in Abschnitt 5.6 wurde davon ausgegangen, daß diese auch im System eine entsprechende Repräsentation besitzen und das System sich in einem definierten Zustand befindet. Wie soll sich das System jedoch verhalten, wenn, wie in den Abbildungen auf der nächsten Seite, eine Restriktion verletzt wird und der Zustand nicht mehr definiert ist und wann soll es darauf reagieren?

Definition 3 (Anfangszustand)**.** *Das System befindet sich zu Beginn der Simulation in einem Zustand, in dem keine Restriktion verletzt wird.*

Definition 4 (Grenzzustand)**.** *Sobald das System eine durch einen Baustein gegebene Restriktion verletzt, wird die Simulation beendet, da das System sich nicht mehr in einem definierten Zustand befindet. Der Zustand, an dem die Verletzung auftritt, wird als* ***Grenzzustand*** *bezeichnet.*

Der Anfangszustand des Systems wird nur dadurch eingeschränkt, daß an jedem Baustein keine Restriktion verletzt sein darf. Grenzzustände sind hiervon nicht betroffen, so daß sie als Anfangszustand erlaubt sind.



(a) Erreichen eines Grenzzustandes bei einem kontinuierlichen Materialfluß

(b) Erreichen eines Grenzzustandes bei einem diskreten Materialfluß

Abbildung 5.11: Überschreitung einer Restriktion bei Erreichen des Grenzzustandes

Als Beispiel für einen Grenzzustand zeigt Abbildung 5.11a den Lagerbestandsverlauf in einem Baustein, der durch einen kontinuierlichen Prozeß verursacht wird. Zum Zeitpunkt t_1 startet eine vorgelagerte Produktion, wodurch sich der Lagerbestand kontinuierlich bis zum Zeitpunkt t_g erhöht. Im Zeitpunkt t_g erreicht der Lagerbestand die obere Grenze des Lagers, so daß ab hier die Simulation nicht mehr definiert ist und abgebrochen wird. Der Zustand, den das System im Zeitpunkt t_g erreicht hat, wird als Grenzzustand bezeichnet. Die Simulation kann und soll auch keine Gegenmaßnahmen ergreifen, um solche Situationen zu verhindern. Die Anwendung von Gegenmaßnahmen ist eine Aufgabe der Planung und somit nicht Bestandteil dieser Ebene. Ein weiteres Beispiel wird in Abbildung 5.11b gezeigt. Die Überschreitung tritt jedoch nicht aufgrund einer kontinuierlichen Materialflußfunktion ein, sondern wird durch einen diskreten Materialfluß ausgelöst. Die Auswirkungen sind identisch zum ersten Fall, die Simulation wird abgebrochen, da eine Restriktion verletzt wurde. Der wesentliche Unterschied ist jedoch, daß diese Überschreitung der Restriktionen nicht vorhersehbar war, sondern nur aufgrund einer diskreten Materialbewegung eingetreten ist.

Die in den Abbildungen dargestellten Ursachen müssen unterschied-

lich behandelt werden. Der geringste Aufwand ist für die Behandlung von diskreten Materialflüssen nötig. Da der Grenzzustand in dem Moment erreicht wird, in dem gleichzeitig auch eine Zustandsänderung im System durchgeführt wird, muß keine Vorausberechnung erfolgen, sondern lediglich eine Reaktion des betroffenen Bausteins. Durch gleichzeitig durchgeführte Materialbewegungen - die nach Saldierung der Materialflüsse keine Restriktion verletzen - darf auch keine Verletzung während der Bearbeitung auftreten. Im Rahmen einer Simulation ist es möglich, alle an einem Zeitpunkt stattfindenden Ereignisse zu bearbeiten, die Materialbewegungen jedoch mit einer temporären Variablen durchzuführen, um nach Abschluß aller Bewegungen den Saldo mit dem Lagerbestand zu verrechnen.

Wesentlich umfangreicher gestaltet sich die Behandlung von kontinuierlichen Materialflüssen. Abbildung 5.11a zeigt, daß im Zeitpunkt t_1 die letzte Berechnung eines Zustandes durchgeführt wurde, so daß das System zunächst in diesem Zustand verharrt. Das System berechnet den Zeitpunkt t_g, da dieser einen Grenzzustand darstellt. Findet bis dahin keine weitere Zustandsänderung im System statt, so wird die Simulation an diesem Zeitpunkt abgebrochen. Die Einführung des Grenzzustandes ist somit die logische Fortsetzung des in Abschnitt 5.6 vorgestellten Prinzips der Zustandsberechnung und -fortschreibung. Innerhalb der Simulation wird das zum Grenzzustand gehörige Ereignis nachrangig behandelt, da sichergestellt sein muß, daß parallel stattfindende Ereignisse, die den Zustand des Systems verändern, zuerst bearbeitet werden.

Definition 5 (Aktualisierung der Grenzzustände)**.** *Tritt an einem Baustein eine Zustandsänderung (vgl. Definition 2) ein, so müssen alle Komponenten des Systems, die über kontinuierliche Materialflüsse mit dem Baustein verbunden sind, die Zeitpunkte für das Erreichen von Grenzzuständen neu berechnen.*

Die Neuberechnung der Grenzzustände muß zu den gleichen Zeitpunkten stattfinden, wie in Abschnitt 5.6 bei der Fortschreibung der Zustände. Ausgenommen sind die Zeitpunkte, an denen nur statistische

Informationen abgefragt werden, die nicht mit einer Zustandsänderung verbunden sind (vgl. die Einteilung in Abschnitt 5.6). Aus technischen Gründen wird in der Simulationsbibliothek die Aktualisierung der Grenzzustände mit der Berechnung der Materialflüsse durchgeführt (vgl. Abschnitt 5.5 auf Seite 128 und Abschnitt 5.6 auf Seite 133).

5.7.2 Berechnung von Grenzzuständen

Die Berechnung der Zeitpunkte, an denen ein Grenzzustand erreicht wird, erfordert eine Betrachtung der Ursachen und beteiligten Komponenten unter mathematischen Gesichtspunkten. Restriktionen treten nur bei Lager-Bausteinen auf, so daß sich die folgenden Ausführungen ausschließlich auf diese beziehen.

Definition 6 (Netto-Materialflußfunktion)**.** *Für die Berechnung des Grenzzustandes wird eine Materialflußfunktion berechnet, die der Summe aus allen von der Restriktion direkt betroffenen Materialflußfunktionen entspricht. In den Baustein ein-/ausgehende Flüsse werden mit einem positiven/negativen Vorzeichen addiert. Die resultierende Materialflußfunktion wird als* ***Netto-Materialflußfunktion*** *bezeichnet.*

Die Berechnung der Netto-Materialflußfunktion hat vor der Berechnung der Zeitpunkte zu erfolgen. Für jede Restriktionengruppe muß eine separate Netto-Materialflußfunktion berechnet werden. Eine Restriktionengruppe ist durch die Eigenschaften des Lagers definiert. Wenn ein Lager für jedes Produkt getrennt eine obere und untere Beschränkung aufweist, so muß die Netto-Materialflußfunktion für jedes Produkt gebildet werden. Besteht hingegen eine gemeinsame Restriktion, die sich über mehrere Produkte oder Produktgruppen erstreckt, so muß hierfür eine entsprechend gewichtete Netto-Materialflußfunktion gebildet werden. Innerhalb dieses Abschnitts wird vereinfachend der Begriff der Materialflußfunktion als Ersatz für die Netto-Materialflußfunktion verwendet.

Tabelle 5.1 auf der gegenüberliegenden Seite zeigt alle Varianten, die durch die Kombination aus Restriktionen und kontinuierlichen Materialflußfunktionen auftreten können. Es gibt zwei Arten von Restriktio-

	Restriktionen	
Grenzüberschreitungen	Obergrenze	Untergrenze
0	Abb. 5.12a	Abb. 5.12a
0 (labil)	Abb. 5.12b	Abb. 5.12b
1	Abb. 5.12c	Abb. 5.12d
> 1	Abb. 5.12e	Abb. 5.12f

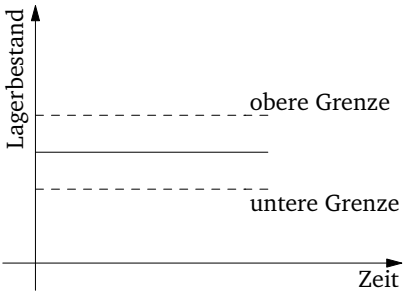
Tabelle 5.1: Systematische Einordnung der Ursachen für Grenzzustände

nen im Bereich der Lager-Bausteine: obere und untere Beschränkungen der Lagerkapazität. Drei Fälle müssen unterschieden werden: Die Materialflußfunktion verletzt nie, einmal, mehrmals eine Restriktion.

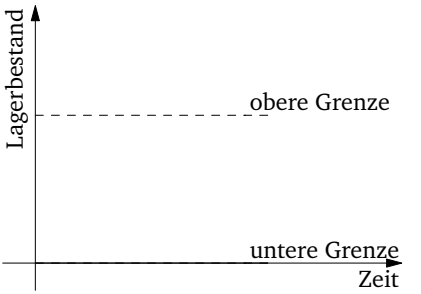
Die Abbildungen 5.12a bis 5.12f auf der nächsten Seite zeigen Beispiele für alle Kombinationen an Grenzüberschreitungen. In Abbildung 5.12a verletzt die Materialflußfunktion keine Grenzen, so daß kein Grenzzustand eintritt und der Zeitpunkt $t_g = +\infty$ gesetzt wird. Ein damit verwandter Sonderfall ist in Abbildung 5.12b zu sehen. Die Materialflußfunktion stimmt hier exakt mit der Ober- oder Untergrenze überein. Das System befindet sich im Grenzzustand. Dieser Zustand ist zulässig, solange es sich nicht darüber hinausbewegt. Ein solches System kann analog zur Physik als labil bezeichnet werden. Da die Materialflußfunktion keine Grenzen verletzt, ist wie in Abbildung 5.12a $t_g = +\infty$ zu setzen.

Ein Verletzung von Ober- bzw. Untergrenze in einem Punkt zeigen die Abbildungen 5.12c und 5.12d. Der Zeitpunkt t_g wird entsprechend berechnet. Als Fortsetzung dieser beiden Fälle können die Grenzen von einer Funktion auch mehrfach verletzt werden, wie in Abbildung 5.12e und 5.12f dargestellt. Der Grenzzustand wird bereits bei der ersten Verletzung der Grenzen im Punkt A erreicht. Die nachfolgenden Verletzungen im Punkt B oder später sind nicht zu berücksichtigen.

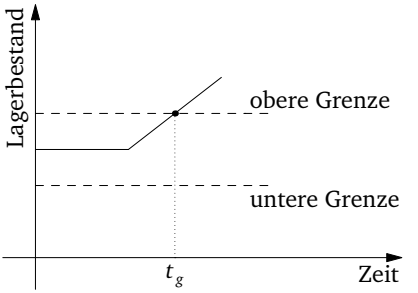
Zur Berechnung des Zeitpunktes t_g muß die Materialflußfunktion betrachtet werden. Gesucht ist der Zeitpunkt t_g an dem $f(t_g) = C$ gilt, wobei C einer der Grenzen entspricht. Da beide Grenzen unabhängig



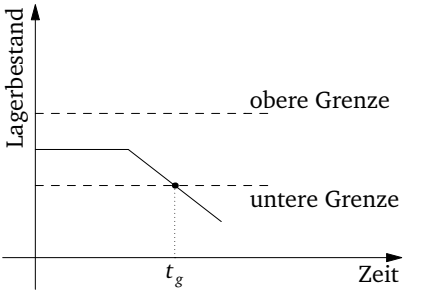
(a) Keine Verletzung von Ober- oder Untergrenzen



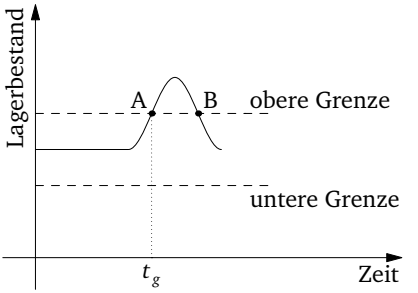
(b) Übereinstimmung von Materialfluß und Kapazitätsgrenzen



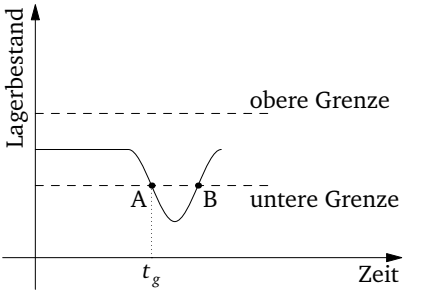
(c) Verletzung der Obergrenze an einem Punkt



(d) Verletzung der Untergrenze an einem Punkt



(e) Verletzung der Obergrenze an mehreren Punkten



(f) Verletzung der Untergrenze an mehreren Punkten

Abbildung 5.12: Ursachen für Grenzzustände

voneinander sind, wird für jede Restriktion ein Zeitpunkt t_g^o und t_g^u ermittelt, um anschließend das Minimum aus beiden als $t_g = \min(t_g^o, t_g^u)$ zu bestimmen. Für jede Grenze ergibt sich durch Umformung $f(t_g) - C = 0$, die Suche nach den Nullstellen einer Funktion. Da eine Funktion keine, eine oder mehrere Nullstellen besitzen kann, gilt folgende Definition:

Definition 7 (Berechnung von t_g)**.** *Ein Grenzzustand ist an der kleinsten Stelle $t_g \in \mathbb{R}_0^+$ erreicht, an dem die Netto-Materialflußfunktion den Wert $f(t_g) = C$ (Grenzwert) annimmt und die keinen Extremwert (Minimum oder Maximum) darstellt. Findet sich kein t, daß diese Bedingungen erfüllt, so wird der Grenzzustand bei $t_g = +\infty$ erreicht.*

5.8 Implementierung

Aufbauend auf den Modifikationen an DESMO-J (vgl. Abschnitt 4.3) wurde eine Objektbibliothek in Java erstellt. Die Bibliothek setzt auf dem in Abschnitt 4.4 vorgestellten Minimalsystem auf. Die folgenden Abschnitte stellen die Ideen und Designaspekte vor, die zum Verständnis der Implementierung notwendig sind.

5.8.1 Rezepte

Ein Rezept im Bereich der chemischen Industrie ist eine Produktionsvorschrift, die Angaben über einen oder mehrere Prozeßschritte enthält, sowie die daran beteiligten Input- und Outputfaktoren (Loos 1997, S. 174). Loos unterscheidet zwischen Ur-, Grund- und Steuerrezepten.

Ein Urrezept besteht aus verschiedenen Teilurrezepten, welche einen abgeschlossenen Abschnitt der chemischen/physikalischen Umwandlung darstellen. Dieser besteht aus einer definierten Menge an In- und Outputfaktoren sowie einer (möglichst geringen) Anzahl an (chemischen/physikalischen) Grundoperationen. Ein Teilurrezept kann zur Durchführung mehreren Anlagen belegen, für eine hohe Flexibilität in der Planung wird es meist auf eine Anlage beschränkt. Vergleichbar mit einem Urrezept ist ein Grundrezept, welches einen vergleichbaren Aufbau besitzt, jedoch mehr auf die an dem Rezept beteiligten Anlagen

fokussiert ist. Auch hier kann das Grundrezept in Teilgrundrezepte unterteilt werden. Ein Teilgrundrezept ist im Gegensatz zum Teilurrezept an eine einzige Anlage gebunden. Somit kann ein Teilurrezept in mehrere Teilgrundrezepte überführt werden. Einen Schritt weiter geht das Steuerrezept, da es bereits die konkreten Anweisungen für eine bestimmte Anlage enthält, auf der ein Los (eine Charge) hergestellt werden soll. Ein Steuerrezept kann somit nach der Zuordnung der Anlagen und der Berücksichtigung der Anlageneigenschaften aus dem Grundrezept hergeleitet werden. Eine detaillierte Darstellung der mit den Rezepten verbundenen Datenstrukturen findet sich in Loos (1997, S. 186).

Die wesentlichen Elemente eines Rezeptes innerhalb der Simulation sind:

- eine systemweit eindeutige ID
- eine fest zugeordnete Ressource, für die dieses Rezept gültig ist
- eine Menge an eingehenden Rezeptbestandteilen (Inputfaktoren)
- eine Menge an ausgehenden Rezeptbestandteilen (Outputfaktoren)
- ein Referenzbestandteil aus der Menge der ausgehenden Bestandteile, der das Verhalten bei den Mengenberechnungen des Rezeptes definiert.

Die Richtung der Rezeptbestandteile ist definiert im Bezug auf die Ressource, für die das Rezept gültig ist. Jeder Rezeptbestandteil entspricht einem kontinuierlichen oder diskreten Materialfluß, der auf einer Kante stattfindet. Jede Kante kann mehrere Materialflüsse beinhalten, wenn diese den gleichen Baustein als Quelle oder als Ziel besitzen.

Die *Recipe* Schnittstelle stellt ein Teilsteuerrezept dar, da es für eine einzige Anlage definiert ist und eine Menge an Input- und Outputfaktoren besitzt. Im Gegensatz zu einem vollständigen Steuerrezept beinhaltet sie jedoch keine Anweisungen, wie lange die Fertigung auf dieser Anlage zu laufen hat oder welche Menge gefertigt werden soll.

Diese Information muß von einer übergeordneten Ebene kommen, da diese Ebene keine Planungsaufgaben übernimmt. Ein Rezept besitzt einen einzigen internen Zustand, so daß es keiner zeitlichen Veränderung unterworfen ist. Es ist vollkommen unabhängig von äußeren Einflüssen. Wenn z. B. ein Rezept nach einem anderen nicht ausgeführt werden kann, so ist dies keine Entscheidung, die auf dieser Ebene getroffen wird. Die darüberliegende Planungsebene entscheidet dies aufgrund der Eigenschaften der Maschinen.

Flußfunktion - FlowFunction

Zur mathematischen Repräsentation von kontinuierlichen Produktionsprozessen wurde die Materialflußfunktion eingeführt (vgl. Abschnitt 5.4.2). Die Standard-Losgrößenmodelle beschränken diese auf eine lineare Funktion, welche für zukünftige Anwendungen zu restriktiv ist. Daher wird als Erweiterung die Materialflußfunktion als ein Polynom n-ten Grades implementiert. Der Definitionsbereich der Materialflußfunktion und des Polynoms stimmen überein, so daß abschnittsweise definierte Funktionen nicht unterstützt werden. Die Beschränkung auf Funktionen, die auf einem einzigen Intervall definiert sind, erfolgt allein aufgrund der Komplexität der Berechnungen. Diese steigen mit jedem zusätzlichen Intervall, so daß diese Option vorerst nicht implementiert wurde.

Durch den Einsatz von Polynomen n-ten Grades ist es möglich, daß nicht polynomial darstellbare Materialflüsse durch eine Taylor-Reihe (Bronstein u. a. 1999, Kap. 6.1.4.5) innerhalb des zu betrachtenden Intervalls mit der geforderten Genauigkeit approximiert werden können.

Die Implementierung der Polynome erfolgt über eine Standardbibliothek. Eingesetzt wird hier die Bibliothek JScience (Dautelle 2007), da sie speziell für Polynome bereits sehr viele Funktionen bereitstellt. Abbildung 5.13 auf der folgenden Seite stellt die Schnittstellendefinition der Klasse vor. Diese enthält bereits Elemente wie Variablen und Terme, die von der Repräsentation der Polynome in JScience abhängig sind. Da diese Repräsentation jedoch generisch ist und sehr gut wiederverwendet werden kann, stellt dies keine Einschränkung dar.

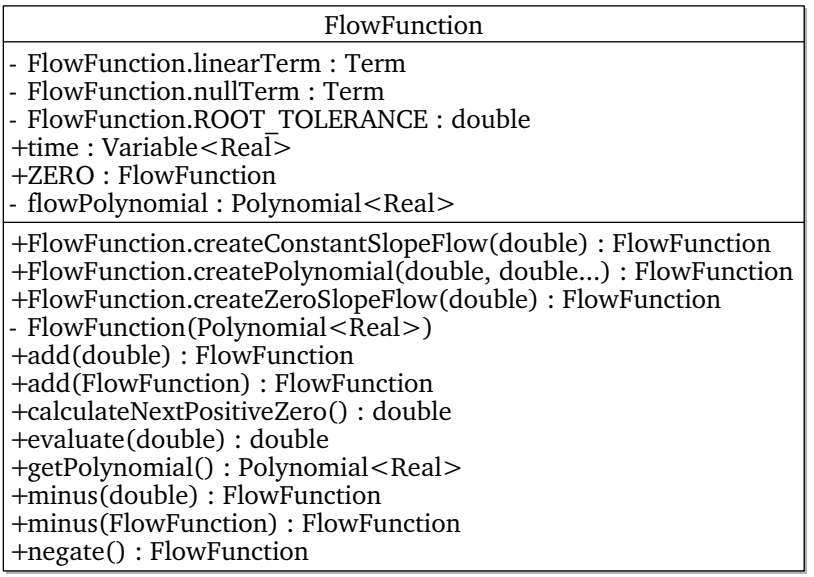


Abbildung 5.13: UML-Klassendiagramm *FlowFunction*

Die Klasse wurde so implementiert, daß sie einem Value-Objekt entspricht (vgl. Abschnitt 4.3 auf Seite 84). Alle mathematischen Operationen, die für diese Klasse zulässig sind, geben eine neue Instanz von *FlowFunction* zurück. Eingesetzt werden sie für die Berechnung der Netto-Materialflußfunktion von *StreamClearing*.

Eine wichtige Funktion ist die Bestimmung der kleinsten positiven Nullstellen zur Berechnung des nächsten Grenzzustandes (vgl. Abschnitt 5.7.2). Da es zur Berechnung der Nullstellen von Polynomen n-ten Grades mit $n \geq 5$ keine allgemeine analytische Lösung gibt (Bronstein u. a. 1999, Kap. 1.6.2.5), muß auf ein numerisches Verfahren zurückgegriffen werden. Geeignete Algorithmen sind das Graeffe-Verfahren, die Methode von Bernoulli oder die Methode von Bairstow (Bronstein u. a. 2003, Kap. 7.4.3). Alternativ zu diesen Verfahren kann auch die Eigenwertmethode (Bronstein u. a. 2003, Kap. 7.4.3.3) eingesetzt werden. Die Berechnung der Nullstellen des Polynoms wird in ein Eigenwertpro-

blem übertragen, indem das Polynom als charakteristisches Polynom einer entsprechenden Matrix angesehen wird. Für die Berechnung der Eigenwerte einer Matrix gibt es numerische Algorithmen, die in Standardbibliotheken implementiert sind. Im Gegensatz zu den anderen Verfahren, muß das Problem nur in eine andere Repräsentation überführt werden. Zur Berechnung der Eigenwerte wird die Colt-Library des CERN (CERN 2004) eingesetzt, da diese Bibliothek ausgereift ist und gut erprobte numerischen Algorithmen verwendet.

Bei der Berechnung der Nullstellen eines Polynoms n-ten Grades ist zu beachten, daß jedes Polynom n-ten Grades n Nullstellen besitzt (Bronstein u. a. 1999, Kap. 1.6.3). Da die Koeffizienten der Flußfunktion aus dem Bereich der reellen Zahlen $\mathbb{R}$ stammen, können die Wurzeln sowohl reelle ,als auch komplexe Werte annehmen. Definition 7 auf Seite 143 schränkt die gültigen Werte auf die positiven reellen Wurzeln ein. Die Eigenwertmethode gibt reelle und komplexe Wurzeln zurück, so daß die komplexen und negativen reellen Wurzeln vor der weiteren Verarbeitung aussortiert werden.

Als letztes muß bei jeder positiven reellen Wurzel überprüft werden, ob es sich um einen Extrempunkt handelt, der nicht zu beachten ist. Die Entscheidung wird anhand folgenden Satzes getroffen (Bronstein u. a. 1999, S. 385):

„Ist die Ordnung der Ableitung, die an der Stelle $t = t_g$ erstmalig nicht verschwindet, gerade, dann besitzt $f(t)$ dort ein relatives Extremum: für einen negativen Wert ein relatives Maximum, für einen Positiven ein relatives Minimum. Ist die Ordnung dieser Ableitung ungerade, dann besitzt die Funktion an dieser Stelle keinen Extremwert, sondern einen Wendepunkt."

Voraussetzung für die Anwendung dieses Satzes ist die Differenzierbarkeit der Funktion $f(t)$. Da die Flußfunktionen durch Polynome dargestellt werden, ist die Differenzierbarkeit immer gegeben (Bronstein u. a. 1999, S. 374). Alle Anforderungen von Definition 7 konnten somit durch den Einsatz von Standardbibliotheken umgesetzt werden.

RecipeItem
- coefficient : double - direction : FlowDirection - function : FlowFunction - hash : int - product : ProductID - type : FlowType
+RecipeItem.createContinuousInput(ProductID, FlowFunction) : RecipeItem +RecipeItem.createContinuousOutput(ProductID, FlowFunction) : RecipeItem +RecipeItem.createDiscreteInput(ProductID, double) : RecipeItem +RecipeItem.createDiscreteOutput(ProductID, double) : RecipeItem - RecipeItem(FlowDirection, ProductID, coefficient, FlowType, FlowFunction) +equals(Object) : boolean +getCoefficient() : double +getDirection() : FlowDirection +getFunction() : FlowFunction +getProductID() : ProductID +getType() : FlowType +isContinuous() : boolean +isDiscrete() : boolean +isInput() : boolean +isOutput() : boolean

Abbildung 5.14: UML-Klassendiagramm *RecipeItem*

Rezeptbestandteile - RecipeItem

Die Input- und Outputfaktoren, aus denen ein Rezept besteht, werden durch die Klasse *RecipeItem* implementiert. Das Klassendiagramm auf dieser Seite stellt die Struktur der Klasse dar. Das Design entspricht einem Value-Objekt (vgl. Abschnitt 4.3 auf Seite 84). Dadurch ist es möglich, daß die Instanzen von unterschiedlichen Rezepten gemeinsam verwendet werden.

Die Attribute eines *RecipeItem* setzen sich zusammen aus dem Produkt, der Flußrichtung bezogen auf den aktiven Baustein, dem Flußtyp (diskret oder kontinuierlich), sowie der Flußfunktion und dem Produktionskoeffizienten. Die beiden letzten Attribute werden je nach Flußtyp belegt und können auch den Wert *null* enthalten. Es werden entsprechende Methoden bereitgestellt, um Flußtyp und Flußrichtung

abzufragen. Flußtyp und Flußrichtung sind in Java als öffentlich zugänglich Klasse vom Typ *Enum* definiert. Der Konstruktor ist nach außen nicht sichtbar und kann nur von den statischen Factory-Methoden verwendet werden. Diese decken alle Kombinationen aus Flußtyp und Flußrichtung ab und konfigurieren den Konstruktor für den Benutzer transparent.

Aufbau eines Rezeptes

Basierend auf den Anforderungen wurde die Schnittstelle *Recipe* geschaffen, die Methoden bereitstellt, um die benötigten Informationen zu erhalten. Als abstrakte Oberklasse implementiert *AbstractRecipe* alle Methoden der Schnittstelle. Die Methoden zur Berechnung von Materialflüssen in einem Intervall oder bis zu einem bestimmten Punkt werden zurückgeführt auf die Berechnung der Flußfunktion zu einem Zeitpunkt t. Durch eine Lineartransformation werden so ausgehend vom Startzeitpunkt die Mengen berechnet. Der Startzeitpunkt ist dem Rezept nur als Parameter bekannt und wird von der Kante übergeben, auf der ein Fluß stattfindet.

Die Klasse implementiert nicht die Methoden, die zur Berechnung der diskreten Bestandteile verantwortlich sind, da diese von der Art des Hauptbestandteils abhängig sind. Das Verhalten der kontinuierlichen Bestandteile ist hingegen identisch und erfolgt in *AbstractRecipe*. Die Implementierung wird von zwei abgeleiteten Klassen durchgeführt, die für kontinuierliche und diskrete Hauptbestandteile das entsprechend angepaßte Verhalten zeigen. Aus dem Klassendiagramm auf der nächsten Seite ist ersichtlich, daß die Klasse zu einem Value-Objekt entwickelt wurde (vgl. Abschnitt 4.3 auf Seite 84). Dieses Design wird auch für die davon abgeleiteten Klassen eingesetzt. Der Vorteil eines Value-Objektes zeigt sich hier in dem bei der Erzeugung definierten Zustand. Das Rezept ist von der Simulationszeit unabhängig und kann von verschiedenen Bausteinen gleichzeitig verwendet werden, sowie während der gesamten Simulationsdauer erhalten bleiben.

Die Erzeugung eines Rezeptes wird über einen Builder unterstützt, der eine öffentliche statische lokale Klasse von *AbstractRecipe* ist (siehe

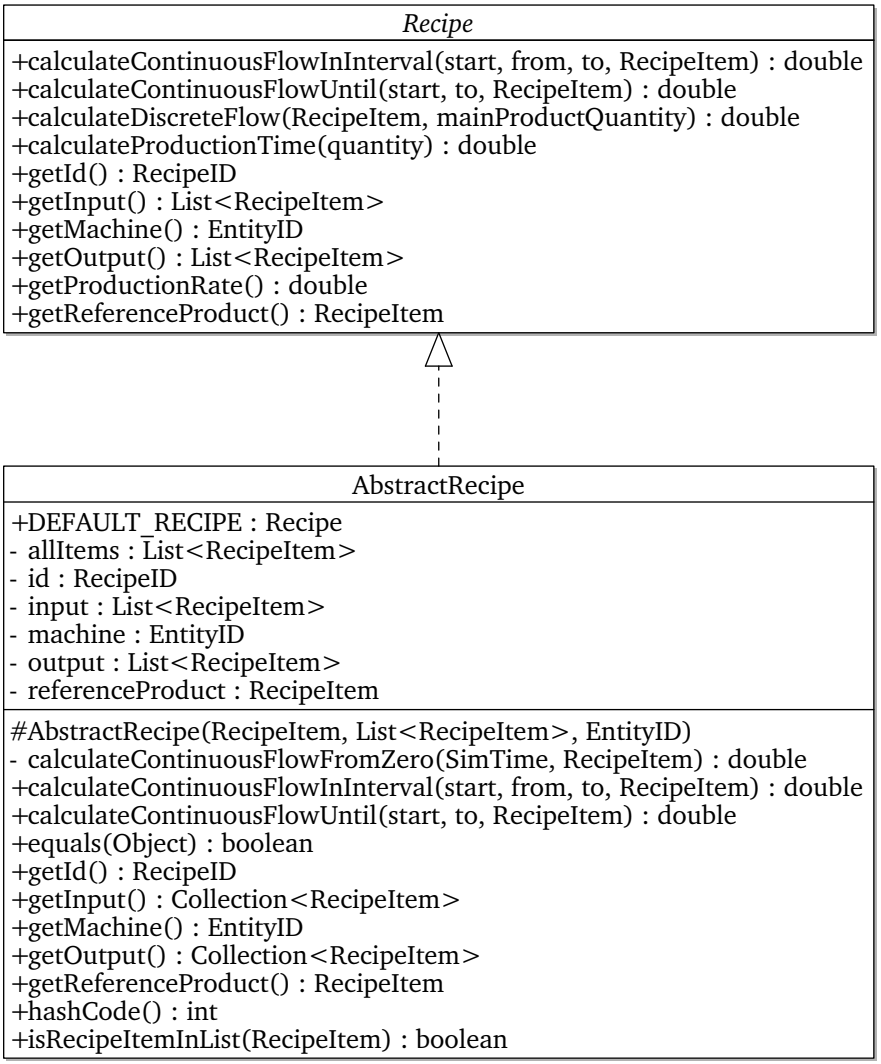


Abbildung 5.15: UML-Klassendiagramm *Recipe*

auch Gamma u. a. 1995, S. 97 und Bloch 2008, S. 11). Der Builder übernimmt bei der Erstellung als zwingenden Parameter den Referenzbestandteil, womit der Charakter des Rezeptes festgelegt ist. Die weiteren Eigenschaften eines Rezeptes werden aus den Bestandteilen *RecipeItem* zusammengesetzt. Diese können diskrete oder kontinuierliche Materialflüsse darstellen, die ggf. eine *FlowFunction* beinhalten. Da jeder Bestandteil eines Rezeptes entweder ein elementarer Datentyp ist oder sich auf einen solchen zurückführen läßt und gleichzeitig jeder Bestandteil ein Value-Objekt ist, entspricht die Klasse *AbstractRecipe* einem zusammengesetzten Value-Objekt (Evans 2004, S. 98). Die Vorteile dieses Designs ergeben sich aus der Rückführung auf andere elementare Objekte, die für sich alleine nur eine geringe Komplexität besitzen, gemeinsam jedoch eine große Flexibilität aufweisen. Die Klasse kann so alle Vorteile eines Value-Objektes nutzen, wie z. B. den Einsatz in einer Multi-Thread Umgebung. Eine Erweiterung der Testumgebung durch parallele Läufe oder eine Parallelisierung einzelner Komponenten ist für die Datenstrukturen unproblematisch, da sie inhärent „thread safe“ sind (Bloch 2008, S. 75).

Die Erzeugung einer *Recipe* Instanz wird nach der Konfiguration durch *build* ausgelöst. Die Methode überprüft den Flußtyp des Referenzbestandteils und erzeugt eine Instanz von *ContinuousRecipe* oder *DiscreteRecipe*. Beide Klassen sind nach außen nicht sichtbar, so daß für den Benutzer die Erzeugung der korrekten Instanziierung vollständig transparent verläuft. Das Verhalten der Klassen wird in den folgenden Abschnitten beschrieben.

Verhalten eines Rezeptes

Das Verhalten eines Rezeptes wird durch den bei der Initialisierung definierten Rezeptbestandteil festgelegt. Es gibt zwei Arten von Rezeptbestandteilen: kontinuierliche und diskrete Bestandteile. Davon abgeleitet entsprechend zwei Rezeptarten, die über den Hauptbestandteil charakterisiert sind. Abbildung 5.16 auf der nächsten Seite zeigt beide Klassen, die von *AbstractRecipe* abgeleitet wurden.

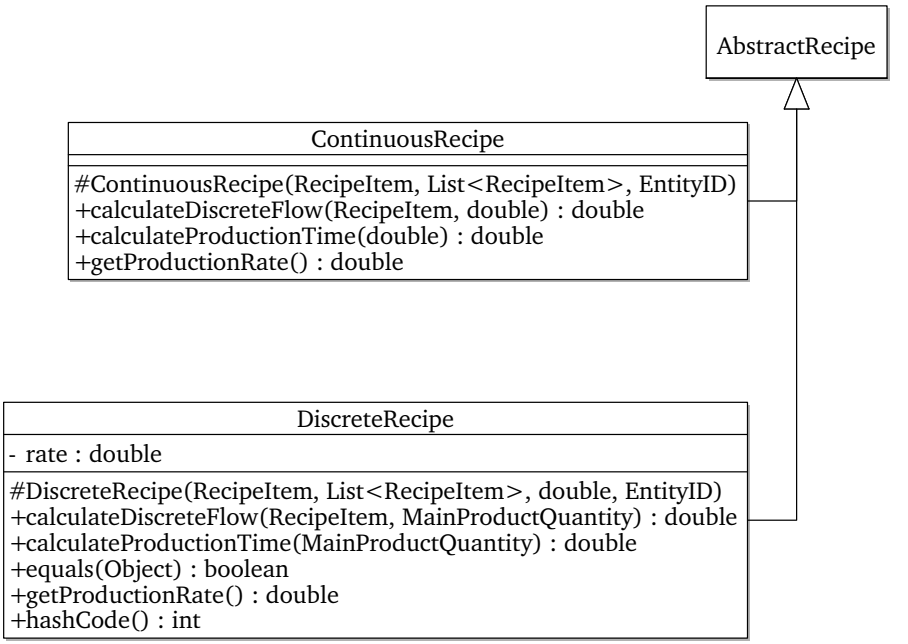


Abbildung 5.16: UML-Klassendiagramm der Nachfolger von *AbstractRecipe*

Kontinuierliche Rezepte

Ein kontinuierliches Rezept erzeugt als Hauptprodukt einen kontinuierlichen Materialfluß. Die Dauer der Produktion wird von einer übergeordneten Planungsebene definiert. Damit ist auch die Produktionsdauer aller anderen kontinuierlichen Input- und Outputfaktoren bestimmt. Die Materialflüsse sind für diese Faktoren während der gesamten Produktionsdauer aktiv. Die Menge der erzeugten und verbrauchten Faktoren wird somit bei allen kontinuierlichen Materialflüssen allein durch die Produktionsdauer bestimmt.

Zu Beginn der Produktion wird mit Hilfe der Flußfunktion des Hauptproduktes die Produktionsdauer berechnet. Voraussetzung ist die Losgröße, die von der Planungsebene bereitgestellt wird. Diskrete Bestandteile werden hingegen unabhängig von der Produktionszeit nur einmal in der angegebenen Menge eingesetzt. Als Inputfaktor zum Beginn der Produktion und als Outputfaktor, wenn die Ressource die Produktion beendet. Ein Aufruf von *calculateDiscreteFlow* wird somit immer das Attribut *coefficient* des *RecipeItem* zurückgeben. Bei der Berechnung der Produktionsdauer für eine vorgegebene Menge greift *calculateProductionTime* auf die Flußfunktion des Referenzbestandteils zurück und läßt diese den Zeitpunkt berechnen, vergleichbar mit der Berechnung eines Grenzzustandes (vgl. Abschnitt 5.7.2).

Diskrete Rezepte

Ein diskretes Rezept erzeugt als Hauptprodukt am Ende der Produktion eine Menge, die abhängig von der Produktionsdauer des Rezeptes ist. Die Produktionsmenge muß nicht ganzzahlig sein, sondern kann auch beliebige Bruchteile annehmen. Kontinuierliche Bestandteile werden während der gesamten Produktionsdauer des Rezeptes erzeugt und sind durch ihre Materialflußfunktion charakterisiert. Inputfaktoren von diskreten Bestandteilen werden zum Beginn der Produktion reserviert, Outputfaktoren am Ende der Produktion bereitgestellt. Die Menge wird in *calculateDiscreteFlow* wie bei einer klassischen Stückliste durch die Mengenverhältnisse und Produktionskoeffizienten bestimmt. Da die

Mengen zum Beginn der Produktion reserviert werden, muß die Produktionsmenge des Hauptproduktes vorher bekannt sein. Die Information hierüber wird von der Planungsebene als Losgröße bereitgestellt. Aus der Losgröße des Hauptproduktes wird mit Hilfe der Produktionsrate einer Maschine in *calculateProductionTime* die Produktionsdauer errechnet, die als Ausgangspunkt für die weiteren Berechnungen der Mengen an Input- und Outputfaktoren dient.

5.8.2 Allgemeine Modellkomponenten

ProductQuantity

ProductQuantity ist eine zentrale Klasse innerhalb dieser Ebene. Sie verbindet Informationen über ein Produkt und eine Menge miteinander und bietet Methoden an, um eine neue Instanz mit veränderter Menge zu erzeugen (vgl. Abbildung 5.17 auf der gegenüberliegenden Seite). Sie stellt wie *SimTime* ein Value-Objekt dar, welches nach der Erzeugung unveränderlich ist (vgl. Abschnitt 4.3 auf Seite 84). Durch die Implementierung als Value-Objekt werden auch gleichzeitig potentielle Fehlerquellen vermieden. Da die Attributwerte unveränderlich sind, kann die Menge nur ausgelesen und nicht erhöht oder verringert werden. Dies ist erst durch Erzeugung einer neuen Instanz möglich, wozu jedoch eine ausdrückliche Zuweisung nötig ist. Eine Überprüfung der Attribute findet bei der Instanziierung statt. Negative Mengenangaben sind hier nicht möglich, gleiches gilt auch für die Rechenoperationen. Unterschreitet die Menge bei einer dieser Operationen den Wert 0, so wird ein Fehler ausgelöst und die Simulation ggf. beendet.

Eingesetzt wird sie in allen Klassen, die an der Verwaltung oder Durchführung von Materialflüssen beteiligt sind. Ihre Funktion als elementare Informationsstruktur ist vergleichbar mit der von *SimTime* auf Ebene 0.

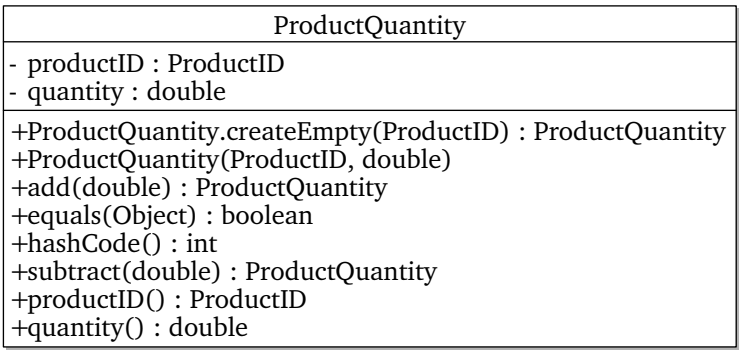


Abbildung 5.17: UML-Klassendiagramm *ProductQuantity*

AlarmClock und AlarmClockManager

Der regelmäßige Aufruf einer Methode nach Ablauf einer vorher eingestellten Zeitspanne ist eine wiederkehrende Aufgabe, die von mehreren Klassen benötigt wird. Um eine Codeduplizierung zu vermeiden, wurde die abstrakte Klasse *AlarmClock* sowie *AlarmClockManager* entwickelt (vgl. Abbildung 5.18 auf der folgenden Seite). Der Abstand zwischen zwei Aufrufen wird bei der Erzeugung festgelegt und kann im Programmablauf nicht mehr geändert werden. Intern wird ein Ereignis vom Typ *AlarmClockEvent* erzeugt, welches ein neues Ereignis erzeugt und dieses auf den nächsten Zeitpunkt terminiert. Aufgerufen wird die Methode *handleEvent*, welche von den Nachfolgern zur Definition eines eigenen Verhaltens überschrieben werden muß.

Methoden zum Abbruch einer *AlarmClock* sind durch die übergeordnete *Entity* Klasse gegeben. Außer für den regelmäßigen Einsatz, kann eine *AlarmClock* auch für einen einmaligen Aufruf verwendet werden.

Werden mehrere *AlarmClock* Instanzen parallel verwendet, so können diese über einen eindeutigen Schlüssel von einem *AlarmClockManager* verwaltet werden. Dieser bietet die Möglichkeit, für einen Schlüssel *K* ein *AlarmClockEvent* abzubrechen oder eine neue *AlarmClock* zu übernehmen. Eingesetzt wird der *AlarmClockManager* z. B. in

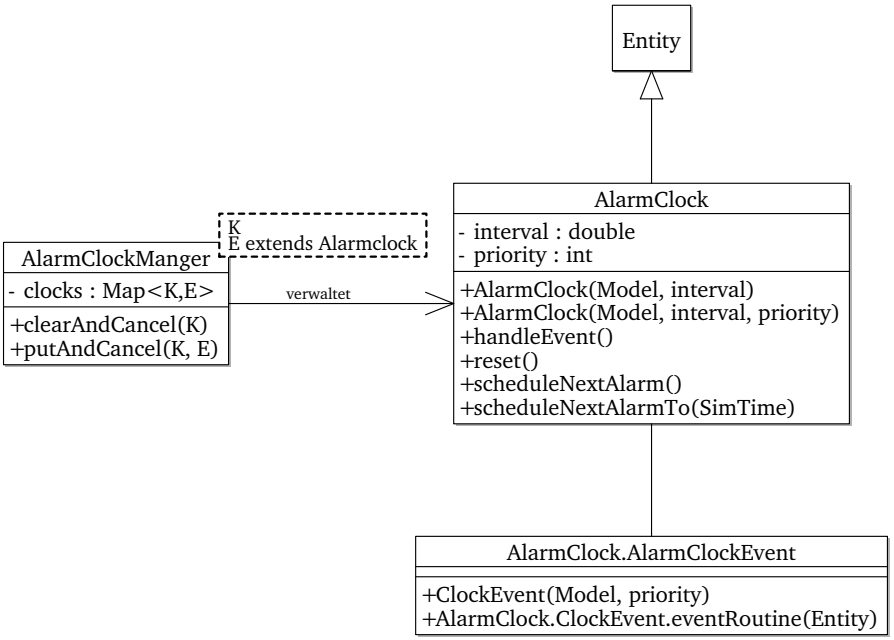


Abbildung 5.18: Abhängigkeiten von *AlarmClock*

einem *AbstractWarehouseNode*, um die eingehenden Materialflüsse zu überwachen. Als Schlüssel *K* dienen hier die *ProductID* Objekte.

AbstractOrder und InternalOrder

Aufbauend auf *ProductQuantity* werden zur Weiterleitung von Bestellungen Nachkommen der Klasse *AbstractOrder* eingesetzt. Nachfrageereignisse vom Typ *Demand* verwenden diese Klasse als Inhalt und transportieren so die Informationen über das Produkt und die Menge (*ProductQuantity*), die Lieferzeit und ob Nachlieferungen bzw. Teillieferungen akzeptiert werden. Durch einen Aufruf der Methode *handleRequest* erhalten die Materialflußbausteine die Informationen und leiten sie z. B. an *DelayedOrderHandler* weiter. Details hierzu werden in Abschnitt 5.8.4 auf Seite 161 beschrieben.

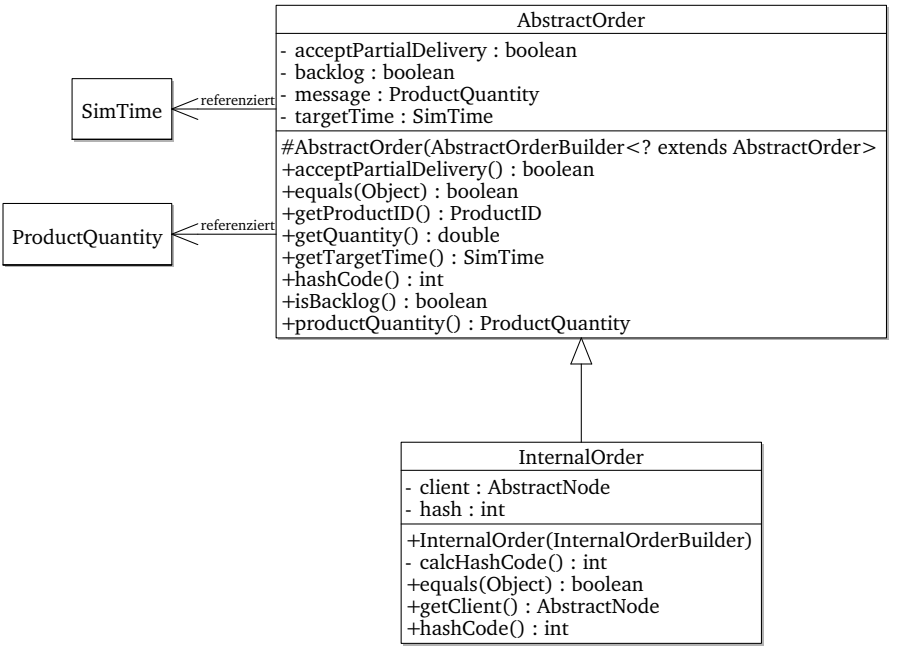


Abbildung 5.19: UML-Klassendiagramm *AbstractOrder*

Die Implementierung der Klasse *AbstractOrder* entspricht wie auch ihrer Bestandteile einem Value-Objekt (vgl. Abschnitt 4.3 auf Seite 84). Da sie jedoch abstrakt ist und somit Nachkommen existieren müssen, kann sie nicht alle Anforderungen von Bloch (2008, S. 73) erfüllen. Nachkommen wie *InternalOrder* hingegen erlauben keine Vererbung mehr, so daß sie ein vollwertiges Value-Objekt darstellen.

Die Erzeugung von Instanzen geschieht über einen Builder (Bloch 2008, S. 11), der ebenfalls vererbt werden kann. Nachkommen erweitern ihn um die neu hinzugekommenen Felder und profitieren so von der bereits vorhandenen Implementierung.

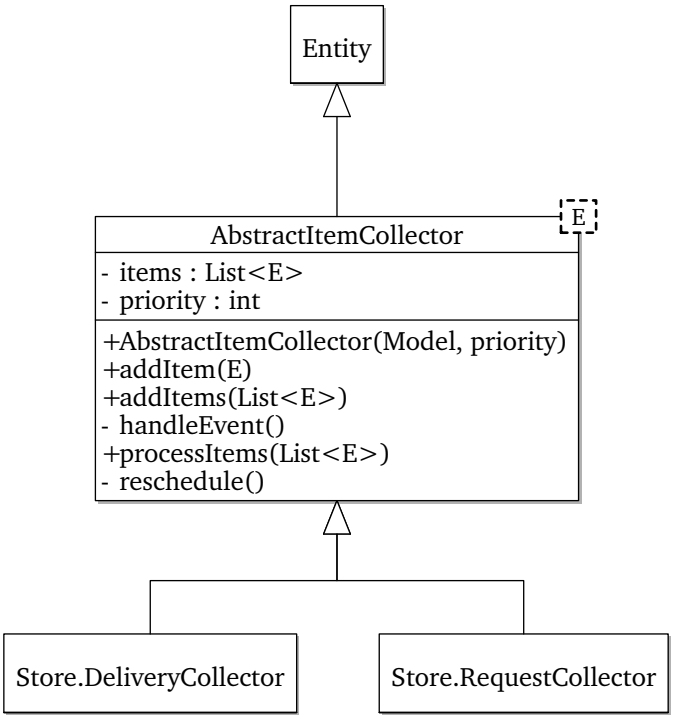


Abbildung 5.20: UML-Klassendiagramm *AbstractItemCollector*

AbstractItemCollector

Bei der Verarbeitung von Ereignissen in Materialflußbausteinen ergibt sich ein Problem, wenn mehrere Ereignisse zum gleichen Zeitpunkt eintreffen. Da während des Eintreffens keine Zeit vergeht, darf sich auch der Zustand des Bausteins nicht ändern. Die Verarbeitung der Ereignisse darf erst stattfinden, wenn sichergestellt ist, daß alle Ereignisse eingetroffen sind. Für diesen Zweck wurde die Klasse *AbstractItemCollector* entwickelt. Nachkommen dieser Klasse sammeln die Inhalte der Ereignisse und führen zum Schluß eine Methode aus, welche alle eingetroffenen Informationen bearbeitet.

Das Klassendiagramm auf dieser Seite zeigt die Struktur der Klasse.

Intern besteht sie aus einer Liste, in der Objekte vom Typ *E* gesammelt werden. Diese werden von außen über *addItem* hinzugefügt. Mit jedem Aufruf wird auch gleichzeitig ein internes Ereignis ausgelöst, daß auf den Zeitpunkt *SimTime.NOW* gesetzt wird. Das Ereignis wird mit dem Collector verbunden, so daß er ein Nachkomme von *Entity* sein muß. Durch Vorgabe einer Priorität bei der Initialisierung des Collectors kann sichergestellt werden, daß dieses Ereignis erst ausgeführt wird, wenn eine andere Gruppe von Ereignissen mit höherer Priorität vorher vollständig verarbeitet wurde. Bei Aufruf des internen Ereignisses wird *processItems* ausgeführt, welches von den Nachkommen dieser Klasse entsprechend überschrieben werden muß. Nach der Ausführung ist für den Collector kein Ereignis mehr terminiert. Die nächste Aktivierung erfolgt erst nach einem Aufruf von *addItem*. Eingesetzt wird der Collector in allen Materialflußbausteinen, die asynchron Ereignisse verarbeiten. Implementiert wird er als nicht statische innere Klasse, die nach außen nicht sichtbar ist. Somit kann sichergestellt werden, daß das Verhalten eines Bausteins nach außen unabhängig von der Sequenz der Aufrufe in einem Zeitpunkt ist.

5.8.3 Asynchroner Materialtransport über Ereignisse

Der asynchrone Transport von Material in einem Produktionsnetzwerk wird über zwei Arten von Ereignissen gesteuert - Lieferung und Nachfrage. Die Ereignisse tragen in sich Informationen, die an die Knoten weitergegeben werden. Zur Verarbeitung besitzen alle Knoten Methoden für Nachfragen und Lieferungen (vgl. Abschnitt 5.8.4 auf Seite 165).

Der Transport erfolgt asynchron, da durch den Einsatz von Ereignissen die Anforderung von der Lieferung getrennt wird. Der Auslöser des Ereignisses muß darauf warten, daß durch ein anderes Ereignis die Lieferung im Knoten eintrifft. Die Simulationszeit, die dabei vergeht, kann gleich null sein, doch die interne Verwaltung des Knotens muß die Trennung der Anfrage und Lieferung zusammenführen können.

Die Klassenstruktur, sowie die Informationen, die diese Ereignisse in sich tragen, sind im Klassendiagramm auf der nächsten Seite dargestellt. Beide Ereignisse können mehrere Informationen in Form einer Liste

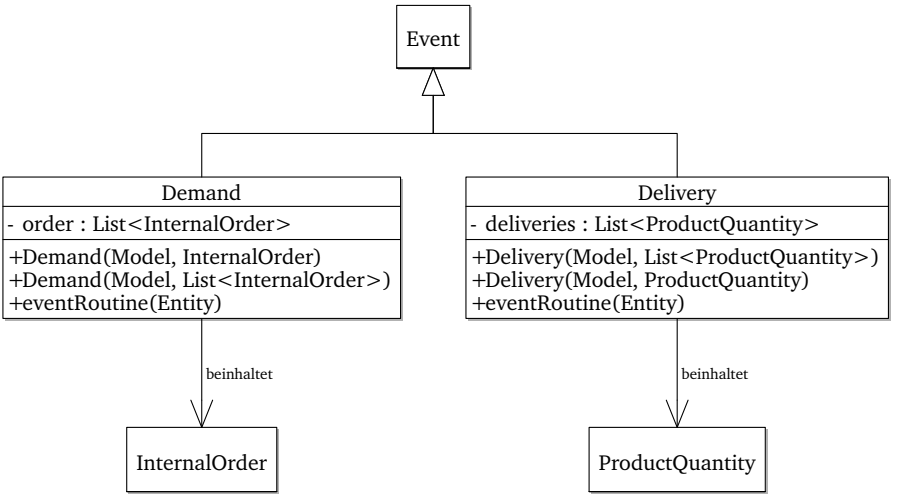


Abbildung 5.21: UML-Klassendiagramm *Delivery* und *Demand*

transportieren. Durch die Konsolidierung von gleichzeitig stattfindenden Nachfragen / Lieferungen wird die Anzahl der Ereignisse im Modell reduziert.

Die Bedeutung beider Ereignisse in einem Modell ist in den innerhalb dieser Arbeit vorgestellten Anwendungen gering, da ein asynchroner Transport nur stattfindet, wenn über externe Ereignisse Material in oder aus dem Netzwerk transportiert wird. Eingesetzt wird dies aktuell bei externen Nachfragen von Kunden, die als *Demand* Ereignis an eine Senke weitergegeben werden. Innerhalb der Senke wird hierauf mit einer Materialanfrage reagiert, die einen synchronen Materialtransport auslöst. Ein weiteres Einsatzgebiet ist die Simulation von mehrstufigen Lagerhaltungsmodellen. Der Transport durch die gekoppelten Lager kann nur über Ereignisse erfolgen, die von einem Steuerungselement erzeugt werden.

Demand

Ziel des Ereignisses ist ein anderer Knoten, bei dem *handleRequest* aufgerufen wird. Die Höhe der Nachfrage, sowie weitere Informationen werden in Form von *InternalOrder* Objekten übergeben. Nach Durchführung des Aufrufs erfolgt keine unmittelbare Rückmeldung an den Aufrufer, so daß selbst bei einer direkten Erfüllung der Nachfrage dies durch ein weiteres Ereignis geschieht. Die Aufrufe erfolgen somit asynchron.

Delivery

Ziel des Ereignisses ist ein anderer Knoten, bei dem *handleDelivery* aufgerufen wird. Die Liefermenge, sowie weitere Informationen werden durch ein *ProductQuantity* Objekt weitergeleitet. Da hier keine Informationen über den Aufrufer (Kunden) enthalten sind, erfolgt der Transport zum Knoten anonym. Die Zuweisung der Lieferung an einen Kunden wird durch den Knoten durchgeführt.

5.8.4 Strukturelemente eines Produktionsnetzwerkes

Die Modellierung der Struktur des Produktionsnetzwerkes erfolgt analog zu den in Abschnitt 5.2 vorgestellten Losgrößenmodellen sowie deren Erweiterungen. Außer dem bereits vorgestellten asynchronen Transport von Material muß jeder Knoten auch den synchronen Transport über Materialflüsse unterstützen.

Zur Modellierung der Knoten stehen vier elementare Bausteine zur Verfügung. Als Schnittstelle nach außen dienen die Bausteine Quelle und Senke, welche einen aus- und eingehenden Materialfluß zulassen. Innerhalb des Netzwerkes können zwei weitere Bausteine, Lager und Ressource eingesetzt werden. Mit Hilfe dieser vier elementaren Bausteine lassen sich alle Losgrößenmodelle aus Abschnitt 5.2 sowie deren Erweiterungen simulieren. Ausgenommen sind Modelle, in denen für das Betreiben oder Umrüsten der Anlage bewegliche Ressourcen oder spezielles Personal nötig ist (ein entsprechendes Modell wird z. B.

von Tempelmeier u. Buschkühl 2008, S. 401 vorgestellt). Durch einen weiteren Baustein kann diese Funktion bei Bedarf nachgebildet werden. Die in dieser Arbeit erstellte Simulationsumgebung soll jedoch nur eine Referenzimplementierung darstellen, so daß auf die Modellierung von diesen weiterführenden Bausteinen verzichtet wird. Eine spätere Erweiterung wird hierdurch jedoch nicht ausgeschlossen. Eine Möglichkeit, diese Fähigkeit zu modellieren, ist die Einführung einer Rüstressource, sowie eines Bausteins, der diese koordiniert. Ein ähnlicher Ansatz wird auch in kommerziellen Simulationspaketen verfolgt.

Das Produktionsnetzwerk kann in der Simulation exakt so aufgebaut werden wie das physische Netzwerk, das simuliert werden soll. Einschränkungen gibt es bei der Kombination der Knoten innerhalb des Graphen. Nicht jede Kombination aus Vorgänger und Nachfolger ist zulässig. Tabelle 5.2 auf der gegenüberliegenden Seite führt die Eigenschaften der Bausteine auf. Ein Merkmal zur Unterscheidung ist die Rolle der Knoten im Bezug auf Materialflüsse.

Definition 8 (Aktive/Passive Bausteine)**.** *Aufgrund ihres Verhaltens im Bezug auf Materialflüsse werden folgende zwei Arten von Bausteinen unterschieden:*

a. Ein aktiver Baustein löst selbständig einen Materialfluß aus.

b. Ein passiver Baustein löst einen Materialfluß aus, wenn er von einem anderen Knoten hierzu aufgefordert wird.

Ein weiteres Unterscheidungsmerkmal der Knoten ist die Flußrichtung aus Sicht der Knoten. Auch hier gibt es Einschränkungen, da nicht jeder Baustein ein- und ausgehende Materialflüsse verarbeiten kann.

Definition 9 (Kompatibilität der Bausteine)**.** *Folgende Regeln gelten für die Kombination der Bausteine zu einem Netzwerk*

*a. Eingehende und ausgehende Materialflüsse **müssen** sich abwechseln.*

*b. Zwei aktive Bausteine **dürfen nicht** miteinander verbunden werden.*

*c. Zwei passive Bausteine **können** miteinander verbunden werden.*

Materialfluß	Quelle	Senke	Ressource	Lager
aktiv/**p**assiv	p	p	a	p
ein-/**a**usgehend	a	e	e/a	e/a

Tabelle 5.2: Klassifizierung der Bausteine in Ebene 1

Vorgänger	Nachfolger			
	Quelle	Senke	Ressource	Lager
Quelle		○	●	●
Senke				
Ressource		●		●
Lager		●	●	○

●: zulässig ○: zulässig, jedoch nicht sinnvoll

Tabelle 5.3: Kombinationsmöglichkeiten der Bausteine

Unter Beachtung dieser Regeln kann die Anordnung der Bausteine innerhalb eines Produktionsnetzwerkes beliebig sein. Eine Übersicht der zulässigen Kombinationen aus Vorgänger- und Nachfolger-Knoten befindet sich in Tabelle 5.3.

Sowohl die Quelle, als auch das Lager können mit jedem anderen Baustein als Nachfolger kombiniert werden. Für Quelle und Senke ergeben sich folgende Einschränkungen: da eine Quelle ausschließlich ausgehende Materialflüsse hat, muß sie immer mindestens einen Nachfolger besitzen und darf keine Vorgänger haben. Eine Senke hat ausschließlich eingehende Verbindungen, so daß sie immer mindestens einen Vorgänger, jedoch keine Nachfolger besitzen darf. Eine Ressource muß als Vorgänger immer mindestens eine Quelle oder ein Lager besitzen und als Nachfolger mindestens eine Senke oder ein Lager. Zwei Lager dürfen aufeinander folgen, jedoch ist dies aufgrund des Verhaltens der Bausteine nicht sinnvoll. Beide Bausteine sind passiv, so daß kein aktiver Materialfluß zwischen den Bausteinen stattfindet. Denkbar

ist jedoch, daß durch weitere Bausteine, die Steuerungsaufgaben für ein Lager übernehmen, asynchrone Materialtransporte ausgelöst werden (vgl. Abschnitt 5.8.3 auf Seite 159). Auch die Kombination zwischen einer Quelle und einer Senke ist möglich, im praktischen Einsatz jedoch sehr selten anzutreffen. Da beide Komponenten passiv sind, ist kein aktiver Materialfluß möglich. Durch einen weiteren Baustein, der z. B. eine Kundennachfrage an der Senke simuliert, kann auch hier ein asynchroner Materialtransport erzeugt werden.

Die in Tabelle 5.3 auf der vorherigen Seite aufgeführte Einschränkung, daß nicht zwei Ressourcen unmittelbar hintereinander folgen dürfen, hat ihren Ursprung im Verhalten der Ressourcen. Eine Ressource wandelt eingehende Materialflüsse um und erzeugt ausgehende Materialflüsse. Der ausgehende Materialfluß der ersten Ressource wäre gleichzeitig der eingehende Materialfluß der zweiten Ressource, ohne daß ein weiterer Knoten dazwischen liegt. Dieser ist jedoch zwingend notwendig, da er zur Konsolidierung der Ströme eingesetzt wird (die Konsolidierung wird in einem passiven Baustein von einem *Stream-Clearing* Objekt durchgeführt, vgl. Abschnitt 5.8.5 auf Seite 181).

Zur Simulation von zwei miteinander verbundenen Ressourcen gibt es zwei Alternativen: Zusammenlegung zu einer Ressource oder Einfügen eines Zwischenlagers. Die erste Alternative bietet sich an, wenn die Unterscheidung zwischen den einzelnen Ressourcen für die Simulation nicht nötig ist und somit die Modellierung vereinfacht. Die zweite Alternative muß eingesetzt werden, wenn durch eine Zusammenlegung die Ergebnisse der Simulation verändern werden.

Das einzusetzende Zwischenlager muß in diesem Fall mit einer Kapazitätsobergrenze von 0 Mengeneinheiten erstellt werden, um zu verhindern, daß Material eingelagert wird. In der Praxis ist es jedoch häufig so, daß zwischen zwei unmittelbar aufeinanderfolgenden Ressourcen ein Zwischenlager mit einer geringen Kapazität eingesetzt wird, um geringe Unterschiede in der Produktionsgeschwindigkeit kurzfristig auszugleichen. Dies spricht ebenfalls für den Einsatz eines Lagers zwischen zwei Ressourcen, hier jedoch mit einer geringen, von null verschiedenen Kapazität.

ProductionEntity

ProductionEntity ist die gemeinsame Oberklasse aller sichtbaren Entitäten in einem Produktionsnetzwerk. Sie stellt Informationen über das Netzwerk zur Verfügung, die vom umgebenden Modell bereitgestellt werden. Diese Informationen werden nicht nur von den Materialflußbausteinen verwendet, sondern auch von anderen Klassen, die direkten Zugang zu den Produktionsdaten besitzen müssen (z. B. *Lot* oder *SetupOperation*). Die Klassenhierarchie ausgehend von *Entity* ist im Klassendiagramm auf Seite 167 dargestellt.

Eine weitere wichtige Eigenschaft ist die Identifikation. Jede Instanz von *ProductionEntity* besitzt eine eindeutige *EntityID*, die vom Benutzer vorgegeben werden kann. Falls keine ID angegeben wird, so wird anhand des Hashcodes und dem Klassennamen eine eindeutige ID erzeugt. Die ID wird eingesetzt, um nach der Serialisierung/Deserialisierung Zugriff auf Modellkomponenten zu erhalten. Da nach einer Deserialisierung alle Referenzen, die von außen auf ein Modell gehalten werden, weiterhin auf das serialisierte Modell zeigen, muß eine alternative Zugriffsmöglichkeit benutzt werden, die trotzdem eine eindeutige Identifikation erlaubt. Ein Nachkomme von *Model*, der auf Produktionsnetzwerke spezialisiert ist, bietet entsprechende Schnittstellen an. Deren Aufgabe ist die Registrierung und Suche von Komponenten. Jede registrierte Entität ist somit nach außen sichtbar, wenn ihr Bezeichner bekannt ist. Die Infrastruktur, die jedem Produktionsmodell zur Verfügung steht, wird in der Abbildung auf der folgenden Seite gezeigt.

Interface Node und AbstractNode

Die Schnittstelle *Node* definiert das Verhalten aller Materialflußbausteine. Eine abstrakte Referenzimplementierung stellt die Klasse *AbstractNode* bereit (vgl. Abbildung 5.24 auf Seite 169). Diese enthält bereits alle Methoden und Felder, um Informationen über Vorgänger-/Nachfolgerknoten zu ermitteln. Diese werden von abgeleiteten Klassen für die Steuerung des Materialflusses verwendet. *AbstractNode* ist wie-

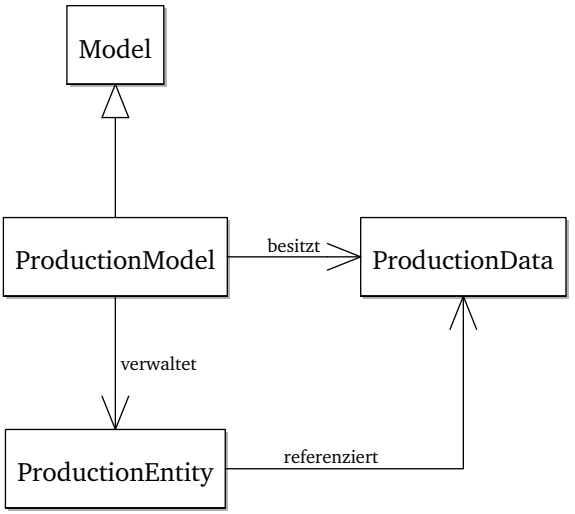


Abbildung 5.22: Infrastruktur eines Produktionsmodells

derum von *ProductionEntity* abgeleitet, da die Bausteine Zugang zu den Daten der Produktionsanlage haben müssen.

Das Klassendiagramm auf der gegenüberliegenden Seite zeigt die Hierarchie der Klassen und die Implementierung der Schnittstelle. Direkte Nachkommen von *AbstractNode* sind die Klassen für Quellen (*Source*) und Ressourcen (*Ressource*). Als dritte Klasse wird *AbstractWarehouseNode* vererbt. Von dieser leiten sich die Klassen zur Modellierung einer Senke (*Drain*) und eines Lagers (*Store*) ab.

Ein weiteres wichtiges Merkmal ist die Behandlung von Lieferungen und Anfragen. Die Schnittstelle *Node* bietet diese in jeweils zwei Ausführungen an - in Listenform und für einzelne Anfragen. Ein Baustein darf in der Bearbeitung dieser Anfragen keine Unterschiede machen. Das Ergebnis muß identisch sein, unabhängig davon, ob eine Sammelanfrage oder eine Einzelabfrage mehrfach hintereinander für unterschiedliche Produkte durchgeführt wurde. Da das Verhalten jedoch für jede Klasse individuell implementiert werden muß, erzeugt ein Aufruf der von

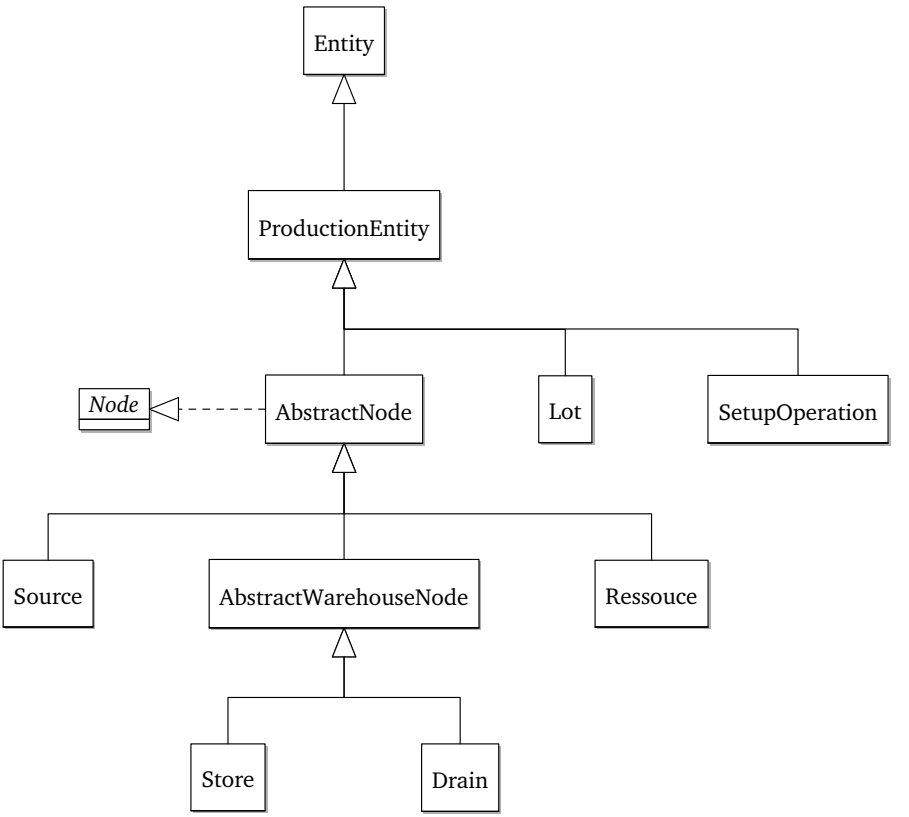


Abbildung 5.23: UML-Klassenhierarchie der Materialflußbausteine

AbstractNode geerbten Methoden immer eine *UnsupportedOperationException*.

AbstractWarehouseNode

Das Verhalten einer Senke und eines Lagers stimmt in vielen Punkten überein, so daß diese Verhaltensweisen in einem gemeinsamen übergeordneten Knotentyp (*AbstractWarehouseNode*) modelliert werden. Der wesentliche Unterschied zwischen einem Lager und einer Senke ist die Fähigkeit, Lagerbestände zu halten. Das Verhalten bei Eintreffen eines Auftrags und einer Lieferung ist nahezu identisch.

Bei Eingang einer Bestellung wird diese registriert (*registerOrder*) und auf die Liste der aktiven Bestellungen gesetzt (*delayedOrders*). Die Liste enthält alle Aufträge, die noch nicht abschließend bearbeitet wurden. Für jeden eintreffenden Auftrag überprüft die Senke, ob ein Vorgängerknoten bekannt ist, der das entsprechende Produkt liefert, und leitet den Auftrag an diesen weiter. Ein Lager hingegen aktualisiert zuerst alle Materialflüsse, damit die Lagerbestände das zu diesem Zeitpunkt gültige Niveau annehmen, und ruft anschließend die Verarbeitung der aktiven Aufträge auf.

Die an der Verarbeitung von Aufträgen beteiligten Klassen sind in Abbildung 5.26 auf Seite 171 dargestellt. Zentrales Element ist die Klasse *DelayedOrderHandler*. Diese Klasse wird bei der Initialisierung eines *AbstractWarehouseNode* entweder als Parameter vom Konstruktor übernommen oder mit dem Standardverhalten (FIFO) erzeugt. Ein *DelayedOrderHandler* besitzt eine Referenz auf eine Instanz von *AbstractPolicyFactory*, welche wiederum eine Instanz der Schnittstelle *Policy* erzeugen kann. Für jedes Produkt wird eine eigene Instanz erzeugt, der bei der Erzeugung die Menge an aktiven Aufträgen sowie der aktuelle Lagerbestand übergeben wird. Die Liste mit Aufträgen ist immer in der Reihenfolge des Auftragseingangs geordnet. Bausteine wie eine Quelle oder Senke erzeugen als Reaktion auf Bestellungen und Lieferungen eine Instanz und überprüfen, ob Aufträge vorliegen. Falls Aufträge im System vorhanden sind, werden diese solange zugewiesen und abgewickelt, bis aufgrund des Lagerbestands oder anderer Kriteri-

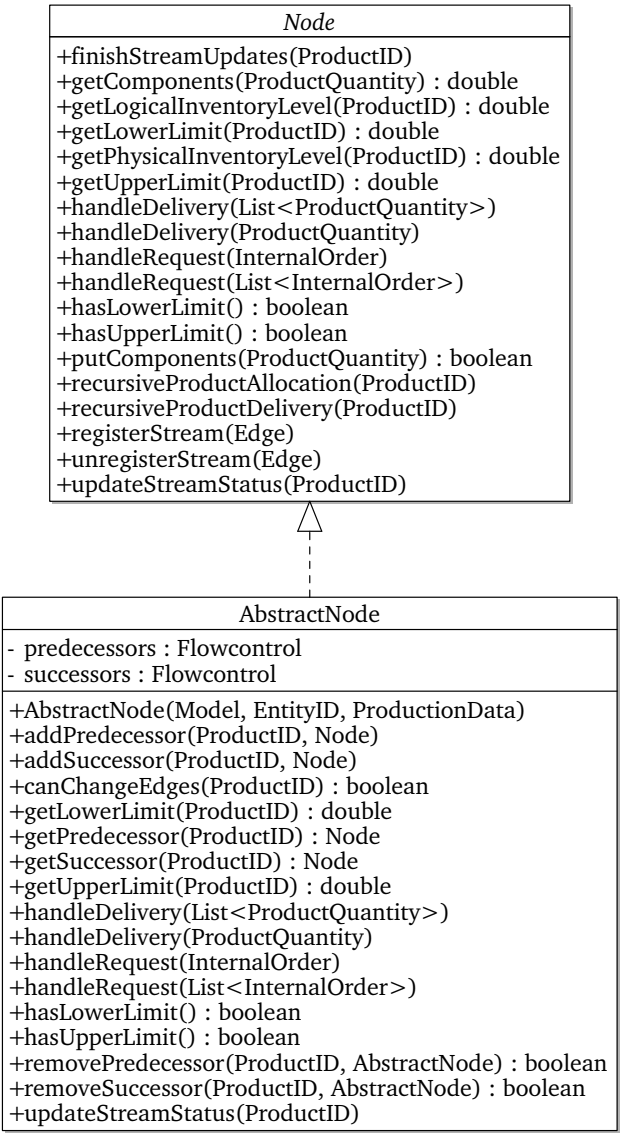


Abbildung 5.24: UML-Klassendiagramm *Node* und *AbstractNode*

AbstractWarehouseNode
- clearing : StreamClearing - customerAlerts : AlarmClockManager<ProductID, CustomerOrderAlarm> - delayedOrders : DelayedOrderHandler
+AbstractWarehouseNode(Model, EntityID, ProductionData) +AbstractWarehouseNode(Model, EntityID, ProductionData, DelayedOrderHandler) #calculateTimeToFulfillNextOrder(ProductID) : double +canChangeEdges(ProductID) : boolean +finishStreamUpdates(ProductID) #fulfillCustomerOrderWithStreamInput(ProductID) +getBacklogLevel(ProductID) : double +getBacklogLevels() : double #getPolicy(ProductID, double) : Policy +recursiveProductAllocation(ProductID) +recursiveProductDelivery(ProductID) #registerOrder(InternalOrder) +registerStream(Edge) #setCustomerAlert(ProductID) #unregisterStream(Edge)

Abbildung 5.25: Elemente von *AbstractWarehouseNode*

en kein Auftrag mehr vom System verarbeitet werden kann. Die *Policy* greift direkt auf die internen Daten von *DelayedOrderHandler* zu und verändert diese.

Die Bausteine rufen ausschließlich Methoden von *DelayedOrderHandler* und *Policy* auf, so daß die tatsächliche Implementierung verborgen bleibt. Auf diese Weise lassen sich unterschiedliche Politiken implementieren, ohne daß der Code der Bausteine geändert werden muß. Auch zusätzliche Kriterien wie z. B. Prioritätsregeln können durch eine Erweiterung der intern verarbeiteten Aufträge (*InternalOrder*) realisiert werden. Die Einführung dieser Abstraktionsschicht hat zunächst zu einer Zunahme der Klassen geführt und die Anzahl an Interaktion erhöht. Durch die verbesserten Erweiterungsmöglichkeiten wird dieser Nachteil jedoch aufgewogen.

Eine weitere wichtige Anwendung tritt zusammen mit eingehenden Materialflüssen auf. Wenn in einem Knoten Aufträge vorliegen, die später erfüllt werden können, müssen diese bei eingehenden Materialflüssen berücksichtigt werden. Zu dem Zeitpunkt, an dem genügend

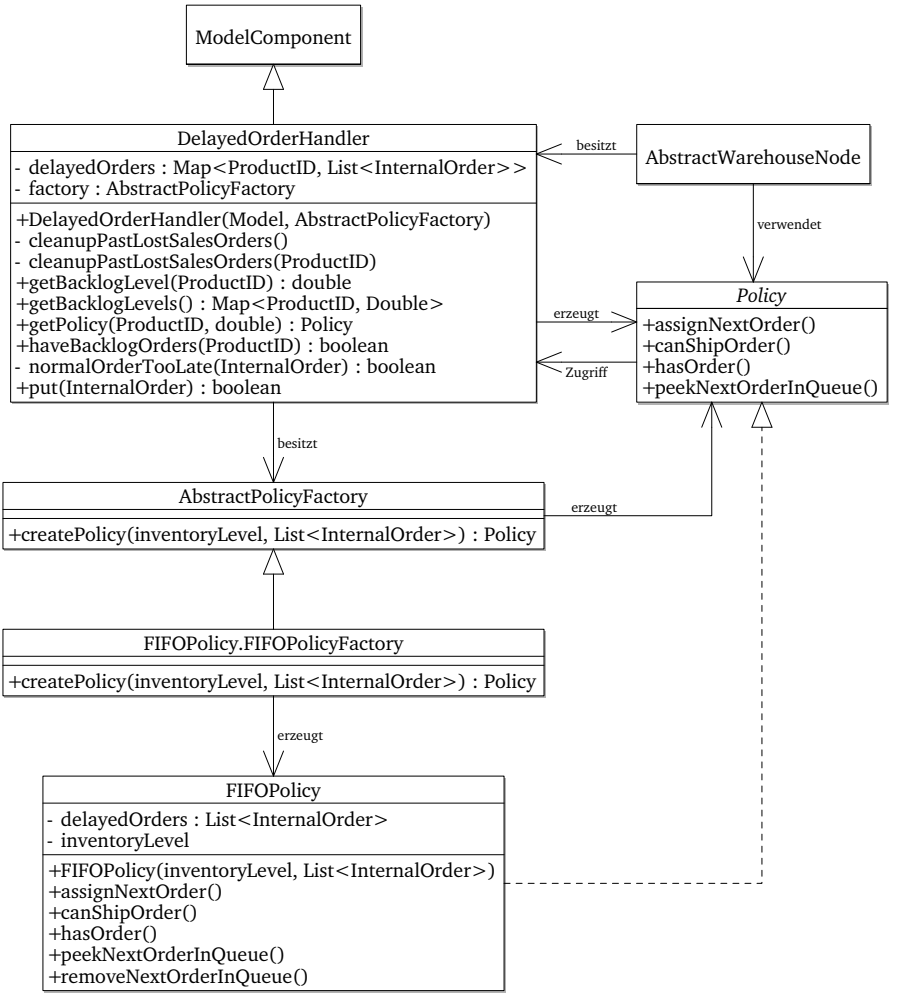


Abbildung 5.26: An der Verarbeitung von Aufträgen beteiligte Klassen

Material eingegangen ist, um den Auftrag zu erfüllen, muß eine Lieferung ausgelöst werden. Verantwortlich hierfür sind die Methoden *setCustomerAlert* und *calculateTimeToFulfillNextOrder*. Letztere greift auf die für das Produkt zuständige *Policy* zurück und ermittelt die Größe des nächsten Auftrags.

Source

Eine Quelle ist die Schnittstelle nach außen, in der ein/mehrere Materialflüsse ihren Ursprung haben. Eine Quelle besitzt keine Vorgänger und mindestens einen Nachfolger. Somit ist sie einzuordnen in den Bereich schöpfender Strukturtypen (Dyckhoff 2006, S. 93). Eine Quelle kann jedes Produkt zu jedem Zeitpunkt in unbeschränkter Anzahl zur Verfügung stellen. Die Anzahl der Quellen innerhalb einer Simulation ist nicht begrenzt.

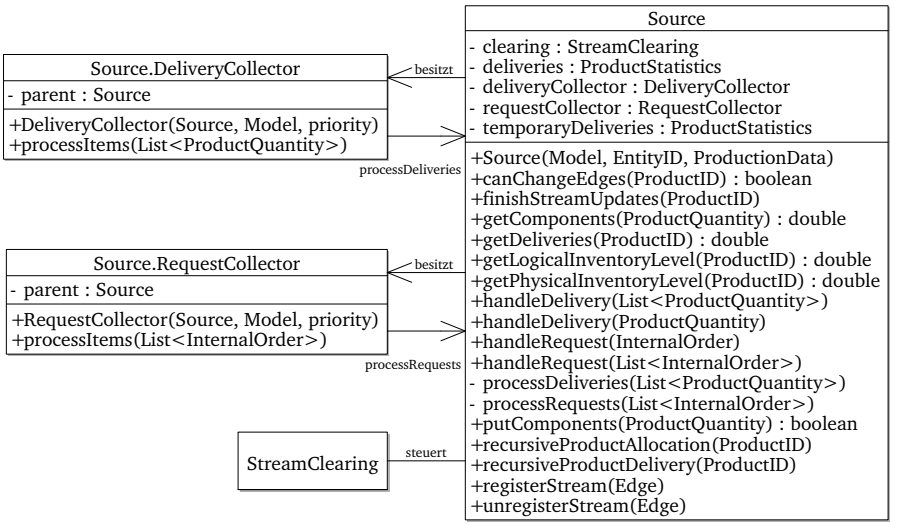


Abbildung 5.27: UML-Klassendiagramm *Source*

Die Implementierung baut direkt auf der Oberklasse *AbstractNode* auf (vgl. Abbildung 5.23 auf Seite 167). Die Anbindung an andere

Klassen zeigt die Abbildung auf der gegenüberliegenden Seite. Da eine Quelle keine Bestände führt und jedes Produkt in unbeschränkter Anzahl liefern kann, ist der Rückgabewert für alle Methoden, die sich auf die Lagerbestände beziehen, $+\infty$. Der asynchrone Materialtransport über *handleRequest* und *handleDelivery* sammelt alle Anfragen in Nachkommen der Klasse *AbstractItemCollector* und führt anschließend *processDeliveries* bzw. *processRequests* aus (vgl. Abschnitt 5.8.2 auf Seite 158). Die hier eingesetzten Klassen sind als private Klassen implementiert, so daß die Implementierung nach außen verborgen bleibt.

Obwohl eine Quelle nur ausgehende Materialflüsse besitzen darf, werden auch Lieferungen an die Quelle bearbeitet. Diese können in Ausnahmefällen auftreten, wenn z. B. nach einem Abbruch der Produktion Material zurückgegeben wird. Die Verwaltung der Materialflüsse wird von einem *StreamClearing* Objekt durchgeführt. Alle Methoden, die den Materialfluß betreffen, werden per Delegation an dieses Objekt weitergeleitet.

Drain

Eine Senke ist die Schnittstelle nach außen, durch die Material und Materialflüsse die Simulation verlassen. Sie darf keine Nachfolger besitzen, muß jedoch mindestens einen Vorgänger haben. Produkte können zu jedem Zeitpunkt in unbeschränkter Anzahl aufgenommen werden. Sie ist daher in den Bereich der vernichtenden Strukturtypen (Dyckhoff 2006, S. 93) einzuordnen. Die Zahl der Senken innerhalb einer Simulation ist nicht begrenzt.

Eine Senke wird aufgrund ihres Verhaltens von *AbstractWarehouseNode* (vgl. Abschnitt 5.8.4 auf Seite 168) abgeleitet. Die Anbindung an andere Klassen zeigt die Abbildung auf der nächsten Seite. Die Methoden des asynchronen Materialtransports sind identisch zu denen von *Source* implementiert. Die Implementierung verwendet private Klassen, die Nachkommen von *AbstractItemCollector* sind.

Die Verwaltung des Materialflusses erfolgt durch die übergeordnete Klasse *AbstractWarehouseNode*, welche auf ein *StreamClearing* Objekt

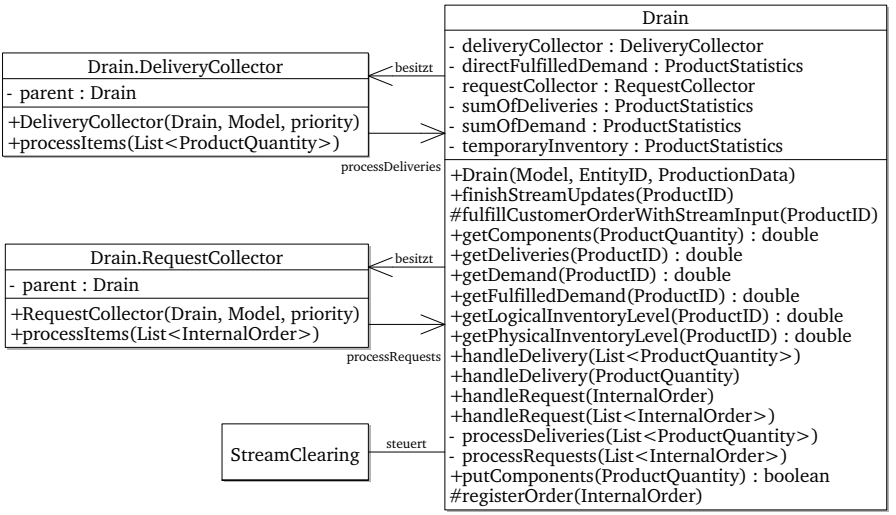


Abbildung 5.28: UML-Klassendiagramm *Drain*

zurückgreift. Abweichend vom vorgegebenen Verhalten akzeptiert eine Senke nur eingehende Materialflüsse und keine Ausgehenden. Da eine Senke im Gegensatz zu einem Lager nicht bestandsführend ist, können keine Informationen über Lagerbestände angegeben werden. Entsprechende Aufrufe erzeugen eine *UnsupportedOperationException*.

Als Erweiterung zu dem in Abschnitt 5.8.4 aufgeführten Verhalten bei eingehenden Materialflüssen implementiert diese Klasse die Methode *fulfillCustomerOrderWithStreamInput*. Immer wenn Aufträge und eingehende Materialflüsse vorliegen, wird der Zeitpunkt berechnet, an dem der nächste wartende Auftrag erfüllt werden kann. Als Reaktion auf das hier ausgelöste Ereignis wird *fulfillCustomerOrderWithStream-Input* aufgerufen. Somit ist sichergestellt, daß externe Aufträge erfüllt werden können, wenn Materialflüsse durch eine Senke das System verlassen.

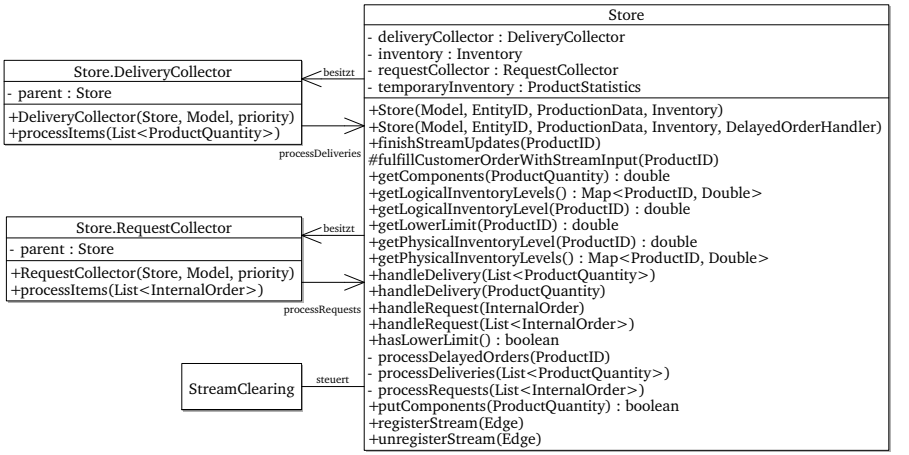


Abbildung 5.29: UML-Klassendiagramm *Store*

Store

Ein Lager ist ein Knoten innerhalb dessen beliebige/begrenzte Mengen eines oder mehrerer Produkte zwischengespeichert werden können. Ein Lager ist ein passives Element, d. h. es löst keine Materialflüsse aus, sondern ist nur an ihnen beteiligt.

Sowohl *Drain*, als auch *Store* werden von *AbstractWarehouseNode* abgeleitet. Im Gegensatz zu *Drain* handelt es sich um einen bestandsführenden Knoten, so daß Informationen über den disponiblen und physischen Lagerbestand oder auch über den Bestand an Nachlieferungen abgefragt werden können.

Die Anbindung an andere Klassen zeigt Abbildung 5.29. Die Methoden zum asynchronen Materialtransport sind identisch zu denen von *Source* implementiert. Die Implementierung verwendet private Klassen, die Nachkommen von *AbstractItemCollector* sind.

Die Verwaltung der Materialflüsse erfolgt durch die übergeordnete Klasse *AbstractWarehouseNode*, welche auf ein *StreamClearing* Objekt zurückgreift. Abweichend vom vorgegebenen Verhalten akzeptiert ein Lager sowohl eingehende, als auch ausgehende Materialflüsse. Durch

die Bestandsführung ist es im Gegensatz zu den anderen Knoten auch möglich, daß der Knoten eine Mengenbeschränkung in Form einer Ober-/Untergrenze unterliegt. Die entsprechenden Methoden überschreiben hier die Vorgabewerte von *AbstractNode*.

Vergleichbar mit dem Verhalten von *AbstractWarehouseNode* und *Drain* muß auch ein Lagerknoten wartende Aufträge überwachen, während Materialflüsse aktiv sind. Sobald der Nettozufluß ausreicht, um den nächsten Auftrag zu erfüllen, wird *fulfillCustomerOrderWithStreamInput* aufgerufen. Als Reaktion hierauf werden die Materialflüsse aktualisiert und anschließend die Verarbeitung der wartenden Aufträge ausgelöst.

Ressource, Lot und SetupOperation

Eine *Ressource* entspricht in einer Materialflußsimulation immer einer Anlage, die innerhalb des Produktionsnetzwerkes eingesetzt wird. Eine *Ressource* bezeichnet einen Knoten, der mit Hilfe eines Rezeptes beliebige diskrete und/oder kontinuierliche ausgehende Materialflüsse erzeugen kann. Zur Erzeugung dieser Materialflüsse kann - muß jedoch nicht - mindestens ein eingehender Materialfluß vorhanden sein.

Das Klassendiagramm auf der nächsten Seite zeigt die mit *Ressource* verbundenen Klassen. Zu Beginn einer Produktion wird aus dem aktuellen Zustand, dem vorangehenden Los (*precedingLot*) und zu fertigenden Los die Rüstzeit ermittelt. Anschließend wird eine *SetupOperation* Entität erzeugt, die sich auf das Ende der Rüstzeit terminiert. Die Maschine wird während des Rüstvorgangs in den Zustand *SETUP* versetzt. Nach Ende des Umrüstens informiert die *SetupOperation* die *Ressource* über *setupFinished*, worauf die Maschine eine *Lot* Entität erzeugt und in den Zustand *WORKING* versetzt wird. Die *Lot* Entität übernimmt Informationen über Fertigungsmenge, das Rezept sowie weitere Daten und terminiert ein Ereignis auf das Ende der Produktion. Dieses ruft *finishProduction* auf, so daß alle Materialflüsse aktualisiert werden und die Maschine in den Zustand *IDLE* übergeht.

Da eine Ressource keine Bestände führt, lösen alle Aufrufe die Informationen zu Beständen liefern eine *Exception* aus. Gleiches gilt auch bei Aufrufen von Methoden, die für den asynchronen Materialtrans-

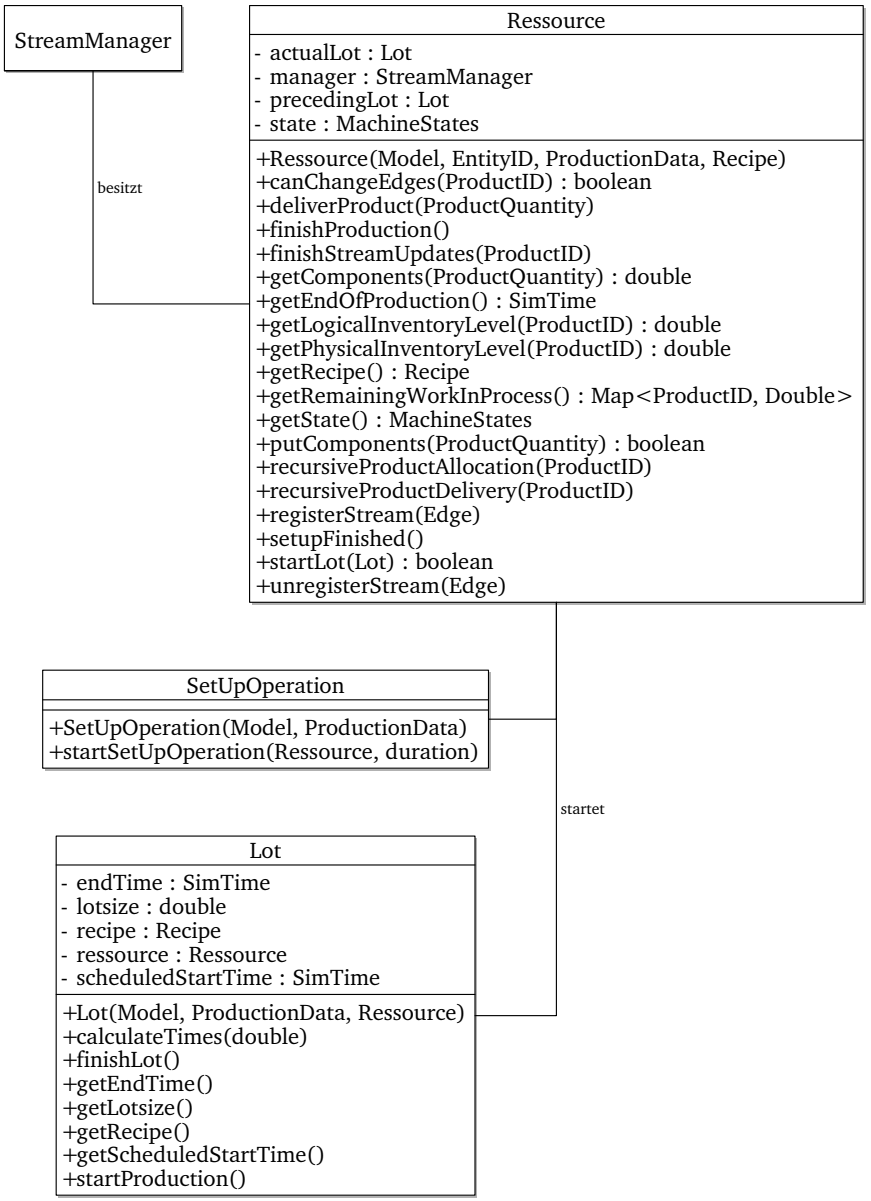


Abbildung 5.30: Anbindung von *Ressource*, *Lot* und *SetupOperation*

port verantwortlich sind. Als aktiver Baustein wird die Erzeugung und Verwaltung der Materialflüsse von einer *StreamManager* Instanz durchgeführt. Alle in *Node* definierten Methoden die Materialflüsse betreffen, werden per Delegation weitergeleitet.

5.8.5 Synchroner Transport über Materialflüsse

Ein synchroner Transport stellt das Material dem Aufrufer unmittelbar bereit, wenn es zur Verfügung steht. Anfrage und Lieferung werden nicht auf zwei Ereignisse verteilt. Alle Algorithmen, die in den Abschnitten 5.4.2, 5.5 und 5.6 für die Simulation von kontinuierlichen Materialflüssen vorgestellt wurden, verwenden intern den synchronen Transport.

Ein synchroner Transport findet immer statt, wenn eine Zustandsaktualisierung nötig ist, wie sie Abschnitt 5.6 auf Seite 133 beschreibt. Die Zustandsaktualisierung wurde durch ein Ereignis aufgerufen und erwartet, daß am Ende des Aufrufs alle Zustände den aktuellen Wert angeben. Ein asynchroner Materialtransport über Ereignisse ist somit ausgeschlossen.

Zum Einsatz kommt der in den Abbildungen 5.6 bis 5.7 auf den Seiten 130–131 vorgestellte Algorithmus. Der wesentliche Teil der Implementierung erfolgt in den Kanten, sowie den *StreamClearing* und *StreamManager* Klassen. Die Bausteine besitzen eine Instanz der Klasse und leiten alle in der Schnittstelle *Node* definierten Aufrufe an diese weiter.

Kanten

Der synchrone Materialtransport innerhalb des Produktionsnetzwerkes findet auf den Kanten zwischen den Knoten statt. Die Kanten werden dynamisch erzeugt, sobald ein aktiver Baustein eine Produktion auslöst. Entsprechend ihrem Charakter wird für jede Rezeptkomponente eine diskrete oder kontinuierliche Kante erzeugt. Daher können zwischen zwei Bausteinen auch mehrere Kanten erstellt werden, wenn der Baustein am Transport mehrerer Komponenten beteiligt ist.

Bei der Erzeugung wird weiter unterschieden, ob es sich aus Sicht des aktiven Bausteins um einen eingehenden oder ausgehenden Materialfluß handelt, da das Verhalten auch von der Flußrichtung abhängig ist. Mit Aufbau einer Kante wird die Information über den Vorgänger-, Nachfolgerknoten vom aktiven Baustein übernommen. Die Strukturinformation bleibt während der gesamten Produktionsdauer unveränderlich, kann jedoch während eines Simulationslaufs geändert werden, um z. B. eine neue Verrohrung von chemischen Anlagen nachzubilden. Um zu verhindern, daß die Verbindungen verändert werden, während ein Materialfluß aktiv ist, überprüft jeder Baustein anhand der aktiven Flüsse, ob eine Änderung zulässig ist.

Ausgangspunkt für die in Abbildung 5.31 auf der nächsten Seite gezeigte Klassenhierarchie ist die Schnittstelle *Edge*. Eine abstrakte Implementierung wird in *AbstractEdge* bereitgestellt. Die Klasse verwaltet alle Informationen über das Rezept, den Rezeptbestandteil, das Produktionslos sowie die Ressource. Zum Zeitpunkt der Erstellung wird im Konstruktor die aktuelle Zeit ermittelt und gespeichert. Sie dient als Nullpunkt für alle Berechnungen der kontinuierlichen Materialflüsse, wie sie in Abschnitt 5.4.2 vorgestellt wurden.

AbstractEdge wird direkt von *ModelComponent* abgeleitet, da sie nur auf die Infrastruktur vom Modell zugreift. Eine Ableitung von einer spezialisierteren Klasse wie *ProductionEntity* wäre auch möglich, würde aber zu einer Inkonsistenz im Design führen. Kanten werden innerhalb eines Netzwerkes für den synchronen Transport eingesetzt, der ohne Ereignisse durchgeführt wird. Daher müssen die beteiligten Komponenten nicht in der Lage sein, Ereignisse zu verarbeiten oder das Ziel von Ereignissen zu sein und somit auch nicht von *Entity* abstammen.

Nachkommen der abstrakten Oberklassen sind die Klassen *DiscreteMaterialFlowEdge* und *ContinuousMaterialFlowEdge*. Die in Abschnitt 5.4 erläuterten Unterschiede zwischen den beiden Arten von Materialflüssen werden hier umgesetzt. Die Unterscheidung zwischen ein- und ausgehenden Flüssen wird von einer statischen Factory-Methode getroffen, die beide Klassen bereitstellen. Intern werden in Abhängigkeit von der Flußrichtung Instanzen von privaten Klassen erzeugt, um ein- und ausgehende Materialflüsse zu modellieren. Die Erkennung der

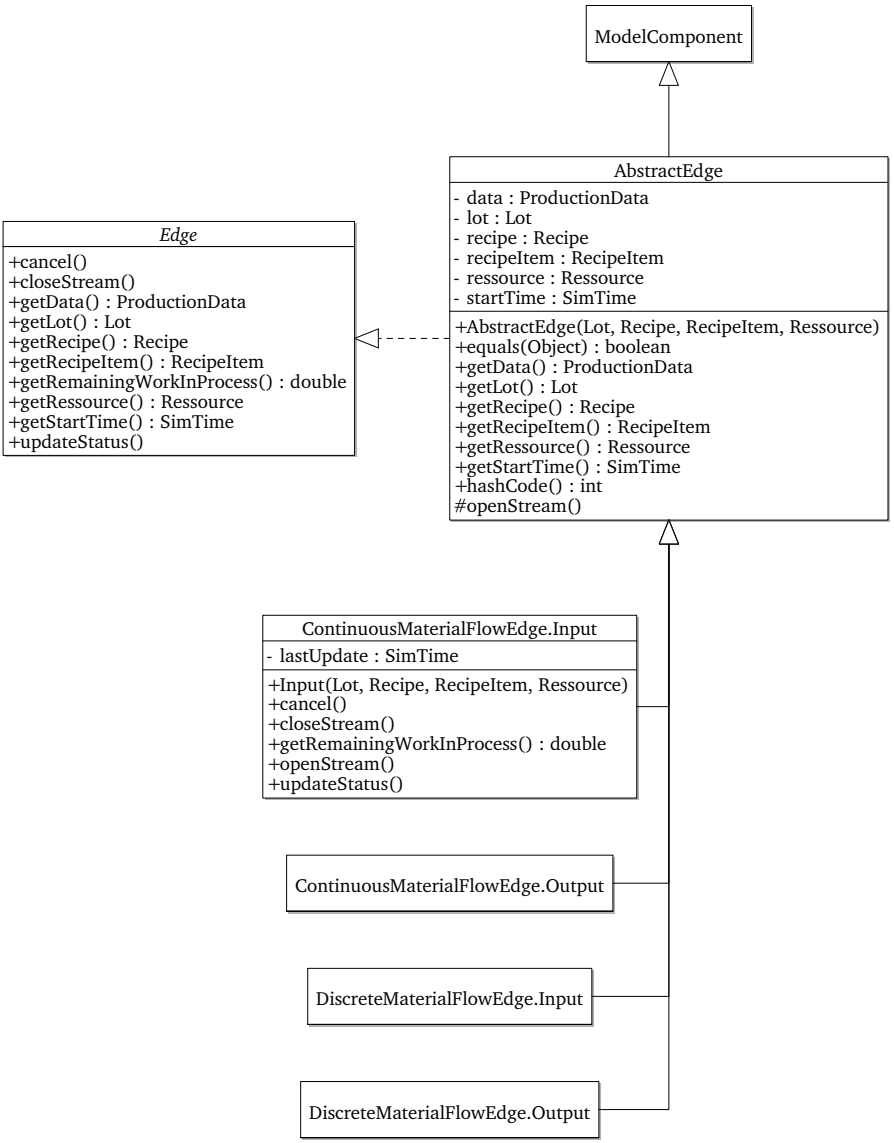


Abbildung 5.31: UML-Klassendiagramm der Schnittstelle *Edge* und abgeleiteter Klassen

Flußrichtung erfolgt mit Hilfe des Rezeptes, welches alle Komponenten mit ihrer Flußrichtung bezogen auf den aktiven Baustein beinhaltet.

Im Gegensatz zu einer diskreten Kante muß eine kontinuierliche Kante auch noch den Zeitpunkt speichern, an dem die letzte Aktualisierung des Zustandes erfolgt ist. So ist sichergestellt, daß bei einer Aktualisierung nur die Materialmenge transportiert wird, die seit dem letzten Transport angefallen ist, und nicht die gesamte Menge seit Beginn der Produktion.

StreamManager

Die Erzeugung und Verwaltung von Materialflüssen für aktive Bausteine führt die Klasse *StreamManager* durch. Abbildung 5.32 auf der folgenden Seite zeigt die Struktur der Klasse.

Zu Beginn einer Produktion erzeugt eine *Ressource* einen neuen *StreamManager* und übergibt ihm die Informationen über das zu fertigende Los. Diese enthält die Informationen über das Rezept und die Rezeptkomponenten. Für jede Komponente wird eine Kante erzeugt, der Vorgänger- und Nachfolgerknoten gesucht und die Kante im Vorgänger registriert. Durch die Registrierung wird die Verbindung zwischen den beiden Knoten aktiviert, so daß die *StreamClearing* Objekte auf der anderen Seite die Materialbewegungen an die Bausteine weiterleiten können. Am Ende der Produktion werden alle Kanten geschlossen und die *StreamClearing* Stellen über die Schließung informiert.

StreamClearing

Die *StreamClearing* Klasse wird von passiven Bausteinen zur Verwaltung von Materialflüssen eingesetzt. Sie stellt die Gegenseite zu *StreamManager* Objekten dar, die Materialflüsse erzeugen und sie hier registrieren. Im Gegensatz zu *StreamManager* erzeugt jeder Baustein eine Instanz, die er während der gesamten Laufzeit der Simulation beibehält. Durch diese dauerhafte Bindung an einen Baustein und die Verwendung der Modellinfrastruktur wurde die Klasse von *ModelComponent* abgeleitet.

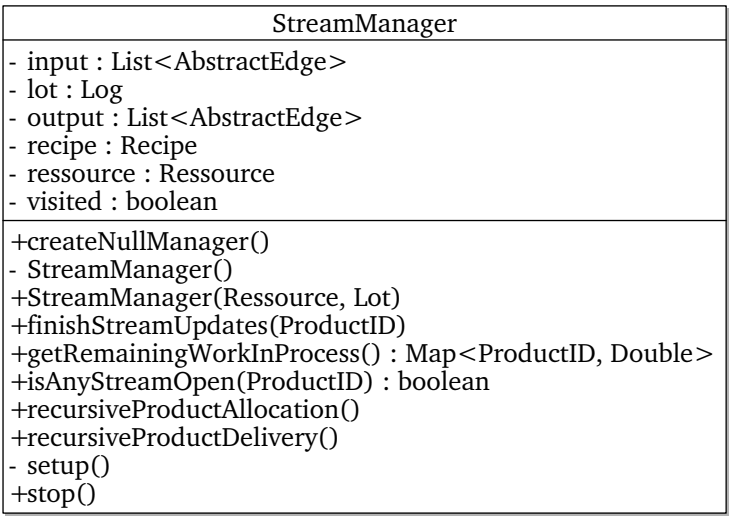


Abbildung 5.32: UML-Klassendiagramm *StreamManager*

Hauptaufgabe der Klasse ist die Berechnung der Grenzzustände. Bei jeder An-/Abmeldung eines Materialflusses wird die Netto-Materialflußfunktion des Bausteins für das Produkt neu berechnet (vgl. Abschnitt 5.7.2 auf Seite 140). Besitzt ein Baustein eine Ober- oder Untergrenze, so wird der Zeitpunkt berechnet, an dem der Grenzzustand eintritt. Intern wird auf diesen Zeitpunkt ein Ereignis terminiert, welches beim Aufruf eine *Exception* auslöst. Die Ereignisse laufen über eine spezielle *AlarmClock*, die nach außen nicht sichtbar ist. Die Verwaltung der *AlarmClock* Instanzen erfolgt durch einen *AlarmClockManager*. Für jedes Produkt, das über einen der aktiven Ströme bewegt wird, existiert ein *AlarmClock* Objekt.

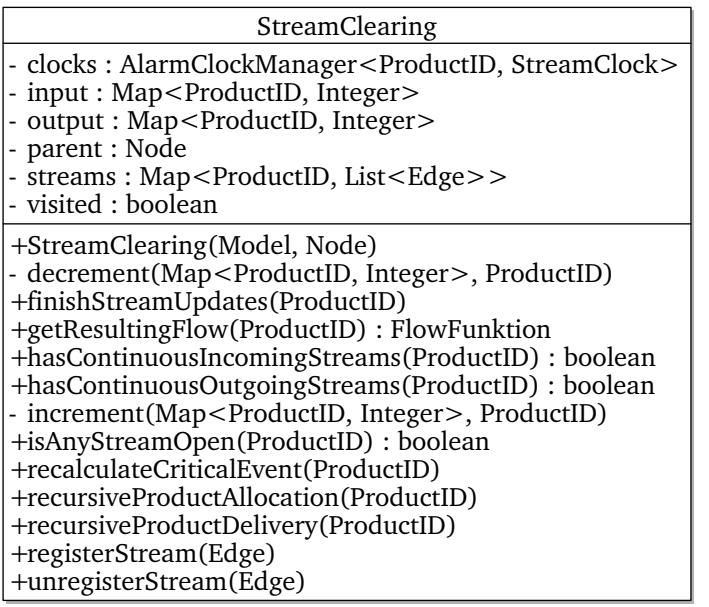


Abbildung 5.33: UML-Klassendiagramm *StreamClearing*

Teil III

Anwendung

Kapitel 6

Versuchsaufbau

Gegenstand der Untersuchungen sind die Auswirkungen von Nachfrageunsicherheit auf den Servicegrad in einem hierarchischen Planungssystem. Die Planungsabläufe entsprechen denen, die auch in der Praxis eingesetzt werden.

Das Kapitel gliedert sich in zwei Abschnitte - einen allgemeinen Teil, der die hierarchische Planung vorstellt sowie die aktuellen Arbeiten zu diesem Bereich, und einen speziellen Teil, der die in dieser Arbeit eingesetzten Modelle und Algorithmen vorstellt.

Der ersten Abschnitt gibt einen Überblick über die in der Literatur behandelten Planungsprobleme und die daraus entwickelten Planungsebenen. Da die Untersuchung mit Hilfe von Simulations- und Planungsmodellen durchgeführt wird, stellt der darauf folgende Teilabschnitt verschiedene Ansätze zur Computergestützten Planung vor, und legt einen Schwerpunkt auf die Material Requirements Planning (MRP) Systeme und ihre Nachfolger. Anschließend werden verschiedene Konzepte zur Kompensation der Unsicherheit in der Planung beschrieben, sowie aktuellen Anwendungen der Simulation von Planungssystemen.

Nach einer kurzen Vorstellung der Versuchsumgebung werden die eingesetzten Module und Planungsebenen detailliert mit den eingesetzten Modellen und Heuristiken behandelt. Analytische Modelle lassen sich auf diese Problemstellung nur sehr schwer oder auch gar nicht

anwenden, da diese meist von vereinfachten Prämissen ausgehen. Diese Arbeit soll jedoch Effekte untersuchen, die durch den Einsatz von gemischt-ganzzahligen Modellen entstehen, wie sie auch in der Praxis zum Einsatz kommen. Die Komplexität der Erzeugnisstruktur wurde daher gering gehalten, um Effekte, die sich durch umfangreichere Erzeugnisstrukturen ergeben, auszuschließen. Die für die Untersuchung eingesetzten Testdaten werden von einem Testdatengenerator erzeugt, der in Kapitel 7 vorgestellt wird.

6.1 Literaturüberblick

6.1.1 Planungsebenen

Die Entscheidungen, welche Produkte zu welchem Zeitpunkt hergestellt und ausgeliefert werden, sind das Ergebnis einer Planung im Unternehmen. Die Art der Planung hat einen entscheidenden Einfluß auf die Qualität der Ergebnisse. Aufbauend auf den Arbeiten von Anthony (1965, S. 19) wird ein dreistufiges Schema verwendet.

Strategische Planung (Strategic Planning) umfaßt alle Entscheidungen eines Unternehmens über seine Ausrichtung und Positionierung. Hierzu gehören unter anderem Entscheidungen zur Produktpalette, Forschung und Standorte an denen produziert wird, sowie die Fertigungstechnologie. Der Planungshorizont reicht über mehrere Jahre.

Taktische Planung (Management Control) umfaßt alle Entscheidungen der Beschaffung sowie des effizienten Einsatzes von Ressourcen in Übereinstimmung mit den Zielen der Unternehmung (Anthony 1965, S. 17). Im Bereich der Produktion umfassen diese Entscheidungen den Personaleinsatz, die mittelfristige Fertigungskapazität und die Zuteilung der Ressourcen. Der Zeithorizont kann zwischen mehreren Wochen und Monaten, sowie in einigen Industrien auch im Bereich von wenigen Jahren liegen.

Zeithorizont	Länge	Entscheidungen
Langfristige Planung	Jahre bis Jahrzehnte	Produktdesign, Fertigungskapazität, Standorte, Fertigungstechnologie
Mittelfristige Planung	Wochen, Monate bis ein Jahr	Personaleinsatz, Zuteilung der Ressourcen, grober Fertigungsplan
Kurzfristige Planung	Stunden bis Wochen	Materialfluß, Rüstoperationen, Personaleinsatzplanung

Tabelle 6.1: Strategische, taktische und operative Planung - Planungshorizonte und Entscheidungen

Operative Planung(Operational Control) umfaßt alle Aktivitäten um sicherzustellen, daß einzelne Aufgaben effektiv und effizient umgesetzt werden (Anthony 1965, S. 18). Für die Produktion bedeutet dies die konkrete Erstellung und Umsetzung von Produktionsplänen mit den vorhandenen Ressourcen. Der Planungshorizont reicht von einigen Tagen bis zu einigen Wochen.

Diese drei Entscheidungsebenen werden als Ausgangspunkt für eine zeitliche Disaggregation der Planungsaufgaben benutzt. Die Aufspaltung mit ihren Entscheidungen zeigt Tabelle 6.1. Diese Art der Disaggregation ist eine von mehreren Möglichkeiten, die Planungsaufgaben zu strukturieren. Eine weitere Möglichkeiten ist die Zusammenfassung nach Produktgruppen oder Ressourcen (Hopp u. Spearman 2001, S. 411).

6.1.2 Computergestützte Planungssysteme

Ausgehend von den hier aufgeführten Aggregationsstufen wurde von Orlicky ein Material Requirements Planning-System (MRP) entwickelt (Orlicky 1975). Die Aufgabe eines MRP-Systems ist die Auflösung der Primärbedarfe in Nettobedarfe der Komponenten, sowie die Erzeugung von Produktions- oder Bestellaufträgen, um den Produktionsplan (der

Endprodukte) umzusetzen (Orlicky 1975, S. 21). Motivation war die Entwicklung einer Alternative zu stochastischen Lagerhaltungspolitiken in Bereichen, wo diese keine befriedigenden Ergebnisse erzielt haben (Orlicky 1975, S. 33). Der Einsatz von stochastischen Lagerhaltungspolitiken war im Bereich der Endprodukte gerechtfertigt, für die Komponenten der Produkte jedoch nicht zufriedenstellend. Das Verfahren war bei seiner Einführung eine Verbesserung gegenüber anderen Planungssystemen und durch die Abkehr von den stochastischen Lagerhaltungspolitiken auch gleichzeitig ein Paradigmenwechsel. Eine detaillierte Beschreibung der Verfahren der Bedarfsauflösung, sowie der Losbildung findet sich u. a. in Hopp u. Spearman (2001, S. 109ff). Durch die Beschränkung der Sichtweise auf die Auflösung der Bedarfe entstehen auch einige Nachteile (Hopp u. Spearman 2001, S. 131). So werden unter anderem die Vorlaufzeiten der Produkte als konstant angenommen und in die Planung mit einbezogen. Die aktuelle Auslastung der Anlagen wird nicht berücksichtigt. Da in vielen Fällen auch die Anreizsysteme der Planer so gestaltet sind, daß die Verspätungen gegenüber dem Kunden ein wesentlich höheres Gewicht haben als zu hohe Lagerbestände ist es immer von Vorteil, größere Vorlaufzeiten in die Planung einzubeziehen und dadurch auch die Lagerbestände zu erhöhen. Ein weiterer wesentlicher Kritikpunkt ist die Nichtbeachtung von Kapazitätsbeschränkungen. Diese werden in der Planung an keiner Stelle berücksichtigt, so daß sich nicht zulässige Pläne ergeben können. Auch die Rollierung und ständige Neuberechnung der Pläne führt zu einer Nervosität, d. h. im Vergleich zu einem vorangehenden Planungslauf kann der gesamten Produktionplan für die folgende Periode umgeworfen werden, so daß erst nach der Auslösung der Aufträge sichergestellt ist, daß sie auch wie geplant durchgeführt werden.

Aufgrund dieser Probleme wurde eine zweite Generation von Planungssystemen entwickelt, die MRP-II-Systeme. Ziel war die Beseitigung der genannten Problemfelder, indem weitere Bereiche in die Planung einbezogen werden. MRP-II steht jetzt für Manufacturing Resources Planning, womit die Bedeutung der Planung von Ressourcen unterstrichen wird. Ein MRP-II System enthält verschiedene Module, die miteinander interagieren. Die Abbildung auf Seite 192 zeigt die

wichtigsten Module. Details zur Funktionalität und zu den verwendeten Algorithmen finden sich in Vollmann u. a. (2005) und Hopp u. Spearman (2001, S. 136). Parallel zu Orlicky wurden von Hax u. Meal (1975, S. 55) ein System zur hierarchischen Integration der Produktionsplanung vorgestellt. Die Produkte werden je nach Planungsstufe aggregiert und einzelne Details, die auf unteren Ebenen in die Planung einbezogen werden, sind auf höheren Ebenen nicht modelliert. Die Pläne werden auf den unteren Stufen regelmäßig neu aufgelegt und an die aktuelle Datenlage angepaßt. Zur Durchführung der Planung werden Modelle eingesetzt, die ein Mengengerüst für die Planungsintervalle erzeugen und dabei bereits Kapazitäten von Fabrikstandorten (oder Anlagen) berücksichtigen. Die von Hax u. Meal vorgestellten Konzepte der Aggregation auf den Planungsebenen lassen sich teilweise in den MRP-II-Systemen (z. B. bei der Antizipation) und in den darauf aufbauenden Planungssystemen wiederfinden.

Mit zunehmender Rechner- und Datenspeicherungskapazität wurden weitere Teile von Unternehmen durch zusätzliche Module abgebildet. Die hieraus entstandenen Systeme fallen unter den Begriff Enterprise Resource Planning (ERP). ERP Systeme basieren auf MRP-II-Systemen und ergänzen diese um weitere Module für Bereiche, die nicht originär mit der Produktion zusammenhängen. ERP Systeme verfolgen einen integrativen Ansatz, indem alle Bereiche eines Unternehmens durch ein System abgebildet und verwaltet werden können (Hopp u. Spearman 2001, S. 143). Der letzte Entwicklungsschritt ist das APS.

Die charakteristischen Merkmale eines APS sind (vgl. Fleischmann u. a. 2008, S. 84):

- Integrative Planung der gesamten Lieferkette eines oder mehrerer Unternehmen
- Optimierung unter Berücksichtigung von Nebenbedingungen und einer (monetär) bewerteten Zielfunktion
- ein hierarchisches Planungssystem als Kompromiß zwischen einer Simultanplanung und Sukzessivplanungskonzepten.

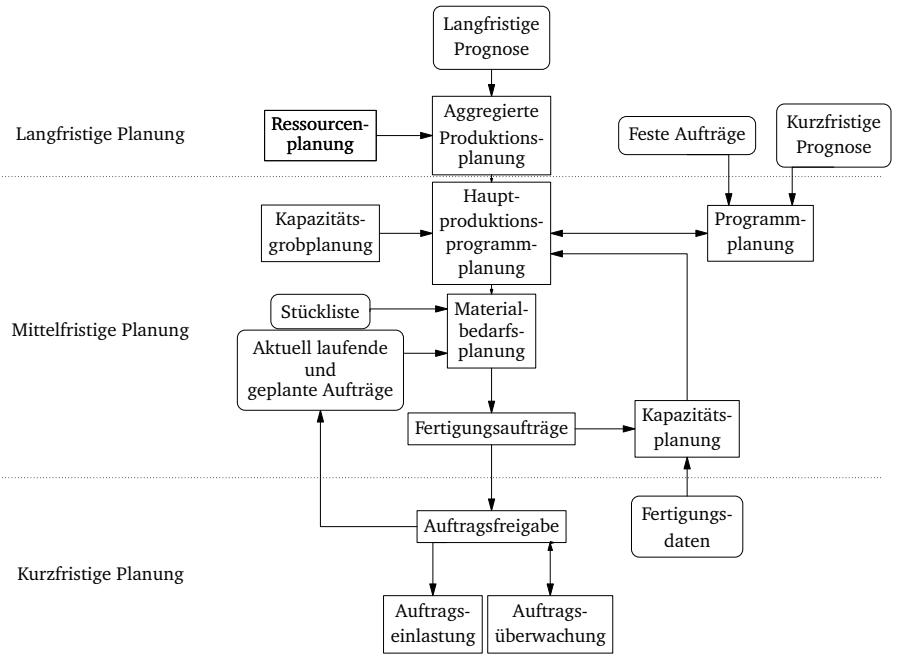


Abbildung 6.1: Aufbau eines MRP-II-Systems (vgl. Hopp u. Spearman 2001, S. 136, eigene Übersetzung)

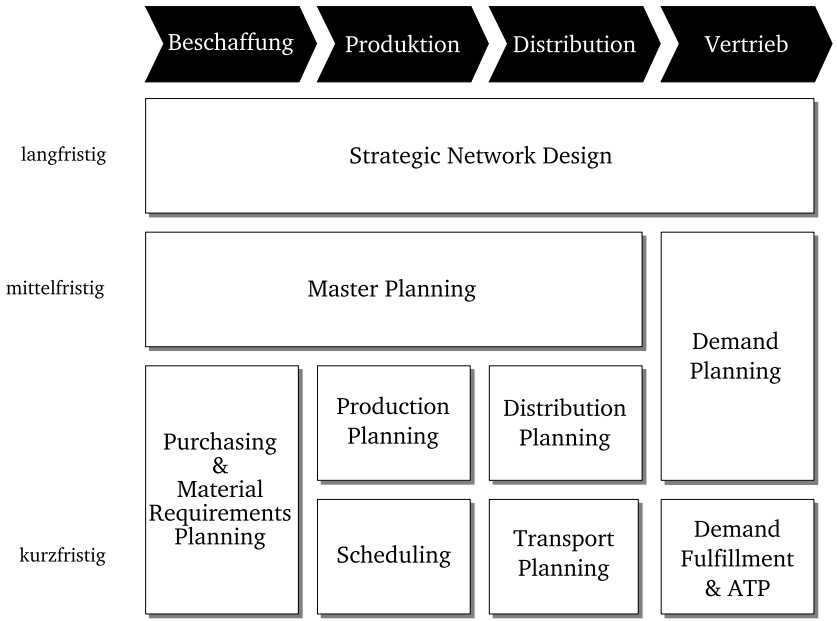


Abbildung 6.2: Software-Module der SCM-Matrix (vgl. Meyr u. a. 2008, S. 109, eigene Übersetzung)

Ein APS setzt die Modellierung durch ein ERP System voraus und bietet die Möglichkeit, für einzelne Bereiche eine Planung durchzuführen und diese aufeinander abzustimmen. Die Planungsbereiche/-module lassen sich anhand ihres Planungshorizonts und dem abzubildenden Prozeß unterteilen (vgl. Abbildung 6.2). Sie bilden so die gesamte Lieferkette innerhalb eines Unternehmens oder zwischen mehrerer Unternehmen ab.

Die Planung erfolgt im Gegensatz zu den MRP-Systemen nicht nur mit dem Anspruch, eine zulässige Lösung zu finden. Gleichzeitig soll diese Lösung bei einer (monetären) Bewertung durch eine Zielfunktion die beste (oder optimale) Lösung unter mehreren Alternativen darstellt. Hierfür werden unterschiedliche Meta-, Heuristiken und Algorithmen aus dem Bereich des Operations Research eingesetzt (Fleischmann u. a.

2008, S. 81ff). Eine systematische Einordnung der Planungsaufgaben in die Planungsbereiche und -horizonte wird in Fleischmann u. a. (2008, S. 87ff) vorgestellt.

6.1.3 Unsicherheit in Planungssystemen

Die Auswirkungen von Unsicherheit in Planungssystemen werden bereits seit der Entwicklung der ersten Planungssysteme untersucht. Bereits vor der systematischen Entwicklung von Planungssystemen wurden für Lagerhaltungssysteme Lagerhaltungspolitiken entwickelt. Die Entwicklungen auf diesem Gebiet gehen auf Clark u. Scarf (1960) zurück, die bei stochastischer Nachfrage eine optimale Politik für eine serielle Erzeugnisstruktur vorstellten. Seither wurde eine Vielzahl an Varianten untersucht und meist auch optimale oder nahezu optimale Lösungen gefunden. Einen guten Überblick der verschiedenen Entwicklungsrichtungen gibt Axsäter (2006). Innerhalb dieser Arbeit werden die Lagerhaltungspolitiken jedoch nicht weiter berücksichtigt, da sie nicht auf die spezifischen Eigenschaften von Produktionsplanungssystemen eingehen.

Nach Vorstellung der MRP-Systeme durch Orlicky (1975) wurde von Whybark u. Williams (1976) die Unsicherheit in MRP-Systemen betrachtet. Die Unsicherheit klassifizieren Whybark u. Williams anhand von zwei Kriterien: die Seite auf der die Unsicherheit auftritt (Nachfrage oder Lieferungen) und der Gegenstand der Unsicherheit (Zeit oder Menge) (Whybark u. Williams 1976, S. 598). Mit Hilfe einer Simulation wird untersucht, welcher Servicegrad erreicht wird, wenn entweder Sicherheitsbestände oder Sicherheitsvorlaufzeiten im MRP benutzt werden. Falls die Unsicherheit die zeitliche Dimension betrifft, so sind Sicherheitsvorlaufzeiten zu bevorzugen. Umgekehrt sind bei einer Unsicherheit in der Menge Sicherheitsbestände zu halten. Diese Aussagen wurden durch Tests mit unterschiedlichen Einstellungen im MRP und statistischen Tests bestätigt.

Chang (1985, S. 40) zeigt, daß Sicherheitsbestände und Sicherheitsvorlaufzeit generell gegeneinander austauschbar sind. Für eine Ersetzung der Sicherheitsbestände in voller Höhe sind jedoch zwei Vor-

aussetzungen notwendig: die Höhe der zusätzlichen Nachfrage muß bekannt sein, bevor auf der ersten Produktionsstufe die Produktion begonnen wird; das Material für die zusätzliche Produktion muß auf allen Stufen verfügbar sein. Da beide Voraussetzungen in der Praxis meist nicht gegeben sind, ergibt sich die Präferenz, wie sie von Whybark u. Williams gezeigt wurde.

Weitere Möglichkeiten, der Unsicherheit in der Planung zu begegnen, werden von Schmitt (1984) untersucht. Außer Sicherheitsbeständen wird der Einsatz von Sicherheitskapazitäten untersucht, sowie eine kontinuierliche Aktualisierung der Pläne („net change“). Letztere ist nur möglich, wenn wie bei Chang die Änderungen rechtzeitig erkannt werden, so daß die Pläne noch angepaßt werden können. Auch hier ergibt sich eine Präferenz für Sicherheitsbestände, da Sicherheitskapazität die rechtzeitige Kenntnis von Mengenabweichungen voraussetzt, um Pläne anzupassen. Gegen eine kontinuierliche Aktualisierung der Pläne spricht die zusätzliche Nervosität, die im System erzeugt wird, da bestehende Pläne ständigen Änderungen unterworfen sind (Schmitt 1984, S. 342).

Eine analytische Betrachtung der Unsicherheit in einem MRP-System wird von Wacker (1985) durchgeführt. Wacker betrachtet die Unsicherheiten, die von Whybark u. Williams (1976, S. 598) definiert wurden, und zeigt, wie sich diese im Planungssystem über Produktionsstufen hinweg ausbreiten.

Ein erster analytischer Ansatz zur Prognose des Servicegrades wird von Anderson u. Lagodimos (1989) entwickelt. Sie verwenden hierfür ein MRP-System, welches auf einen vorher berechneten Hauptproduktionsprogrammplan zurückgreift und zur Freigabe der Lose eine (s,S) Politik einsetzen. Die Nachfrage ist stochastisch und wird durch eine Normalverteilung abgebildet. Zur Untersuchung wird eine einstufige Erzeugnisstruktur eingesetzt. Die unter diesen Voraussetzungen abgeleiteten Formeln ließen sich durch eine Simulation bestätigen.

Eine weitere Untersuchung von Buzacott u. Shanthikumar (1994) zeigt wie auch schon Whybark u. Williams (1976), daß Sicherheitsbestände den Sicherheitsvorlaufzeiten vorzuziehen sind. Weiterhin wird

der Einfluß der Prognosegüte untersucht, die sich auf die Höhe der Unsicherheit und damit auch auf den Servicegrad auswirkt.

Koh (2004) untersucht den Einsatz eines MRP/MRP-II in einer mehrstufigen Fertigung mit Batchrestriktionen. Die Auftragsfreigabe erfolgt über eine Prioritätsregel. Die Anzahl der Maschinen und Produkte ist hier wesentlich höher als in anderen Untersuchungen. Auch der Einsatz von Prioritätsregeln (nicht ausschließlich FIFO) zur Freigabe ist neu. Unsicherheiten werden in verschiedenen Ausprägungen und Arten simuliert, vergleichbar mit Whybark u. Williams (1976). Durch statistische Tests werden Aussagen über Einflußfaktoren gewonnen, die den Servicegrad beeinflussen. Durch die Untersuchung mehrerer Faktoren wird gezeigt, daß Faktoren, die alleine keinen signifikanten Einfluß haben, in Kombination einen signifikanten Einfluß entwickeln.

Eine andere Forschungsrichtung benutzt Modelle und Algorithmen der robusten Planung und Optimierung, um anhand von vorher definierten Szenarien eine robuste Lösung zu erhalten (Scholl 2001). Ein aktueller Ansatz im Bereich der hierarchischen Planung wird von Kuhn u. Gebhard (2008, S. 99) vorgestellt. In diesem Modell wird ein zweistufiges Planungssystem verwendet. Die aggregierte Planung stellt Informationen über saisonale Lagerbestände sowie verfügbare Überstunden der Losgrößenplanung zur Verfügung. Nach Lösung des Losgrößenmodells ergeben sich die tatsächlichen Lagerbestände und die Rüstzeiten, die vom aggregierten Modell in der nächsten Iteration berücksichtigt werden (Kuhn u. Gebhard 2008, S. 107). Die Ergebnisse zeigen, daß im Gegensatz zur klassischen Planung die Lagerkosten höher sind und die Rüstkosten sinken. Insgesamt tritt jedoch eine Kostensenkung ein, da die Rüstkosten stärker sinken als die Lagerkosten steigen. Dies ist darauf zurückzuführen, daß die Produktionspläne stabiler geworden sind und tendenziell längere Lose produziert werden.

Ein verwandtes Forschungsgebiet ist Entwicklung von Modellen, die Unsicherheit bereits in der Planung berücksichtigen. Spitter (2005) untersucht hierzu die Auswirkungen von Vorlaufzeiten auf den Servicegrad im Kontext einer rollierenden Planung. Die Vorlaufzeiten werden jedoch nicht als feste Grenze, sondern als Intervall definiert, in dem die Produktion zu erfolgen hat. Durch diesen zusätzlichen Freiheitsgrad

läßt sich ein besserer Servicegrad erreichen, als bei klassischen Modellen. Die Arbeit untersucht auch einen weiteren Aspekt - den Vergleich zwischen einem analytischen „Base-Stock“ System und einem äquivalenten MRP-System mit rollierender Planung. Es wird gezeigt, daß durch ein Modell mit einer quadratischen Zielfunktion Lösungen berechnet werden können, die denen der analytischen Modelle entsprechen. Durch quadratische Terme für die Lagerbestände und Nachlieferungen wird die Selektion der linearen Modelle und ihren Präferenzen für einzelne Produkte so weit aufgehoben, daß die Kostendifferenz zu Base-Stock Modelle auf einen geringen einstelligen Prozentbereich sinkt. Somit konnte gezeigt werden, daß die Ergebnisse der Base-Stock-Modelle sich mit modifizierten Modellen der linearen Programmierung reproduzieren lassen und beide Ansätze in ihren Ergebnissen nahezu gleichwertig sind. Der erhöhte Rechenaufwand für quadratische Modelle spricht aber noch gegen den breiten Einsatz dieser Modellklasse.

Einen aktuellen Überblick über den Stand der Forschung im Bereich MRP/MRP-II und Unsicherheit gibt Mula u. a. (2006). Dort wird auch eine Klassifizierung der Literatur sowie der Forschungsrichtungen und Ansätzen durchgeführt.

6.1.4 Rollierende Planung

Die in der Produktionsplanung eingesetzten Modelle besitzen einen endlichen Horizont, für den angenommen wird, daß die Daten vollständig bekannt sind. Baker (1977) führt zwei wesentliche Gründe auf, warum eine Begrenzung des Plans notwendig ist: Die Daten sind mit einer Unsicherheit behaftet, so daß nur mit einem begrenzten Planungshorizont gearbeitet werden kann. Ein weiterer Grund ist die Beschränkung der Planung und Datenerfassung auf den angenommenen Operationsbereich des Unternehmens. Baker untersucht, welchen Einfluß die Größe des Zeitfensters hat, mit dem eine Planung auf einem endlichen Horizont durchgeführt wird. Die Produktionsentscheidung wird nur für die erste Periode fixiert und danach wird das Zeitfenster verschoben und der Plan erneut berechnet. Die Ergebnisse zeigen, daß die Abweichungen vom Optimum - wenn die Daten für den gesamten Horizont im

Voraus exakt bekannt sind - gering sind und für die Praxis eine ausreichende Genauigkeit besitzen. Die Länge des Zeitfensters hängt sowohl von der Qualität der Daten ab, als auch von der TBO, die sich aufgrund der Economic Order Quantity (EOQ) ergibt (Baker 1977, S. 26). Es wird nachgewiesen, daß für eine vorausschauende Planung die Länge des Zeitfensters die TBO übersteigen muß. Die Arbeit zeigt aber auch, daß das Nachfrageprofil einen starken Einfluß auf die erreichte Genauigkeit besitzt. Die Ergebnisse werden von einer darauf aufbauenden Arbeit noch analytisch unterstützt (Baker u. Peterson 1979, S. 345). Die Untersuchung von Carlson u. a. (1982, S. 144) bestätigt im Wesentlichen die Ergebnisse von Baker, präsentiert jedoch auch Fälle, in denen mit einer weiteren Verlängerung des Zeitfensters bessere Ergebnisse erzielt werden können. Diese betreffen alle Nachfragemuster, die Saisonmuster oder stark schwankenden Muster aufweisen.

Der Einsatz der rollierenden Planung in einem MRP-System wird von de Bodt u. a. (1982, S. 67) untersucht. Die Unsicherheit in der Nachfrage beeinflußt die Planung nachhaltig. Unabhängig vom eingesetzten Algorithmus zur Bildung der Losgrößen führen die Schwankungen zu starken Abweichungen von der optimalen Lösung (bei Kenntnis der gesamten Nachfrage im Voraus). Es läßt sich sogar in einigen wenigen Fällen zeigen, daß der Einsatz der festen Losgröße auf Basis der EOQ zu besseren Ergebnissen führt als die Anwendung von Losgrößenheuristiken (de Bodt u. a. 1982, S. 74). Ähnliche Ergebnisse werden von Blackburn u. Millen (1982, S. 130) vorgestellt. Auch hier zeigt sich, daß eine Losgrößenheuristik im rollierenden Kontext zu geringeren Kosten führt, als der Einsatz eines exakten Losgrößenmodells. Gezeigt wurde dies für den mehrstufigen Fall, mit einem Endprodukt in verschiedenen linearen und konvergierenden Erzeugnisstrukturen und mehreren Kombinationen aus Lagerkostensätzen und Rüstkosten.

Einen Vergleich von verschiedenen Losgrößenheuristiken und exakten Verfahren in einer rollierenden Planung führen Wemmerlöv u. Whybark (1984, S. 467) durch. Sie vergleichen die Algorithmen ohne und mit Unsicherheit in Form von Prognosefehlern. Untersucht wird eine einstufige Produktion mit einem Produkt. Durch Simulation wird

gezeigt, daß exakte Verfahren unter Unsicherheit höhere Kosten erzeugen als Heuristiken, während es ohne Unsicherheit umgekehrt ist.

Weitere Untersuchungen zum Einsatz von Losgrößenmodellen auf rollierender Basis werden von Federgruen u. Tzur (1994) durchgeführt. Diese enthalten Methoden zur Bestimmung der Länge eines Zeitfensters anhand von bestehenden Daten oder Genauigkeitskriterien. Federgruen u. Tzur (1994, S. 457) beschreiben auch die Ursache für die Nervosität der Pläne - aufgrund der Rollierung werden die vorher ungenauen Prognosedaten aktualisiert und durch die Verschiebung kommen weitere Daten hinzu, die in die Planung neu einbezogen werden. Ein weiteres Problem stellt die sogenannte „terminal condition“ der Losgrößenmodelle dar. Diese besagt, daß der Lagerendbestand gleich null ist. Somit wird in einem rollierenden Kontext die zu produzierende Menge gegen Ende des Zeitfensters immer unterschätzt (Baker u. Peterson 1979, S. 348). Um diesem Problem zu begegnen, wurde von Stadtler (2000, S. 320) eine Modifikation vorgeschlagen. Die Rüstkosten werden innerhalb des Zeitfensters gleichmäßig auf ein Intervall verteilt, dessen Länge der TBO entspricht, die aus der EOQ-Formel berechnet wurde. Somit können auch Rüstoperationen am Ende eines Zeitfensters wirtschaftlich erfolgen. Es wird gezeigt, daß durch diese Modifikation der Einsatz von Losgrößenmodellen dem von Heuristiken durch geringere Kosten überlegen ist.

6.1.5 Simulation von Planungssystemen - aktuelle Anwendungen

Die bisher vorgestellten Arbeiten haben sich mit den Eigenschaften der Planungssysteme oder mit Effekten, die bei der Rollierung innerhalb der Planungssysteme auftreten, beschäftigt. Analytische Ansätze zur Beschreibung des Verhaltens gehen meist davon aus, daß die Erzeugung von Produktionsplänen nicht von Losgrößenmodellen durchgeführt wird, sondern durch den Einsatz von Lagerhaltungspolitiken erfolgt. Die hierauf basierende Analyse läßt sich von den bereits bekannten Ergebnissen der Lagerhaltungspolitiken ableiten.

Im Gegensatz zu den analytischen Ansätzen werden speziell in komplexen Umgebungen simulationsbasierte Ansätze eingesetzt. Diese Ansätze verwenden verschiedene Methoden, um iterativ durch Veränderung von einem oder mehreren Eingabeparametern einen Zielwert zu erreichen (simulationsbasierte Optimierung). Eine Anwendung aus dem Bereich der chemischen Industrie wird von Jung u. a. (2004) vorgestellt. Die Simulation wird hier eingesetzt, um die Unsicherheit der Kundennachfrage zu simulieren. Ziel ist es, die Höhe des Sicherheitsbestandes zu bestimmen, so daß ein vorgegebener Servicegrad erreicht wird. Die Planung erfolgt auf einem Zeitfenster, welches rollierend über den gesamten Horizont verschoben wird. Nach Beendigung eines Durchlaufs werden die Ergebnisse ausgewertet und die Sicherheitsbestände entsprechend angepaßt. Die betrachtete Produktionsanlage ist mehrstufig, linear auf den ersten Stufen und divergierend auf der letzten Stufe. Die Komplexität der Planung ist entsprechend höher, als bei den bisher betrachteten einstufigen Modellen. Da auch ein vereinfachtes Planungsmodell während der Rollierung mehrfach ausgeführt wird, ist der Rechenaufwand entsprechend hoch. Aufgrund der hohen Rechenzeit eignet sich dieser Ansatz eher für die strategische Planung und weniger für die taktische Planung, in der auf mittelfristiger Ebene Sicherheitsbestände festgelegt werden können. In einer Erweiterung des gleichen Problems werden auch die Abweichungen vom Sicherheitsbestand in Form des Servicegrades berücksichtigt (Jung u. a. 2008). Mit Hilfe der gleichen Methoden werden zusätzliche Fragestellungen untersucht, die sich durch den Gebrauch des Servicegrades in der Planung ergeben.

Eine weitere Anwendung der simulationsbasierten Optimierung wird von Boulaksil u. a. (2009, S. 121) vorgestellt. Boulaksil u. a. betrachten ein mehrstufiges Lagerhaltungssystem, dessen Aufträge mit Hilfe eines linearen Optimierungsmodells Modells gesteuert wird. Das Modell berechnet die Auftragsgrößen und -zeitpunkte, die von einem Produktionssystem erfüllt werden müssen (dieses wird hier nicht weiter betrachtet). Durch Einsatz des Modells können auch Erzeugnisstrukturen betrachtet werden, die keine lineare Struktur aufweisen. Die Planung erfolgt rollierend, wobei die erste Periode nach Abschluß der Planung fixiert wird. Die Zielfunktion beinhaltet Strafkosten für nicht

gelieferte und später nachgelieferte Mengen. Strafkosten für die Unterschreitung von Sicherheitsbeständen dürfen nicht enthalten sein, da diese das Ziel der Planung sind. Boulaksil u. a. (2009, S. 129) schlagen vor, daß zunächst das Modell ohne Sicherheitsbestände simuliert wird. Hieraus ergeben sich auf einem Zeitraum von n Perioden ein Lagerbestandsverlauf und eine Verteilung der Nachlieferungen am Beginn einer Periode. Die Autoren verwenden diese Verteilung, um die Höhe des Sicherheitsbestandes zu bestimmen, der zu dem geforderten Servicegrad gehört. Die implizite Rückkopplung des Modells mit den Lagerbeständen wird nicht berücksichtigt. Wenn nach Anhebung der Sicherheitsbestände die gleiche Planungssituation noch einmal simuliert wird, werden sich andere Produktionspläne ergeben, da jetzt kein oder nur ein geringerer Backlog auftritt.

Die Höhe der Sicherheitsbestände ist korrekt, wenn davon ausgegangen wird, daß der Produktionsplan beibehalten wird. Diese Annahme ist jedoch unwahrscheinlich, da der Plan unter der Annahme der entstehenden Backlog berechnet wurde und somit auch Strafkosten auslöst. Diese treten jedoch mit den Sicherheitsbeständen nicht mehr auf, so daß die Entscheidungssituation eine vollkommen andere ist. Ein weiterer Kritikpunkt ist die zeitpunktgeballte Nachfrage am Ende der Periode. Fehlmengen können so auch nur an den Periodengrenzen auftreten, in der Praxis sind diese jedoch auch innerhalb einer Periode auf. Trotz dieser Schwachpunkte ist dieser Ansatz ein erster Versuch, komplexe Lagerhaltungssysteme unter realistischeren Bedingungen zu simulieren und gleichzeitig die Höhe der Sicherheitsbestände zu bestimmen.

Ein neuer Aspekt bei der Betrachtung des Servicegrades wird von Thomas (2005) vorgestellt. Thomas betrachtet einen endlichen Planungshorizont und den hierauf entstehenden β-Servicegrad. Im Unterschied zu den analytischen Modellen, die von einem unendlichen Horizont ausgehen, stellt sich hier kein Gleichgewicht ein. Der Servicegrad ist somit keine Konstante, sondern eine Verteilung. Für die Praxis, die auf einem beschränkten Horizont arbeitet, hat dies zur Konsequenz, daß die Lagerbestände tendenziell höher angesetzt werden müssen. So wird sichergestellt, daß der gewünschte Servicegrad mit einer vorgegebenen Wahrscheinlichkeit auch erreicht wird. Je nach Länge des

Horizonts sinkt dieser Wert, liegt aber immer noch deutlich über dem Gleichgewichtswert.

6.1.6 Ausblick

Die in den vorangehenden Abschnitten vorgestellten Arbeiten stellen nur einen kleinen Ausschnitt aus der Vielzahl an Modellen und Ideen im Spannungsfeld zwischen Produktionsplanung und Unsicherheit dar. Die Menge der eingesetzten Methoden reicht von analytischen Modellen über Modifikationen von klassischen Modellen hin zu heuristischen oder metaheuristischen Ansätzen. In einigen Bereichen sind Ansätze für weitere Entwicklungen erkennbar - die Annäherung von analytischen Modellen und rollierender Planung/MRP-Systemen sowie deren Überprüfung durch Simulationsstudien. Damit verwandt ist ein weiterer Trend: die Simulation komplexer Systeme und der Einsatz von Suchverfahren, um die Höhe und Position der Sicherheitsbestände zu bestimmen. Alle hier aufgeführten Arbeiten berücksichtigen jedoch nicht alle Planungsebenen, so daß es zu Abbildungsfehlern kommt (z. B. nur Fehlmengen an Periodengrenzen). Die Ursachen hierfür liegen in der eingesetzen Modellierung der Planungssysteme.

Eine wesentliche gemeinsame Eigenschaft aller Planungsmodelle ist die Einteilung in Perioden. Im Gegensatz zu stochastischen Modellen der Lagerhaltung findet die Nachfrage und Produktion nur an den Periodengrenzen statt. Der gesamte Zu-/Abgang von Material erfolgt zeitpunktgeballt an den Periodengrenzen. Auch Modelle, die das periodenübergreifende Rüsten erlauben, können die Nachfrage nur an den Periodengrenzen erfüllen. Diese Annahmen sind sinnvolle Restriktionen, da so die Anzahl an Zeitpunkten mit Veränderungen der Lagerbestände gering gehalten wird und die Planung mit Hilfe von gemischt-ganzzahligen Modellen erst ermöglicht. Diese Eigenschaft wird bei der Simulation weiter aufrecht erhalten. Die Nachfrage ist zwar in ihrer Höhe stochastisch, jedoch ist der Zeitpunkt deterministisch. Die Zeitpunkte werden auch in den Planungsmodellen übernommen, die ein- oder zweistufig sind und einen Jahresplan und rollierenden Monatsplan nachbilden. Modelle der stochastischen Lagerhaltung kennen diese Probleme meist

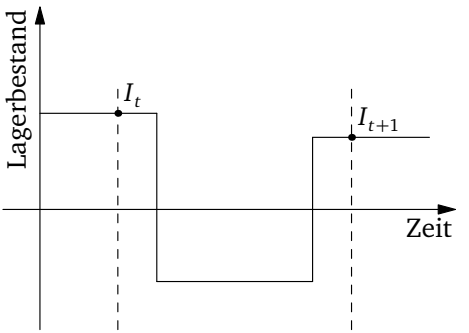


Abbildung 6.3: Modellierungsfehler der periodischen Nachfrage

nicht, da die Nachfrage sowohl in der Höhe, als auch im Zeitpunkt eine stochastische Größe ist und auf einer kontinuierlichen Zeitachse gearbeitet wird. Stochastische Modelle, die auf einem Periodenraster basieren, stehen jedoch den gleichen Problemen gegenüber.

Eine Berücksichtigung der Nachfrage außerhalb der Periodengrenzen ist erst möglich, wenn auch die Zeitpunkte der Produktion aller Lose innerhalb einer Periode bekannt sind. Die Voraussetzung ist somit die Modellierung einer Feinplanung und damit eines vollständigen dreistufigen hierarchischen Planungssystems. Der stark erhöhte Rechenaufwand ist einer der Gründe, daß solche Systeme für die in den vorangehenden Abschnitten vorgestellten Fragestellungen noch nicht eingesetzt wurden. Ein weiterer Grund sind die Schwierigkeiten, eine offene Produktion in einer Simulation mit einem geringen Rechenaufwand wiederzugeben. Die Grundlagen hierfür und die Implementierung wurden im Rahmen dieser Arbeit bereits vorgestellt (vgl. Kapitel 5).

Die Notwendigkeit zur Simulation einer dritten Planungsebene, sowie einer kontinuierlichen Produktion zeigt Abbildung 6.3. Die Periodengrenzen liegen bei t und $t+1$. In einem rein periodischen System würden nur die Lagerbestände I_t und I_{t+1} in die Berechnungen einbezogen werden. Da die Produktion und Nachfrage nur an den Periodengrenzen auftritt, würde in diesem Fall keine Fehlmenge auftreten und der β-Servicegrad läge bei 100%. In der Praxis kann es jedoch so sein,

daß die Nachfrage am Beginn der Periode auftritt und die Lieferung erst am Ende, so daß eine Fehlmenge entsteht und der β-Servicegrad sinkt. Da in der Praxis die Messung der Kennzahlen auch auf kontinuierlicher Basis und nicht stichtagsbezogen erfolgt, liegt die kontinuierliche Sichtweise näher an der Realität. Die periodische Sichtweise erzeugt Modellierungsfehler, die direkt von der Länge der Perioden abhängen und mit steigender Länge zunehmen. Um diese zu vermeiden und eine bessere Vergleichbarkeit mit analytischen Modellen der Lagerhaltung herzustellen ist die kontinuierliche Sichtweise vorzuziehen.

Folgende Aspekte sind nach einem Wechsel von der periodischen auf die kontinuierliche Sichtweise zu beachten:

- Wenn die Nachfrage zwischen den Periodengrenzen auftritt, muß auch die Produktion dazwischen durchgeführt werden, um einen Abbildungsfehler zu vermeiden.

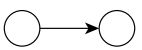
- Zur Abbildung der Produktion zwischen den Periodengrenzen ist eine weitere Planungsebene nötig. Die Kosten, die auf dieser Ebene entstehen, müssen mit denen der darüberliegenden Ebenen abgestimmt werden.

- Bei der Berechnung von Kennziffern muß darauf geachtet werden, daß die Erhebung der Daten entsprechend angepaßt wird, um Meßfehler zu vermeiden.

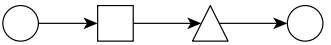
Alle drei Aspekte wurden in den vorangehenden Arbeiten nicht ausreichend beachtet. Mit Hilfe des in Kapitel 5 beschriebenen Simulationssystems ist eine exakte Abbildung möglich. Die dazugehörigen Planungsalgorithmen werden in den folgenden Abschnitten vorgestellt.

6.2 Untersuchungsgegenstand

Aufgrund der Bedeutung der APS Systeme für eine optimierende Unternehmensplanung soll ein Ausschnitt aus einem APS simuliert werden. Die Untersuchung beschränkt sich hierbei auf die Module zur Planung



(a) Erzeugnisstruktur



(b) Anordnung der Maschinen

Abbildung 6.4: Erzeugnisstruktur und Maschinenanordnung der Versuchsumgebung

der Produktion und Beschaffung. Die Planungsabläufe werden im folgenden Abschnitt 6.3 auf Seite 208 vorgestellt. Sie orientieren sich hierbei an den in der Praxis eingesetzten Verfahren.

Die Erzeugnisstruktur wurde möglichst einfach gehalten, um die Rechenzeit zu verringern. Die Abbildungen 6.4a bis 6.4b auf dieser Seite zeigen die Erzeugnisstruktur sowie die entsprechende Anordnung der Maschinen. Die Fertigung besteht aus einer Produktionsstufe, die ihre Komponenten aus einer Quelle bezieht. Diese Quelle ist eine ideale Quelle, d. h. sie kann jedes Produkt in jeder beliebigen Menge zu jedem Zeitpunkt bereitstellen. Die von der Ressource erzeugten Produkte werden in ein Lager überführt, welches keine Kapazitätsbeschränkung besitzt. Dieses Lager verwaltet außer den physisch vorhandenen Produkten auch Nachlieferungen, wenn Aufträge eintreffen, die aktuell nicht erfüllt werden können. Der letzte Knoten ist eine Senke, die eine Quelle für Kundenaufträge ist und gleichzeitig eine Senke für die Produkte, die hier aus der Simulation entfernt werden.

Der Einsatz einer hierarchischen Planung für eine einstufige Fertigung, mit einer geringen Anzahl an Produkten und einer Nachfrage, die keine saisonalen Muster aufweist, ist nicht notwendig, da auch einfachere Planungsverfahren zum Einsatz kommen könnten. Der Einsatz einer Jahresplanung kommt in der Praxis zum Einsatz, um z. B. den Auf-/Abbau von Saisonlagerbeständen zu steuern oder wechselnde Anlagekapazitäten aufgrund von Wartungsarbeiten. Diese Effekte werden innerhalb der Untersuchung nicht berücksichtigt, da sie einen weiteren Einflußfaktor darstellen.

Die nachfolgend beschriebenen Versuche wurden mit einem hierar-

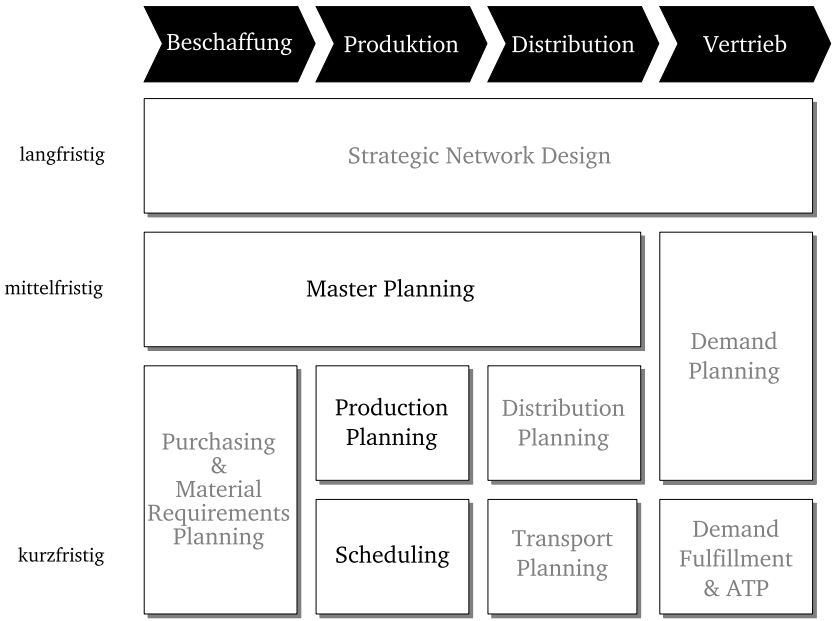


Abbildung 6.5: Eingesetzte Planungsmodule innerhalb der Versuchsumgebung

chischen Produktionsplanungssystem durchgeführt. Die Entscheidungs- und Informationsstruktur entspricht im Wesentlichen einem APS, wie es in Abbildung 6.2 auf Seite 193 vorgestellt wurde. Hieraus wurde ein reduziertes System entwickelt, dessen Funktionalität sich auf den Bereich der Produktionsplanung beschränkt. Das verwendete Planungssystem ist in der Abbildung auf dieser Seite dargestellt.

Der Bereich der strategischen Planung wird vollständig ausgeblendet. Das Ziel dieser Arbeit ist, wie bereits zu Beginn des Kapitels dargelegt, die Untersuchung der Interaktion zwischen Parametern der mittelfristigen Planung und den Sicherheitsbeständen. Entscheidungen, die auf strategischer Ebene fallen, werden als gegeben vorausgesetzt und innerhalb der betrachteten Zeitskala als konstant vorausgesetzt. Das Modul Strategic Network Design, welches den langfristigen Auf-

/Abbau von Personal/Ressourcen/Standorten betrachtet, kann somit vernachlässigt werden. Als Ergebnis der Planung auf mittelfristiger Ebene wird ein Plan erstellt, der für einen Zeitraum von einem Jahr auf Monatsbasis die Produktionsmengen, Anzahl an Losen, sowie den Lagerbestand am Ende einer Periode festlegt (Master Planning). Dieser Plan wird während einer Simulation nicht geändert und muß daher nur einmal zu Beginn berechnet werden. Das hierfür verwendete Modell wird in Abschnitt 6.3.2 vorgestellt.

Die Struktur der Nachfrage wird innerhalb des Jahres als konstant angenommen, so daß das Modul für die mittel-/kurzfristige Vorhersage (Demand Planning) nicht weiter betrachtet wird. Innerhalb der Planung treten zwar Schwankungen in der Kundennachfrage auf, es werden jedoch für die Berechnungen der mittelfristigen Pläne die Planzahlen aus dem vorher berechneten Jahresplan übernommen. Zukünftige Untersuchungen können dieses Modul in die Planung mit einbeziehen, wenn auch Effekte der Vorhersage untersucht werden sollen.

Die zentralen Module der mittel-/kurzfristigen Ebene sind das Production Planning und Scheduling. Das Production Planning Modul erstellt Pläne für einen Zeitraum von drei Monaten mit Hilfe eines gemischt-ganzzahligen Modells, welches in Abschnitt 6.3.3 auf Seite 223 vorgestellt wird. Die Losgrößen werden vom Scheduling Modul übernommen, um die Reihenfolge der Lose auf den Maschinen festzulegen. Dies erfolgt durch den Einsatz einer Heuristik zur Tourenplanung, die in Abschnitt 6.3.5 auf Seite 227 detailliert vorgestellt wird. Die Verbindung und der Datenaustausch der Ebenen untereinander erfolgen über eine Rollierung, deren Funktionsweise in Abschnitt 6.3.1 erläutert wird. Die Anbindung nach außen realisiert ein Kundennachfrage-Modul (vgl. Abschnitt 6.3.6), welches eine stochastische Nachfrage erzeugt.

6.3 Planungsmodule

6.3.1 Planaktualisierung - Rollierung

Nachdem ein Plan auf einer Ebene erstellt worden ist, ändern sich meist schon kurze Zeit später die Bedingungen, unter denen dieser Plan erstellt worden ist. Es gibt zwei Alternativen diese Änderungen zu antizipieren - ereignisorientierte Neuplanung oder regelmäßige Neuplanung. Eine ereignisorientierte Neuplanung wird durchgeführt, wenn aufgrund der eintreffenden geplanten und ungeplanten Ereignisse Grenzen überschritten werden, die als Indikator für eine Neuplanung gesetzt wurden.

Eine regelmäßige Neuplanung wird unabhängig von den eintreffenden Ereignissen ausgeführt, wenn der nächste geplante Zeitpunkt erreicht ist. Die Abstände zwischen diesen Zeitpunkten sind meist identisch, vergleichbar mit einer Frequenz (Hopp u. Spearman 2001, S. 123). Die Frequenz, mit der eine Aktualisierung durchgeführt wird, ist entscheidend für die Leistung des Planungssystems. Bei einer hohen Frequenz besteht die Gefahr, daß durch die neuen Ereignisse zu schnell ein vollständig neuer Plan erstellt wird. Eine zu geringe Frequenz hingegen kann eine zu langsame Anpassung bedeuten. Innerhalb dieser Untersuchung wird nur das letzte Verfahren betrachtet, da es in der Praxis weiter verbreitet ist und durch seine Regelmäßigkeit auch leicht zu realisieren ist.

Das Master Planning erstellt einen Produktionsplan, der für die gesamte Simulationsdauer gültig ist. Dieser Plan hat eine Länge von einem Jahr und eine Periodeneinteilung von 12 Monaten (vgl. Abschnitt 6.3.2). Das Production Planning Modul übernimmt die Informationen der Lagerbestände (am Ende eines Monats) aus diesem Plan und erstellt zu Beginn jedes Monats einen Plan mit einer Länge von drei Monaten und einer Periodenlänge von einem Monat (vgl. Abbildung 6.6 auf der gegenüberliegenden Seite). Aus dem 3-Monatsplan wird nur der erste Monat fixiert und an die Feinplanung (Scheduling) übergeben. Die Berechnungen für die anderen beiden Monate werden durchgeführt,

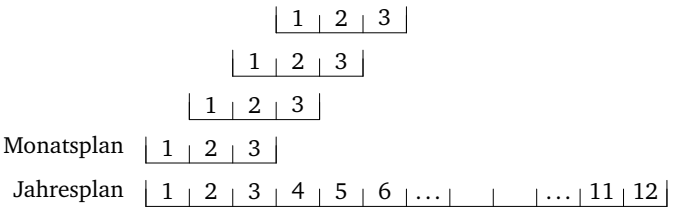


Abbildung 6.6: Rollierung des Monatsplans

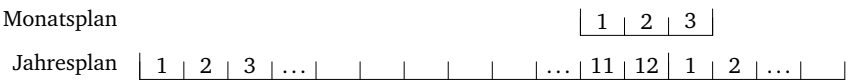


Abbildung 6.7: Rollierung am Ende des Jahresplans

um Effekte zu vermeiden, die am Ende des Planungshorizonts auftreten können (Stadtler 2000, S. 318).

Die Feinplanung übernimmt die Anzahl und Größe der Lose und führt eine Reihenfolgeplanung auf den Ressourcen durch. Außer dem Lagerbestand werden auch die im System bekannten Kundenaufträge und aktuell laufende Produktion berücksichtigt. Die Feinplanung wird zu Beginn jeden Tages mit den jeweils aktuellen Daten durchgeführt. Der Produktionsplan für den kommenden Tag wird daraufhin an einen Fertigungsleitstand übergeben, der den Plan umsetzt, indem er die Aufträge zum geplanten Zeitpunkt an die Ressourcen übergibt.

Ist die Rollierung so weit fortgeschritten, daß sie am Ende des Planungshorizonts von einem Jahr angekommen ist (vgl. Abbildung 6.7), so werden die Daten vom Beginn des Planungshorizonts zyklisch fortgeschrieben. Der Versuchsaufbau soll die Effekte auf einem Zeitraum von einem Jahr untersuchen, so daß angenommen wird, daß die Werte zu Beginn des folgenden Jahres im Wesentlichen mit denen vom Beginn des aktuellen Jahres übereinstimmen. Mit diesem Ansatz lassen sich keine einmaligen Effekte, wie sie z. B. bei der Einführung eines neuen Produktes entstehen, berücksichtigen. Die gleiche Funktionalität wird auch eingesetzt, um die Daten aus der Datenbank auf den Pla-

nungshorizont für den Jahresplan abzubilden (siehe auch Abschnitt 7.2.2).

6.3.2 Jahresplanung - Master Planning

Die Jahresplanung erstellt einen Plan, der für den Zeitraum von einem Jahr für jeden Monat die Produktionsmenge und die Anzahl der zu produzierenden Losen angeben soll. Dieser Plan soll unter Berücksichtigung der in diesem Zeitraum verfügbaren Kapazität, sowie der Lagerkosten und der für das Rüsten zu verbrauchenden Kapazität berechnet werden. In dem in Abbildung 6.5 vorgestellten APS-System entspricht dies dem Master Planning.

Erzeugnisstruktur und Anordnung der Maschinen entsprechen den in Abbildung 6.4 auf Seite 205 gezeigten Strukturen. Die Fertigung wird auf eine Stufe, die jedoch mehrere parallele Maschinen enthalten kann, beschränkt. Die Erzeugnisstruktur ist linear, es werden jedoch mehrere Produkte betrachtet, die sich die Ressourcen teilen. In die Planungen einbezogen wird ein von außen vorgegebener Sicherheitsbestand. Unterschreitungen des Sicherheitsbestandes sind zulässig, werden jedoch mit Strafkosten belegt. Ein wichtiger Aspekt ist die Möglichkeit, daß die Nachfrage auch verspätet erfüllt werden kann. Eine Einschränkung der Lieferfrist ist nicht vorgesehen. Dieser Fall wird in der Literatur auch als Backlogging bezeichnet (vgl. Zangwill 1966, S. 105 und Pochet u. Wolsey 2006, S. 303ff). Daher stehen folgende Eingabedaten zur Verfügung:

- deterministische Nachfrage für jedes Produkt und jede Periode
- verfügbare Kapazität je Ressource und Periode
- Kapazitätsbedarf, um auf einer Ressource einen Rüstvorgang für ein Produkt durchzuführen.
- Höhe des Sicherheitsbestandes je Produkt und Periode
- Höhe der Lagerkostensätze je Produkt und Periode

- Lageranfangsbestand je Produkt - ein Anfangsbestand an Nachlieferungen ist nicht vorgesehen
- Ziellagerbestand am Ende des Planungshorizonts je Produkt - dieser muß mindestens so hoch sein wie der Sicherheitsbestand $i_j^T \geq SS_{j,T} \quad \forall j \in 1 \ldots J$
- Fertigungsgeschwindigkeit je Produkt und Ressource
- Erzeugnisstruktur
- Strafkostensätze für Unterschreitungen des Sicherheitsbestandes und des Lagerendbestandes. Die Strafkostensätze stehen in folgender Relation zueinander $c_j^{BL} > c_j^{SS} > c_j^{IT} > h \quad \forall j \in 1 \ldots J$

Die Anlage ist für die kontinuierliche Produktion geeignet und muß keine festen Losgrößen produzieren. Einen aktuellen Überblick über Modelle zur Batchproduktion, die diese Eigenschaft berücksichtigen, findet sich in Méndez u. a. (2006, S. 913).

Aufgrund der Struktur des Problems, sowie der Planungsebene wird ein Modell vom Typ CLSP verwendet, wie es von Billington u. a. (1983, S. 1130ff) vorgestellt wurde. Die Formulierung orientiert sich an den Arbeiten von Trigeiro u. a. (1989, S. 354) sowie Miller u. Wolsey (2003, S. 557). Das von Miller u. Wolsey aufgestellte allgemeine Modell enthält bereits alle Erweiterungen, die zur Abbildung von Nachlieferungen und Lageranfangsbeständen notwendig sind. Die Modellierung der Nachlieferungen erfolgt analog der von Zangwill (1969, S. 510) vorgestellten Erweiterung des klassischen Wagner-Whitin Modells um Backlog, durch Anpassung der Kontinuitätsbedingung.

Mathematische Formulierung

$$\min \sum_j \sum_t h_j \cdot I_{j,t} + \sum_m \sum_{j \in w_j^M} \sum_t sc_{m,j} \cdot Y_{m,j,t} + \sum_j \sum_t c_j^{BL} \cdot BL_{j,t} + \sum_j \sum_t c_j^{SS} \cdot SS_{j,t}^- + \sum_j c_j^{IT} \cdot I_j^- \tag{6.1}$$

$$I_{j,t-1} - BL_{j,t-1} + \sum_{m \in w_j^M} X_{m,j,t} = d_{j,t} + I_{j,t} - BL_{j,t} \quad \forall \begin{matrix} j = 1 \dots J, \\ t = 1 \dots T \end{matrix} \tag{6.2}$$

$$I_{j,t} + SS_{j,t}^{-} \geq ss_{j,t} \quad \forall \begin{matrix} j = 1 \dots J, \\ t = 1 \dots T \end{matrix} \tag{6.3}$$

$$I_{j,T} + I_j^{-} + SS_{j,T}^{-} \geq i_j^T \quad \forall j = 1 \dots J \tag{6.4}$$

$$i_j^T - ss_{j,T} \geq I_j^{-} \quad \forall j = 1 \dots J \tag{6.5}$$

$$\sum_{j \in w_m^J} \frac{X_{m,j,t}}{\kappa_{m,j}} + \sum_{j \in w_m^J} st_{m,j} \cdot Y_{m,j,t} \leq c_{m,t} \quad \forall \begin{matrix} m = 1 \dots M, \\ t = 1 \dots T \end{matrix} \tag{6.6}$$

$$\begin{aligned} X_{m,j,t} \leq & Y_{m,j,t} \cdot \min \Big(\\ & \max\big(0, \kappa_{m,j} \cdot (c_{m,t} - st_{m,j})\big), \\ & \max\big(0, \sum_{\tau=1}^{T} d_{j,\tau} + \max(\max_{\tau=t \dots T}(SS_{j,\tau}), i_j^T) \\ & - I_{j,0} + BL_{j,0}\big)\Big) \end{aligned} \quad \forall \begin{matrix} m = 1 \dots M, \\ j \in w_m^J, \\ t = 1 \dots T \end{matrix} \tag{6.7}$$

$$X_{m,j,t},\, I_{j,t},\, BL_{j,t},\, SS_{j,t}^{-},\, I_j^{-} \geq 0 \tag{6.8}$$

$$Y_{m,j,t} \in \{0, 1\} \tag{6.9}$$

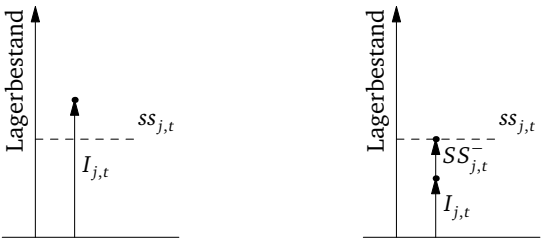
Das Ziel des Modells ist die Minimierung der Lager- und Rüstkosten, sowie der Strafkosten für die Unterschreitung der Sicherheits- und Lagerendbeständen, sowie für Nachlieferungen (6.1). Gleichung (6.2) beschreibt die Lagerkontinuitätsbedingung ergänzt um die Möglichkeit an Nachlieferungen. Die Aufteilung des physischen Lagerbestandes in einen Sicherheitsbestand und normalen Lagerbestand wird durch Gleichung (6.3) beschrieben. Die Funktionsweise zeigen die Abbildungen 6.8 auf Seite 215. Wenn der Lagerbestand $I_{j,t}$ oberhalb der Grenze

M	Anzahl an Maschinen
J	Anzahl an Produkten
T	Anzahl an Perioden
$I_{j,t}$	Lagerbestand für Produkt j am Ende von Periode t
$BL_{j,t}$	Bestand an Nachlieferungen (Backlog) für Produkt j am Ende von Periode t
$X_{m,j,t}$	Produktionsmenge von Produkt j auf Maschine m in Periode t
$Y_{m,j,t}$	binäre Indikatorvariable für einen Rüstvorgang auf Maschine m für Produkt j in Periode t
I_j^-	Höhe der Unterschreitung des Zielbestands von Produkt j am Ende des Planungshorizonts
$SS_{j,t}^-$	Höhe der Unterschreitung des Sicherheitsbestands von Produkt j in Periode t

Tabelle 6.2: Übersicht der Variablen des Modells zur mittelfristigen Planung

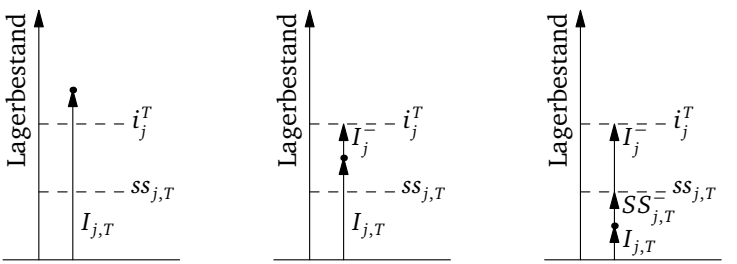
$d_{j,t}$	Nachfrage nach Produkt j in Periode t
i_j^T	Ziel-Lagerbestand von Produkt j am Ende des Planungshorizonts
$ss_{j,t}$	Sicherheitsbestand von Produkt j in Periode t
w_j^M	Menge der Maschinen, auf denen Produkt j produziert werden kann
w_m^J	Menge der Produkte, die auf Maschine m produziert werden können
$\kappa_{m,j}$	Fertigungsgeschwindigkeit von Produkt j auf Maschine m
$c_{m,t}$	Verfügbare Kapazität von Maschine m in Periode t
$st_{m,j}$	Rüstzeit von Maschine m für Produkt j
$I_{j,0}$	Lageranfangsbestand von Produkt j
h_j	Lagerkostensatz für eine Periode und Mengeneinheit von Produkt j
$sc_{m,j}$	Kosten für einen Rüstvorgang von Produkt j auf Maschine m
c_j^{BL}	Strafkosten für eine Mengeneinheit Backlog in einer Periode von Produkt j
c_j^{SS}	Strafkosten pro Periode für jede Mengeneinheit um die der Sicherheitsbestand von Produkt j unterschritten wird
c_j^{IT}	Strafkosten für jede Mengeneinheit um die der Ziellagerbestand von Produkt j unterschritten wird

Tabelle 6.3: Übersicht der Eingabedaten des Modells zur mittelfristigen Planung



(a) $I_{j,t} \geq ss_{j,t}$ (b) $I_{j,t} < ss_{j,t}$

Abbildung 6.8: Funktionsweise der Strafkosten für eine Unterschreitung des Sicherheitsbestandes



(a) $I_{j,T} \geq i_j^T$ (b) $i_j^T \geq I_{j,T} \geq ss_{j,T}$ (c) $ss_{j,T} \geq I_{j,T}$

Abbildung 6.9: Funktionsweise der Strafkosten für eine Unterschreitung des Lagerendbestandes

von $ss_{j,t}$ liegt, so nimmt $SS^-_{j,t}$ den Wert 0 an, da es für das Modell günstiger ist, einen Lagerbestand vorzuhalten, als den Sicherheitsbestand zu unterschreiten. Sobald dieser unterschritten wird (vgl. Abbildung 6.8b) nimmt $SS^-_{j,t}$ den kleinsten Wert an, so daß Ungleichung (6.3) erfüllt ist. Durch die Kostenstruktur in Verbindung mit Gleichung (6.3) gilt automatisch $ss_{j,t} \geq SS^-_{j,t} \geq 0 \quad \forall j,t$.

Die Unterschreitung des Lagerendbestandes wird durch Gleichung (6.4) und (6.5) ausgedrückt. Die Abbildungen 6.9 zeigen die unterschiedlichen Zustände, die der Lagerbestand am Ende des Planungsho-

rizonts annehmen kann. Solange sich der Lagerbestand $I_{j,T}$ oberhalb von i_j^T befindet, nehmen bei Strafkostenterme den Wert 0 an. Sinkt der Lagerbestand unter den Ziellagerbestand i_j^T, so werden Strafkosten für die Unterschreitung I_j^T berechnet. Sinkt der Lagerbestand unter den Sicherheitsbestand ss_j^T, so werden beide Strafkostenterme aktiviert. Die Höhe der Strafkosten wird jedoch getrennt ermittelt. Eine der Voraussetzungen dieses Modells ist $i_j^T \geq ss_{j,T}$, so daß immer zuerst die Strafkosten für die Unterschreitung des Lagerendbestandes anfallen und danach für die Unterschreitung des Sicherheitsbestandes. Um zu vermeiden, daß die Unterschreitungen doppelt berechnet werden, wird die Höhe der Unterschreitung des Lagerendbestandes in Gleichung (6.5) auf die Differenz zwischen beiden Grenzen festgelegt.

Alle Maschinen besitzen eine Kapazitätsbeschränkung, die durch Gleichung (6.6) modelliert wird. Die maximale Produktionsmenge auf einer Maschine innerhalb einer Periode wird durch die Kapazität abzüglich der Rüstkapazität begrenzt. Da keine Rüstzustandsübertragung im Modell vorgesehen ist, wird für jeden Rüstvorgang die Rüstzeit berechnet. Die Rüstzeit wird als Pauschale ohne Berücksichtigung der Reihenfolge veranschlagt. Von der Wahl der mittleren Rüstzeit hängt die Lösbarkeit der später durchzuführenden Feinplanung ab. Details zur Berechnung der Rüstzeit und der damit verbundenen strukturellen Fehler in der Testdatenerzeugung werden in Abschnitt 7.2.11 auf Seite 270 behandelt.

Die letzte Nebenbedingung (6.7) beschreibt die Aktivierung der Rüstindikatoren. Wenn die Produktionsmenge die Bedingung $X_{m,j,t} > 0$ erfüllt, wird ein Rüstvorgang ausgelöst. Die Produktionsmenge je Periode wird begrenzt von der zur verfügbaren Kapazität, dem Lageranfangsbestand sowie der Nachfrage und dem Lagerendbestand. Der erste Teil $\max(0, \kappa_{m,j} \cdot (c_{m,t} - st_{m,j}))$ beschreibt die maximale Produktionsmenge, wenn ausreichend Kapazität vorhanden ist. Falls die Kapazität $c_{m,t}$ so gering ist, daß kein Rüstvorgang ausgelöst werden kann, wird die rechte Seite negativ und der gesamte Term durch die max-Funktion auf 0 gesetzt. Der zweite Teil der Nebenbedingung beschreibt die Abhängigkeit von der Nachfrage, dem Lageranfangsbestand und den Nachlieferungen.

Falls zu Beginn ein Lagerbestand $I_{j,0}$ vorhanden ist, so muß dieser nicht produziert werden. Umgekehrt steigt die Produktionsmenge $X_{m,j,t}$, falls Nachlieferungen $BL_{j,0}$ zu Beginn vorliegen. Da Nachlieferungen ohne zeitliche Beschränkung erlaubt sind, ist es zu jedem Zeitpunkt möglich, daß die gesamte Nachfrage der vorangehenden und nachfolgenden Perioden ($\sum_{\tau=1}^{T} d_{j,\tau}$) hergestellt wird.

Da eine Unterschreitung des Sicherheitsbestandes oder des Ziellagerbestandes Strafkosten auslöst, müssen diese bei der Berechnung der maximalen Produktionsmenge ebenfalls berücksichtigt werden. Für das Modell ist es günstiger, einen Lagerbestand aufzubauen, als einen Sicherheitsbestand zu unterschreiten. Der maximal mögliche Sicherheitsbestand $\max_{\tau=t\ldots T}(SS_{j,t})$ stellt eine obere Schranke dar. Falls der Lagerendbestand i_j^T noch nicht produziert wurde, muß er in der oberen Schranke auch enthalten sein. Die Auswahl des maximalen Sicherheitsbestandes ist eine grobe obere Schranke. Eine niedrigere obere Schranke läßt sich durch eine umfangreichere Vorberechnung erhalten. Hierzu wird die Periode mit dem maximalen Sicherheitsbestand betrachtet. Am Ende dieser Periode soll der zusätzliche Lagerbestand gleich 0 sein. Für alle weiteren Perioden kann durch eine Berechnung der Nachfrage unter Berücksichtigung der Sicherheitsbestände und dem Lagerendbestand eine geringere obere Grenze ermittelt werden. Die so erhaltene obere Schranke ist niedriger und kann eingesetzt werden, wenn zum Ende des Planungshorizonts der Sicherheitsbestand sinkt. Bei gleichbleibenden Sicherheitsbeständen besitzt diese Formulierung keinen Vorteil.

Das Modell erzeugt einen Produktionsplan, der Informationen enthält, welche Produktionsmenge $X_{m,j,t}$ von Produkt j in Periode t gefertigt wird. Die Zuordnung der Maschine m, auf der die Fertigung durchgeführt wird, kann ebenfalls berücksichtigt werden. Für den Jahres- und Monatsplan ist diese Information jedoch noch nicht relevant, da sich durch Planänderungen Verschiebungen ergeben können. Erst wenn der Lösungsraum durch die individuellen Eigenschaften der Maschinen - die Zuordnung von Produkten zu Maschinen - stark eingeschränkt wird, muß eine Abbildung erfolgen.

Schnittebenen

Das im vorangehenden Abschnitt vorgestellte Standardmodell kann zur Beschleunigung des Lösungsprozesses noch um Schnittebenen ergänzt werden. Schnittebenen sind (problemspezifische) Ungleichungen, die den Lösungsraum der LP-Relaxation verringern und so den Branch & Bound Prozeß zum Teil stark verkürzen können. Die meisten der heute eingesetzten Programme zur gemischt-ganzzahligen Optimierung erzeugen bereits problemunspezifische Schnittebenen, wie Gomory-Cuts (vgl. Gomory 1958, S. 275 oder Marchand u. a. 2002, S. 401). Diese allgemeinen Schnittebenen können durch problemspezifische Schnittebenen ergänzt werden, die je nach Stärke den Lösungsprozess beschleunigen. Alternativ lassen sich deutliche Verbesserungen auch durch eine Reformulierung erreichen. Für Losgrößenmodelle gibt es zwei Arten der Reformulierung - die Darstellung als kürzeste-Wege Problem (Pochet u. Wolsey 2006, S. 222), sowie als Standortplanungsproblem (Pochet u. Wolsey 2006, S. 221). Beide Reformulierungen sollen nicht weiter betrachtet werden, da das hier verwendete Grundmodell durch seine Erweiterungen unmittelbar auf der Standardformulierung aufsetzt. Durch Tests hat sich gezeigt, daß eine Reformulierung als Standortplanungsproblem zwar eine geringere Ganzzahligkeitslücke besitzt, durch die größere Anzahl an Variablen und Nebenbedingungen erhöhen sich die Rechenzeit und der Speicherbedarf. Insgesamt benötigt diese Variante mehr Zeit als die Standardformulierung mit entsprechenden Schnittebenen, um eine Lösung mit einer vergleichbaren Qualität zu erreichen.

Untersuchungen im Bereich der Losgrößenmodelle haben gezeigt, daß es Klassen von Schnittebenen gibt, die sich ausgehend vom Grundmodell auch auf andere Fälle anwenden lassen. Die erste Gruppe dieser Ungleichungen sind die (l,S)-Schnittebenen (Pochet u. Wolsey 2006, S. 218), die eine Verbindung zwischen der Produktionsmenge, den Rüstvariablen und dem Lagerbestand herstellen. Für den unkapazitierten Ein-Produkt, Ein-Maschinen Fall beschreiben diese Ungleichungen bereits vollständig die konvexe Hülle (Pochet u. Wolsey 2006, S. 218). Daraus abgeleitete Schnittebenen für den Mehr-Produkt-Fall werden

von Pochet u. Wolsey (2006, S. 384ff) vorgestellt. Belveaux u. Wolsey (2001, S. 995) stellen auch Erweiterungen vor, die den Rüstzustand über Periodengrenzen hinweg übertragen, oder Nachlieferungen berücksichtigen. Diese Erweiterung wird auch im Folgenden auf das in Abschnitt 6.3.2 vorgestellte Modell angewendet. Eine zweite Gruppe von Schnittebenen basiert auf den Ideen von Gomory (Pochet u. Wolsey 2006, S. 276) und beschreibt die Anzahl an Rüstvorgängen anhand der zur Verfügung stehenden Kapazität. Auch diese Gruppe von Schnittebenen kann auf den Fall mit Nachlieferungen erweitert werden (Belveaux u. Wolsey 2001, S. 995).

Außer diesen allgemeinen Schnittebenen wurden für unterschiedliche Sonderfälle weitere Schnittebenen oder Reformulierungen entwickelt. Miller u. Wolsey (2003) entwickeln Reformulierungen für den Fall, daß Nachlieferungen auftreten, Sicherheitsbestände berücksichtigt werden oder Lageranfangsbestände existieren. Wenn die Kapazitäten in jeder Periode identisch sind, wurden von Miller u. Wolsey (2003, S. 557) weitergehende Formulierungen entwickelt. Diese decken Varianten des Grundmodells mit und ohne Backlog sowie mit Lageranfangsbeständen ab. Auch Sicherheitsbestände in Verbindung mit stückweise linearen Kosten wurden eingeführt (Miller u. Wolsey 2003, S. 559). Schnittebenen bei Modellen mit Rüstzeiten werden von Marchand u. a. (2002, S. 436), Miller u. a. (2003, S. 73), Pochet u. Wolsey (2006, S. 376) und Absi u. Kedad-Sidhoum (2008) untersucht. Für den Fall, daß identische parallele Maschinen vorhanden sind, wird der Einsatz von zusätzlichen Nebenbedingungen empfohlen, um die Symmetrie zwischen den Maschinen aufzubrechen (Sherali u. Smith 2001). Muß die Nachfrage nicht vollständig erfüllt werden (Lostsales), so können die von Absi u. Kedad-Sidhoum (2008, S. 1351) entwickelten Schnittebenen und Algorithmen eingesetzt werden.

Für das Modell aus Abschnitt 6.3.2 werden folgende Schnittebenen eingesetzt, die aus Belveaux u. Wolsey (2001, S. 995) übernommen wurden.

$$Y_{m,j,t} \leq Z_{j,t} \quad \forall \begin{matrix} m = 1 \ldots M, \\ j = 1 \ldots J, \\ t = 1 \ldots T \end{matrix} \tag{6.10}$$

$$\sum_{m} Y_{m,j,t} \geq Z_{j,t} \quad \forall \begin{matrix} j = 1 \ldots J, \\ t = 1 \ldots T \end{matrix} \tag{6.11}$$

$$I_{j,t-1} + \sum_{\tau=k}^{t} BL_{j,\tau} \geq \sum_{\tau=k}^{t} d_{j,\tau} \cdot \left(1 - \sum_{\theta=k}^{\tau} Z_{j,\theta}\right) \quad \forall \begin{matrix} j = 1 \ldots J, \\ t = 1 \ldots T, \\ k = 1 \ldots t \end{matrix} \tag{6.12}$$

$$C_j^{max} = \max_t \left(\sum_m C_{m,t}\right) - \sum_m st_{m,j} \tag{6.13}$$

$$d_{j,k,t} = \sum_{\tau=k}^{t} d_{j,\tau} \tag{6.14}$$

$$\eta_{j,k,t} = \left\lceil \frac{d_{j,k,t}}{C_j^{max}} \right\rceil \tag{6.15}$$

$$\rho_{j,k,t} = d_{j,k,t} - C_j^{max} \cdot (\eta_{j,k,t} - 1) \tag{6.16}$$

$$I_{j,k-1} + BL_{j,t} \geq \rho_{j,k,t} \cdot \left(\eta_{j,k,t} - \sum_{\tau=k}^{t} Z_{j,k}\right) \quad \forall \begin{matrix} j = 1 \ldots J, \\ t = 1 \ldots T, \\ k = 1 \ldots t \end{matrix} \tag{6.17}$$

Als Hilfsvariable wird ein zusätzlicher Indikator $Z_{j,k,t}$ eingeführt, der einen Rüstvorgang für ein Produkt j in Periode t auf einer oder mehrerer Maschinen signalisiert. Die Aktivierung der Hilfsvariablen geschieht über die Kombination der Nebenbedingungen (6.10) und (6.11). Eine Einschränkung des Gültigkeitsbereichs ist nicht nötig, da

$Z_{j,t}$	Indikator für eine Rüstvorgang von Produkt j in Periode t
C_j^{max}	Maximal verfügbare Kapazität für Produkt j während des gesamten Planungshorizonts, aggregiert über alle Maschinen abzüglich der Rüstzeiten
$d_{j,k,t}$	Kumulierte Nachfrage für Produkt j von Periode k bis einschließlich Periode t
$\eta_{j,k,t}$	Untere Grenze für die Anzahl an Rüstvorgängen für Produkt j im Intervall $[k,t]$
$\rho_{j,k,t}$	Untergrenze der Nachfrage, die im Zeitraum $[k,t]$ nicht durch die verfügbare Kapazität abgedeckt werden kann

Tabelle 6.4: Übersicht der Variablen in den eingesetzten Schnittebenen

alle $Y_{m,j,t}$ Variablen bereits binär sind. Die erste Schnittebene (6.12) ist eine Modifikation der (l,U) Cuts von Belveaux u. Wolsey (2001, S. 995). Die Funktionsweise dieser Schnittebene erschließt sich, indem der Fall $k = t$ betrachtet wird.

$$I_{j,t-1} + BL_{j,t} \geq d_{j,t} \cdot \left(1 - Z_{j,t}\right)$$

Falls keine Produktion in t stattfindet ($Z_{j,t} = 0$), so muß die Nachfrage entweder durch den Lagerbestand erfüllt werden oder tritt als Backlog am Ende der Periode $BL_{j,t}$ auf. Wird hingegen produziert, so ist die linke Seite nach unten nur durch 0 begrenzt. Durch Ausweitung des zu überbrückenden Zeitraums $k = 1 \ldots t$ ergibt sich die vorgestellte Schnittebene.

Für die zweite Schnittebene (6.17) ist die Einführung von weiteren Hilfsgrößen notwendig. Die Größe C_j^{max} in Gleichung (6.13) beschreibt die maximale Kapazität, die für die Produktion von Produkt j in einer Periode während des gesamten Planungszeitraums zur Verfügung steht. Da für jeden Rüstvorgang eine Rüstzeit anfällt, wird der Term noch um die Rüstzeit von Produkt j korrigiert. Ein weiterer Term (6.14)

$d_{j,k,t}$ beschreibt die kumulierte Nachfrage von Produkt j im Intervall $[k,t]$. Der Quotient beider Größen $\lceil \eta_{j,k,t} \rceil$ aufgerundet auf die nächste ganze Zahl ist eine untere Grenze für die Anzahl an Perioden, in denen eine Produktion von j erfolgen wird. Gleichzeitig entspricht dies der Mindestanzahl an Rüstvorgängen.

$\eta_{j,k,t} - 1$ ist äquivalent zu $\left\lfloor \frac{d_{j,k,t}}{C_j^{max}} \right\rfloor$, dem nach unten abgerundeten Quotienten aus kumulierter Nachfrage und maximaler Periodenkapazität. Multipliziert mit der maximalen Periodenkapazität und von der kumulierten Nachfrage subtrahiert entsteht $\rho_{j,k,t}$. $\rho_{j,k,t}$ entspricht dem Teil der Nachfrage im Intervall $[k,t]$, der nicht mit der zur Verfügung stehenden Produktionskapazität hergestellt werden kann. Schnittebene (6.17) gibt somit vor, daß die Restnachfrage multipliziert mit der Differenz aus der Mindestanzahl an Rüstvorgängen abzüglich der tatsächlichen Anzahl an Rüstvorgängen entweder aus dem Lagerbestand in t erfüllt werden muß oder als Backlog am Ende von t entsteht.

Modellgröße

Das hier vorgestellte Modell entspricht bei Vernachlässigung der Strafkosten für die Sicherheitsbestände und dem Lagerendbestand dem von Miller u. Wolsey (2003, S. 562) vorgestellten Modell, falls nur eine Maschine betrachtet wird. Für diesen Modelltyp ist die Komplexität unbekannt, falls die Anzahl an Produkten $J > 1$ ist (Miller u. Wolsey 2003, S. 563). Es ist anzunehmen, daß durch den zusätzlichen Freiheitsgrad ein Produkt auf mehreren Maschinen zu fertigen, die Komplexität nicht abnimmt.

Die Abhängigkeit der Modellgröße von der Anzahl an Perioden T, Produkten J und Maschinen M ist in Tabelle 6.5 auf der gegenüberliegenden Seite aufgeführt. Hieraus läßt sich ablesen, daß im Grundmodell sowohl die Variablen, als auch die Nebenbedingungen linear mit der Anzahl an Perioden, Produkten und Maschinen zunehmen. Durch die Einführung von Schnittebenen wächst die Anzahl an Nebenbedingungen quadratisch mit der Zeit, während die zusätzlichen Hilfsvariablen linear mit der Zeit und der Anzahl an Produkten wachsen. Somit ist

Typ	Anzahl	Ordnung: Perioden	Ordnung: Produkte	Ordnung: Maschinen
Variablen	$J+3JT+MJT$	$O(n)$	$O(n)$	$O(n)$
davon binär	MJT	$O(n)$	$O(n)$	$O(n)$
Nebenbedingungen	$2JT+2J+MTJ+MT$	$O(n)$	$O(n)$	$O(n)$
Schnittebenen	$JT(M+T+2)$	$O(n^2)$	$O(n)$	$O(n)$
zusätzliche Variablen	JT	$O(n)$	$O(n)$	$O(1)$

Tabelle 6.5: Komplexitätsbetrachtung der Variablen, Nebenbedingungen und Schnittebenen

die Anzahl an Perioden eine entscheidende Größe im Bezug auf die Modellgröße.

6.3.3 Monatsplanung - Production Planning

Auf dieser Ebene wird ein Produktionsplan für die kommenden drei Monate erstellt mit einer Auflösung von einem Monat. Der Plan wird im Rahmen der Rollierung zu Beginn jedes Monats neu berechnet, um die Abweichungen vom Jahresplan zu antizipieren. Die Rahmenbedingungen sind identisch zu denen der übergeordneten Ebenen in Abschnitt 6.3.2. Die Planung erfolgt mit einem Modell, welches mit dem in Abschnitt 6.3.2 vorgestellten Modell im Wesentlichen übereinstimmt. Unterschiede gibt es in den zusätzlichen Restriktionen bei Unterschreitung von Zielbeständen. Auf Jahresebene wurde ein Zielbestand am Jahresende vorgegeben, bei dessen Unterschreitung Strafkosten anfallen. Das Ergebnis des Jahresplans waren die Ziel-Lagerbestände auf Monatsbasis. Das Monatsmodell übernimmt diese als Eingabedaten und belegt eine Unterschreitung mit Strafkosten. Es gibt zwei Ansätze, die Zielbestände in die Planung mit einzubeziehen.

Zielbestand für die letzte Periode Das Modell besitzt einen größeren Freiheitsgrad, da der Bestand erst in der letzten Periode erfüllt werden muß. Freie Kapazitäten in den vorangehenden Perioden können so genutzt werden, um Schwankungen besser auszugleichen. Die Losauflagen können sich von denen des übergeordneten Plans unterscheiden.

Zielbestand für jede Periode Das Modell übernimmt Zielbestände für jede Periode, bei deren Unterschreitung ein Strafkostensatz anfällt. Der Freiheitsgrad ist geringer, da durch die Vorgabe der Bestände nur noch die Höhe der Losauflagen berechnet wird und keine Flexibilität mehr für ein Verschieben der Lose innerhalb des Planungshorizonts vorhanden ist.

Die Entscheidung für eine der beiden Alternativen hängt vom Grad der Flexibilität ab, der erreicht werden soll. Innerhalb dieser Untersuchungen wird nur die letzte Periode als Zielwert verwendet. Das Modell ist somit identisch zu der in Abschnitt 6.3.2 vorgestellten Formulierung. Die eingesetzten Schnittebenen können unverändert übernommen werden. Falls die zweite Alternative gewählt wird, muß die Modellformulierung leicht geändert werden, da die Strafkosten für die Lagerbestände jetzt in jeder Periode anfallen. Diese Modifikationen erfolgen analog zu den Gleichungen (6.4) und (6.5). Das Ergebnis dieser Planungsebene ist ein Produktionsplan mit Informationen über Anzahl und Größe der Produktionslose für jedes Produkt und jede Periode.

6.3.4 Aufspaltung in Mikroperioden

Das in Abschnitt 6.3.2 vorgestellte Modell gehört zur Klasse der „big bucket“ Modelle, so daß nur ein Rüstvorgang je Periode modelliert werden kann. Zur Erstellung eines Mengengerüstes auf mittelfristiger Ebene ist diese Intervallgröße meist ausreichend. Auch die Annahme, daß nur ein Rüstvorgang je Periode durchgeführt wird, ist auf dieser Ebene akzeptabel. Es gibt jedoch zwei Problemfelder, die eine detailliertere Betrachtung notwendig machen:

- Wenn die TBO eines Produktes kleiner als eine Periodenlänge ist, dann entstehen Abbildungsfehler, da nur ein großes Los eingeplant wird. Eine geringe TBO bedeutet, daß entweder die Rüstkosten gering oder die Lagerkosten hoch sind. Im ersten Fall verringert sich die Flexibilität, da nur ein großes, statt vieler kleiner Lose eingeplant wird. Im zweiten Fall entstehen zu hohe Kosten, da der mittlere Lagerbestand bei einem großen Los steigt und damit auch die Lagerkosten.
- Wenn die Kapazitäten stark ausgelastet sind, entstehen Fehler beim Übergang vom Jahresplan auf den Monatsplan oder zur Feinplanung. Der Monatsplan kann ein Los auf mehrere (parallele) Maschinen aufteilen, so daß mehrfache Rüstvorgänge innerhalb einer Periode anfallen. Diese verringern die zur Verfügung stehende Kapazität, so daß die Ziellagerbestände nicht erreicht werden können. Daher müssen übergeordnete Ebenen bei einer hohen Auslastung mehrfachen Rüstvorgängen ausreichend berücksichtigen.

Eine Möglichkeit, beide Aspekte bei der Planung zu berücksichtigen ist die Modellierung von mehreren Losen, die innerhalb einer Periode produziert werden können. Damit verbunden ist eine zunehmende Modellgröße, die jedoch im Rahmen dieser Untersuchung nicht weiter ins Gewicht fällt.

Die Abbildung von mehreren Produktionslosen innerhalb einer Periode erfolgt durch eine Unterteilung einer Periode in mehrere Mikroperioden. Aufgrund diese Aufspaltung verhält sich das Modell in jeder Mikroperiode wie in einer normalen Periode. Der Vorteil bei diesem Ansatz ist die einfache Realisierung, da lediglich die Daten anders aufbereitet werden. Da auch der Charakter des Modells erhalten bleibt, wird weiterhin ein Produktionsplan erzeugt, der jetzt jedoch mehrere Produktionslose je Produkt enthalten kann. Die Produktionslose sind voneinander unabhängig und können unterschiedliche Größen annehmen.

Die Aufbereitung der Eingabedaten geschieht unter der Annahme, daß die Mikroperioden sich nicht überlappen, eine identische Länge

besitzen und eine Makroperiode vollständig ausfüllen. Die Anzahl am Mikroperioden innerhalb einer Makroperiode ist für alle Makroperioden gleich. Die Mikroperioden innerhalb einer Makroperiode besitzen identische Eigenschaften und unterscheiden sich nur durch ihre Position. Die Aufbereitung der Eingabedaten basiert auf diesen Annahmen.

Die Nachfrage $d_{j,t}$ bleibt erhalten, wird jedoch an das Ende der letzten Mikroperiode innerhalb einer Makroperiode verlegt. Die Nachfrage in jeder anderen Mikroperiode ist null. Die verfügbare Periodenkapazität wird gleichmäßig auf die Mikroperioden aufgeteilt. Der Sicherheitsbestand hingegen bleibt als untere Grenze in jeder Mikroperiode erhalten. Somit wird verhindert, daß innerhalb einer Makroperiode die Lagerbestände unter den Sicherheitsbestand fallen, ohne daß Strafkosten entstehen. Die Kostensätze für die Unterschreitung des Sicherheitsbestandes sowie die anderen Strafkostensätze werden auf die Mikroperioden gleichmäßig verteilt. Auch die Kosten für die Lagerhaltung werden gleichmäßig auf die Mikroperioden verteilt. Diesem Ansatz liegt die Annahme einer linearen Abhängigkeit zwischen dem Lagerbestand und den Lagerkosten zugrunde, wie sie auch durch die Zielfunktion (6.1) modelliert wird.

Durch die Vervielfachung der Anzahl an Perioden wird auch die Anzahl an Variablen, Nebenbedingungen und Schnittebenen erhöht. Ein kritischer Punkt ist die Anzahl an Nebenbedingungen, die durch die zusätzlichen Schnittebenen entsteht (vgl. Tabelle 6.5 auf Seite 223). Bei einer Aufspaltung in vier Mikroperioden erhöht sich die Anzahl an Perioden um den Faktor 4, wodurch die Anzahl an Schnittebenen auf das 16-fache zunimmt. Als Konsequenz für den Lösungsprozeß ist der erhöhte Speicherverbrauch für die Verwaltung der Schnittebenen zu berücksichtigen, sowie die erhöhte Rechenzeit, die sich durch die Anwendung der Schnittebenen ergibt. Die zusätzlichen Nebenbedingungen, die im Modell entstehen, sind im Vergleich hierzu gering. Problematisch ist auch die Zunahme an Binärvariablen, da diese für die Qualität der Lösung im Branch & Bound Prozeß eine entscheidende Rolle spielen. Es ist zu erwarten, daß durch die Zunahme an Variablen der gesamte Prozeß verlängert wird und in den Fällen, wo der Prozeß vorher abgebrochen wird, eine größere Lücke zwischen der besten

bekannten Lösung und der besten unteren Schranke auftritt. Um die Rechenzeit gering zu halten, ist die Anzahl an Mikroperioden so zu wählen, daß sie - über alle Produkte betrachtet - der maximalen Anzahl der in einer Periode zu produzierenden Lose entspricht.

6.3.5 Feinplanung - Scheduling

Innerhalb eines hierarchischen Produktionsplanungssystems besteht die Aufgabe der Feinplanung darin, eine Reihenfolgeplanung der vorher berechneten Lose auf den zur Verfügung stehenden Ressourcen durchzuführen (Hopp u. Spearman 2001, S. 141). Das Ergebnis der Planung ist eine Zuordnung der Lose zu den Maschinen, sowie ein Startzeitpunkt für jedes Los. Dieser Plan wird zur Auftragsfreigabe an eine Steuerungslogik weitergeleitet.

Die Feinplanung übernimmt die Ausgabedaten der Monatsplanung unverändert. Es wird nur eine Zuordnung der Produktionslose auf den Maschinen mit gleichzeitiger Reihenfolgeplanung durchgeführt. Im Vergleich zur Master Planning (Jahresplan) und Production Planning (Monatsplan) die Feinplanung detailliertere Planungsdaten.

Der Planungshorizont entspricht dem Abstand zwischen zwei Neuplanungen des Production Planning, in diesem Fall ein Monat mit 30 Tagen. Die Zeitachse wird nicht mehr in Perioden eingeteilt, sondern kontinuierlich dargestellt, um eine exakte Simulation der Produktion zu ermöglichen. Die Feinplanung wird zu Beginn jeden Tages durchgeführt, unter Berücksichtigung der jeweils aktuellen Umweltbedingungen. Eine Veränderung der Losgrößen, die vom Production Planning berechnet wurden, erfolgt innerhalb des Planungshorizonts nicht mehr („frozen horizon“).

Aufgrund der höheren zeitlichen Auflösung und der Anbindung an die Simulation sind zusätzliche Anlageneigenschaften zu modellieren, die auf den bisherigen Planungsebenen teilweise nur in aggregierter Form betrachtet wurden. Abweichend von den in Abschnitt 6.3.2 aufgeführten Eigenschaften werden folgende Rahmenbedingungen zusätzlich berücksichtigt:

- die Menge an Kundenaufträgen, die im System zum Planungszeitpunkt bekannt sind. Jeder Kundenauftrag setzt sich zusammen aus der nachgefragten Menge sowie dem Bedarfszeitpunkt, an dem die Lieferung erfolgen soll. Details zur Struktur und zur Erzeugung der Kundennachfrage werden in Abschnitt 6.3.6 erläutert.

- Rüstzeiten hängen von der Maschine und von der Reihenfolge der Produkte. Die Rüstmatrix kann asymmetrisch sein und darf die Dreiecksungleichung verletzen.

- Lagerkosten innerhalb des Planungshorizonts werden nicht berücksichtigt.

- Strafkosten für die Unterschreitung des Sicherheitsbestandes oder für Nachlieferungen sind von der Dauer abhängig und werden nicht mehr auf Basis von Perioden berechnet, sondern auf einer kontinuierlichen Zeitachse. Da auf dieser Ebenen keine Lagerhaltungskosten berücksichtigt werden, stehen sie auch in keiner Relation zu den Strafkosten der anderren Ebenen.

Das Ziel des Modells ist die Erstellung eines zulässigen Produktionsplans, der möglichst geringe Kosten-/Strafkosten aufweist. Kosten entstehen durch die Produktions- und Rüstzeiten, Strafkosten durch die Unterschreitung von Sicherheitsbeständen, physischen Beständen und die verspätete Lieferung eines Auftrags an einen Kunden. Die Anzahl der Lose und deren Größe sind vorgegeben, so daß das Modell nur die Maschinenbelegungsplanung durchführt. Das Modell gehört somit zur Klasse der deterministischen Maschinenbelegungsprobleme. Ein Überblick der verschiedenen Modelle und Algorithmen in diesem Bereich wird von Zhu u. Wilhelm (2006) und Lee u. a. (1997) gegeben.

Die detailliertere Darstellung der Rüstzeiten in der Planung muß bei der Wahl des Lösungsalgorithmus berücksichtigt werden. Innerhalb des Jahres- und Monatsplans wurden die Rüstzeiten als reihenfolgeunabhängig modelliert, d. h. $st_{i,j} = st_j$ für alle Produkt i, j. Diese Vereinfachung war zulässig, da auf diesen Planungsebenen keine Reihenfolgen,

sondern nur Mengen und Kapazitäten betrachtet wurden. Durch den veränderten Planungshorizont und das Planungsziel muß jetzt auch die Reihenfolge der Lose berücksichtigt werden. Der einfachste Fall von reihenfolgeabhängigen Rüstzeiten ist eine symmetrische Matrix, in der gilt $st_{i,j} = st_{j,i}$ für alle Produkte i, j. In einer asymmetrischen Matrix ist diese Bedingung nicht mehr gültig. Die höchste Stufe der Komplexität wird erreicht, wenn auch die Dreiecksungleichung nicht mehr gilt. Die Dreiecksungleichung ist über folgende Beziehung definiert $st_{i,j} + st_{j,k} \geq st_{i,k}$ für alle Produkt i, j, k. Wenn die Dreiecksungleichung gilt, dauert ein Umrüsten von einem Produkt i über Produkt j zu Produkt k mindestens so lang wie der direkte Rüstvorgang von i zu k. Gilt die Dreiecksungleichung nicht mehr, ist es möglich, daß in mindestens einem Fall das Umrüsten über ein Zwischenprodukt weniger Zeit beansprucht, als der direkte Weg. Dieser Fall tritt unter anderem in der chemischen Industrie auf, wenn durch das Einfügen von einem Zwischenprodukt eine Anlage gereinigt wird und somit eine umfangreichere Reinigung bei einem direkten Umrüsten entfällt. Die Planung wird hierdurch allerdings erschwert, wenn bei einer gleichzeitigen Losbildung die Produktionsmenge dieses Zwischenproduktes auf null oder einen sehr kleinen Wert gesetzt wird, da es nur zum Reinigen benötigt wird (Fleischmann u. Meyr 1997, S. 13). Dies kann jedoch auch von Vorteil sein, da ein Los dieses Produktes in mehrere kleine Teile aufgebrochen werden kann, die jeweils eine Reinigung der Anlage durchführen. Eine Lösung hierfür ist die Vorgabe von Mindestlosgrößen (Fleischmann 1994, S. 397). Dies ist hier jedoch nicht notwendig, da die Mengen bereits von der übergeordneten Ebene bereitgestellt werden und die Feinplanung nur noch die Reihenfolgeplanung durchführt. Diese muß jedoch in der Lage sein, eine Planung durchzuführen, auch wenn die Dreiecksungleichung verletzt ist.

Die Komplexität des hier betrachteten Modells ist nicht bekannt. Für den Fall, daß nur eine Maschine mit reihenfolgeabhängigen Rüstzeiten betrachtet wird, läßt sich zeigen, daß dieses Problem NP-schwer ist (Pinedo 2002, S. 79). Ein anderes Problem, bei dem die Produktionsdauer auf kapazitierten parallelen Maschinen minimiert wird, ist ebenfalls

Maschinenbelegungsplanung	Tourenplanung
Maschine	Fahrzeug
Auftrag	Kunde (Knoten)
Produktionszeit	Verweilzeit
Rüstzeit	Fahrzeit
Kapazität einer Maschine	Zeitfenster eines Fahrzeugs
Zeitpunkt der Nachfrage für einen Auftrag	Ende des Zeitfensters für die Bedienung von einem Kunden
Anfangs-/Endzustand	Depots

Tabelle 6.6: Äquivalenz zwischen einem Problem zur Maschinenbelegungsplanung und Tourenplanung

NP-schwer (Pinedo 2002, S. 539). Es ist anzunehmen, daß durch die zusätzlichen Maschinen die Komplexität nicht abnimmt.

Abbildung als Tourenplanungsproblem

Das im vorangehenden Abschnitt vorgestellte Modell zur Maschinenbelegungsplanung läßt sich in ein äquivalentes Modell zur Tourenplanung überführen. Da die Planung für mehrere Maschinen gleichzeitig durchgeführt wird, müssen zwei Entscheidungen getroffen werden: die Zuordnung der Aufträge zu den Maschinen sowie die Reihenfolge (zeitliche Einordnung) der Aufträge auf einer Maschine (Bodin u. a. 1983, S. 149). Unter Berücksichtigung der Zeitfenster ergibt sich ein Standardmodell, das Vehicle Routing and Scheduling Problem with Time-Windows (VRSPTW), wie Solomon (1987, S. 254) es beschreibt. Die Abbildung der Elemente zwischen Maschinenbelegungsplanung und Tourenplanung erfolgt analog zu Fleischmann (1994, S. 397) und Pinedo (2002, S. 541) (vgl. Tabelle 6.6).

Im Gegensatz zu den Standardmodellen mit m-Routen wird kein zentrales Depot von den Fahrzeugen angefahren (Solomon 1987, S.). Jedes Fahrzeug besitzt seine eigenen Depots für den Anfang und das Ende einer Tour. Der Depotknoten am Anfang modelliert die Maschinenbele-

gung zu Beginn des Planungszeitraums. Ist die Maschine noch durch ein anderes Los belegt, so wird der Zeitpunkt, an dem das Fahrzeug das Depot verlassen kann entsprechend angepaßt. Da jede Maschine auch über einen initialen Rüstzustand verfügt, wird dieser dem Depot zugewiesen. Das Ende einer Route wird durch einen zusätzlichen Depotknoten modelliert, der bei der Initialisierung erzeugt wird. Da jeder Auftrag der Letzte auf dieser Maschine sein kann, muß der entsprechende Depotknoten auch von jedem anderen Knoten erreichbar sein, ohne zusätzliche Kosten zu verursachen. Für den Depotknoten wird daher ein zusätzliches Produkt eingeführt. Gleichzeitig wird auch die Entfernungsmatrix ergänzt, so daß die Entfernung zum Endknoten von jedem anderen Knoten null ist. Die Algorithmen sind so gestaltet, daß die Depotknoten ihre Position am Anfang/Ende der Routen nicht verändern können. Die Depots sind somit von außen nicht sichtbar und stellen auch keine Einschränkung der Vorgänger/Nachfolger Beziehungen dar.

Die Abbildung der Lieferzeiten erfolgt über Zeitfenster, die mit einem Kunden assoziiert werden. Details zur Implementierung, sowie zu den Erweiterungen werden später vorgestellt. Ein weiteres Zeitfenster wird für jede Route modelliert. Die Länge des Zeitfensters entspricht der auf einer Maschine zur Verfügung stehende Kapazität. Dieses Zeitfenster definiert auch gleichzeitig die maximale Länge der gesamten Route. Die Länge des Zeitfensters ist fest und kann nicht überschritten werden.

Ein weiterer Aspekt ist die Abbildung der maschinenspezifischen Eigenschaften. Außer den Rüstzeiten sind auch die Produktionszeiten abhängig von der verwendeten Maschine, sowie die mögliche Zuordnung zwischen Aufträgen und Maschinen. Nicht jedes Produkt kann auf jeder Maschine hergestellt werden. In der Tourenplanung entspricht dies Fahrzeugen, die unterschiedliche Eigenschaften besitzen, sowohl bei den Fahrt- und Verweilzeiten, als auch bei der Zuordnung zwischen Auftrag und Fahrzeug.

Zeitfenster Jeder Kundenauftrag besitzt einen Zeitpunkt, bis zu dem die Lieferung erfolgt sein muß. Die Tourenplanung kann dies über Zeitfenster abbilden. Ein Zeitfenster besteht aus einem frühesten und

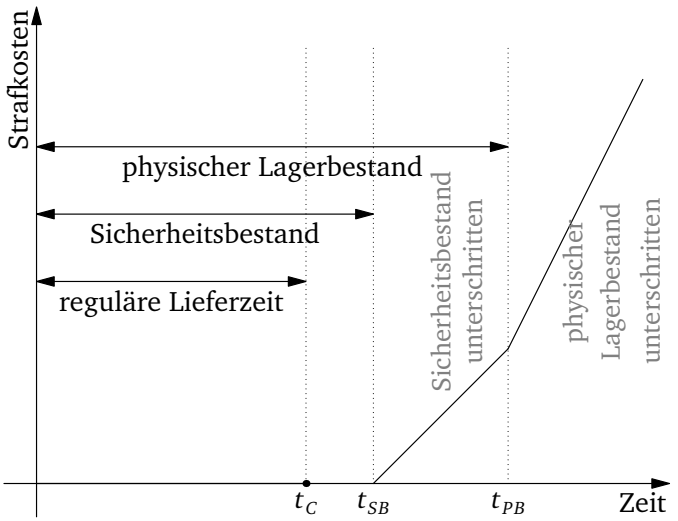


Abbildung 6.10: Aufbau der Zeitfenster und Strafkosten

spätesten Zeitpunkt, an dem die Lieferung erfolgen soll. Kann die Lieferung nicht innerhalb dieses Zeitfensters erfolgen, so gibt es zwei Alternativen: Die Lieferung wird abgewiesen oder sie wird zugelassen, unter Berücksichtigung von Strafkosten (Koskosidis u. a. 1992, S. 71). Da das Modell Nachlieferungen zuläßt, muß es auch möglich sein, daß die Zeitfenster der Kunden überschritten werden (weiche Zeitfenster).

Das Konzept der Zeitfenster wird erweitert, um auch den aktuellen Lagerbestand und den Sicherheitsbestand in der Feinplanung zu berücksichtigen. Außer dem Zeitfenster für die Lieferung werden Zeitfenster für die Unterschreitung des Sicherheitsbestandes, sowie für die Unterschreitung des physischen Lagerbestandes erzeugt. Zur Erstellung der Zeitfenster übernimmt die Feinplanung Daten über die aktuellen Lager- und Fehlbestände sowie die im System bekannten Aufträge. Die Zeitfenster bauen hierarchisch aufeinander auf, d. h. das Zeitfenster für die Lieferung ist im Zeitfenster für den Sicherheitsbestand enthalten oder mit ihm identisch. Das Zeitfenster für den physischen Lagerbestand ist wiederum dem für den Sicherheitsbestand übergeordnet.

Durch einen Abgleich der bekannten Aufträge mit den zu produzierenden Losen werden Zeitfenster für jedes Los festgelegt. Diese Zeitfenster können je nach Lagerbestand und Auftragslage im aktuellen Zeitpunkt enden (wenn z. B. kein Lagerbestand vorhanden ist oder der Sicherheitsbestand bereits unterschritten ist) oder unbeschränkt sein (wenn keine Aufträge vorhanden sind oder der Lagerbestand ausreichend hoch ist). Abbildung 6.10 auf der vorherigen Seite zeigt den Zusammenhang zwischen den Zeitfenstern sowie den Strafkosten für deren Überschreitung. Der Auftrag soll bis zum Zeitpunkt t_C erfüllt sein. Wird er in der Zeit von $[0; t_C]$ erfüllt, so fallen keine Strafkosten für die vorgezogene Lieferung an. Die Ware steht im Lager bereit und es entstehen Lagerkosten (die hier aber nicht erfaßt werden). Erfolgt die Lieferung im Intervall $]t_C; t_{SB}]$, so kann die Lieferung noch aus dem aktuellen Lagerbestand erfüllt werden. Bei einer weiteren Verzögerung der Lieferung wird zuerst der Sicherheitsbestand bei t_{SB} und danach der physische Lagerbestand bei t_{PB} unterschritten, wofür entsprechend höhere Strafkosten anfallen.

Lösungsverfahren Es gibt zwei Arten von Lösungsverfahren für diese Probleme - exakte Verfahren und Heuristiken. Zur ersten Gruppe gehören der Branch & Bound Algorithmus, Constraint Programming, Spaltengenerierungsverfahren sowie Varianten oder Kombinationen dieser Algorithmen. Aufgrund der Komplexität verhält sich die Laufzeit nicht-polynomial, so daß bei einer entsprechenden Größe des Modells diese Verfahren innerhalb einer vorgegebenen Zeitschranke nicht zu einer optimalen Lösung führen werden, bzw. dies nicht zu belegen ist. Die zweite umfangreichere Gruppe beinhaltet heuristische Lösungsverfahren. Diese lassen sich weiter unterteilen in Metaheuristiken, deren Funktionsweise auf unterschiedliche Problemgruppen anwendbar ist, sowie problemspezifische Heuristiken.

Eine grundlegende Klassifikation der verschiedenen Arten von Tourenplanungsproblemen und die Einordnung des VRSPTW werden in Bodin u. a. (1983, S. 149) und Solomon u. Desrosiers (1988) vorgenommen. Einen aktuellen Überblick der verschiedenen Modellformulie-

rungen und exakten Algorithmen gibt Kallehauge (2008). Die Anwendung verschiedener klassischer heuristischer Ansätze wird von Solomon (1987, S. 255) sowie von Cordeau u. a. (2002, S. 514) vorgestellt und bewertet. Alternative Ansätze, die nicht auf einer problemspezifischen Heuristik basieren, sondern eine Metaheuristik einsetzen werden von (Cordeau u. a. 2001, S. 929), (Potvin u. a. 1996, S. 158) (Tabu-Search) und (Potvin u. Bengio 1996, S. 165) (Evolutionäre Algorithmen) vorgestellt.

In ihrer Untersuchung kommen Kurz u. Askin (2001, S. 3767) zu dem Schluß, daß eine modifizierte Insertion-Heuristik bei dieser Problemgruppe am besten geeignet ist, um in kurzer Zeit Lösungen zu erzeugen, die sich nahe am Optimum befinden. Eine vergleichbare Heuristik soll auch hier eingesetzt werden, da die Simulation am häufigsten die Feinplanung aufruft und somit die Rechenzeit zu einem kritischen Faktor wird.

Eröffnungsheuristik Zur Erzeugung einer ersten zulässigen Lösung wird eine Eröffnungsheuristik benutzt. Diese basiert auf der Arbeit von Potvin u. Rousseau (1993). Der Algorithmus wurde entwickelt, um parallel n-Routen zu erzeugen. Die Anzahl der Routen muß vorher nicht bekannt sein, sondern kann geschätzt werden. Da die Anzahl der Maschinen vorher bekannt ist, entfällt dieser Teil des Algorithmus. Auch die Initialisierung durch den am weitesten entfernt liegenden Kunden (Potvin u. Rousseau 1993, S. 333) entfällt, da die Depots bekannt sind und jede Tour mit einem Start- und Endknoten versehen wird. Die Erzeugung der Routen erfolgt mit Hilfe des auf der gegenüberliegenden Seite gezeigten Algorithmus (vgl. Potvin u. Rousseau 1993, S. 333, die Bezeichnungen wurden übernommen).

Die Heuristik übernimmt die Menge an nicht geplanten Kundenaufträgen sowie die Menge an Routen mit den bereits geplanten Kunden - diese Menge kann auch leer sein. Für jeden Kundenauftrag werden anschließend für jede Route und jede Kante innerhalb dieser Route die Kosten $c_{1r}(r_r, u, j_r)$ bestimmt, die durch das Einfügen des Auftrags entstehen. Die Kosten setzen sich aus den mit α_i gewichteten c_{1ir} zusammen.

Globale Konstanten

$\alpha_1 \ldots \alpha_5$ {Gewichtungsfaktoren der (Straf-)Kosten. Es gilt $\sum_{i=1}^{5} \alpha_i = 1$}

Parameter

U {Menge der nicht geplanten Kunden}

R {Menge der Routen}

$\forall u \in U$

$\forall u \in U$

Ab hier werden für jeden Auftrag alle Einfügepunkte überprüft und mit Kosten bewertet.

$\forall r \in R$

$m \leftarrow$ Anzahl an Kanten der aktuellen Route r

$\forall p \in 1 \ldots m$

Kosten für das Einfügen von u in r an Position p berechnen

$c_{11r} \leftarrow \Delta$ Entfernung

$c_{12r} \leftarrow \Delta$ Ankunftszeit der nachfolgenden Kunden

$c_{13r} \leftarrow$ Verweildauer

$c_{14r} \leftarrow$ Dauer der Unterschreitungen des Sicherheitsbestandes

$c_{15r} \leftarrow$ Dauer der Unterschreitungen des Lagerbestandes

$c_{1r}(i_r, u, j_r) \leftarrow \sum_{n=1}^{5} \alpha_n \cdot c_{1nr}(i_r, u, j_r)$

$c^*_{1r}\left(i_r(u), u, j_r(u)\right) \leftarrow \min_{p=1\ldots m}\left[c_{1r}(r_r, u, j_r)\right]$

$c^*_{1r'}\left(i_{r'}(u), u, j_{r'}(u)\right) \leftarrow \min_{r \in R}\left[c^*_{1r}(i_r(u), u, j_r(u))\right]$

$c_2(u) \leftarrow \sum_{r \in R, r \neq r'}\left[c^*_{1r}(i_r(u), u, j_r(u)) - c^*_{1r'}(i_{r'}(u), u, j_{r'}(u))\right]$

Wähle den Kunden u aus, für den gilt:
$c_2(u^*) \leftarrow \max_u\left[c_2(u)\right]$

Füge u an der entsprechenden Position in R ein.

R, U

Abbildung 6.11: Parallele Erzeugung von Routen

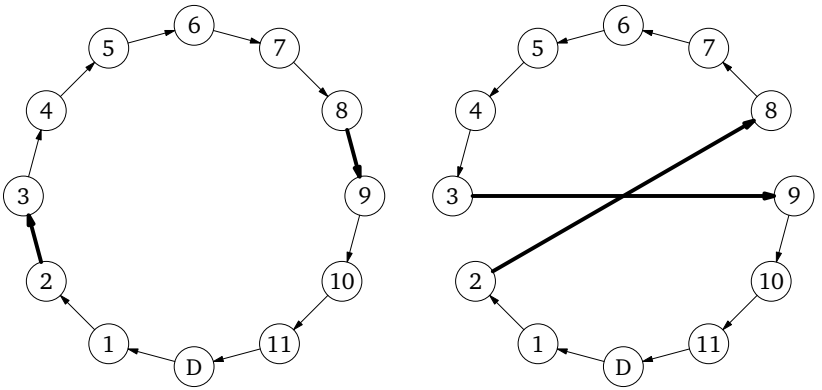
Es werden fünf Komponenten betrachtet: die Differenz der Entfernung c_{11r}, die Verschiebung der Ankunftszeit bei nachfolgenden Kunden c_{12r}, die Verweildauer c_{13r} und die Zeit während der Sicherheitsbestand c_{14r} und Lagerbestand c_{15r} unterschritten werden. Die Gewichtung der Faktoren untereinander kann beliebig eingestellt werden. Aufgrund der Konstruktion der Zeitfenster muß die Unterschreitung des Lagerbestandes (Backlog) höher gewichtet werden, als die Unterschreitung der Sicherheitsbestände. Für jede Route werden so die minimalen Einfügekosten c^*_{1r} berechnet. In einem zweiten Schritt werden die minimalen Einfügekosten über alle Routen c^*_{1r} bestimmt. Die Summe der Differenz zwischen den minimalen Einfügekosten je Route und den minimalen Einfügekosten über alle Routen ergeben die „Regret"-Kosten $C_2(u)$ für einen Kunden u (Potvin u. Rousseau 1993, S. 333). Die Regret-Kosten sollen ausdrücken, was passieren kann, wenn die aktuelle beste Lösung nicht realisiert wird und auf die Nächstbeste ausgewichen werden muß. Daher wird von allen Aufträge der Auftrag ausgewählt, dessen Regret-Kosten maximal sind, da er am schwierigsten einzufügen ist. Umgekehrt werden so Aufträge, die leicht einzufügen sind auf einen späteren Zeitpunkt verschoben, da sie mit einer hohen Wahrscheinlichkeit auch bei einem dichter besetzten Plan leichter einzufügen sind. Kann ein Auftrag in einer Route aufgrund nicht vorhandener Kapazität nicht eingefügt werden, so wird die Route nicht berücksichtigt. Kann ein Auftrag in keiner Route eingefügt werden, so wird er zurückgestellt und am Ende in der Menge der ungeplanten Aufträge zurückgegeben. Der Algorithmus wird solange ausgeführt, bis alle Kundenaufträge entweder erfolgreich eingeplant oder in die Liste der ungeplanten Aufträge eingetragen wurden. Diese Liste wird in einem zweiten Schritt nach Durchführung einer Verbesserungsheuristik weiter verarbeitet.

Bei der Anwendung auf kapazitätsbeschränkte Maschinen kann es jedoch vorkommen, daß viele Aufträge nicht eingeplant werden können. Da das Verfahren zuerst die ungünstigen oder großen Aufträge auf den Maschinen einplant, die schnell produzieren können oder insgesamt günstiger sind, werden diese Maschinen auch zuerst voll ausgelastet. Die anderen Aufträge werden entsprechend der Vorteilhaftigkeit auf die restlichen Maschinen verteilt. Dabei kann es jedoch vorkommen, daß

gegen Ende der Verteilung einige Aufträge nicht mehr berücksichtigt werden können. Um dem vorzubeugen, wird während der Verteilung die Auslastung berücksichtigt. Bei jeder Iteration werden die Kosten $c_{1r}(i_r, u, j_r)$ mit einem Faktor $1 + \alpha \cdot \mathit{Auslastung}$ multipliziert. Eine Maschine, die nur gering ausgelastet ist, wird somit eher das Minimum c^*_{1r} annehmen, als eine Maschine mit hoher Auslastung. Umgekehrt wird so auch verhindert, daß eine Maschine mit hoher Auslastung noch mehr belastet wird, speziell wenn andere Maschinen nur eine geringe Auslastung haben, aber der Auftrag eine höhere Rüst- oder Produktionszeit hat. Durch einen weiteren Gewichtungsfaktor läßt sich einstellen, wie stark die Auslastung in die Kosten einfließt.

Verbesserungsheuristik Nachdem die Eröffnungsheuristik die Aufträge auf die Routen verteilt hat, kann die bestehende Lösung durch Anwendung unterschiedlicher Heuristiken weiter verbessert werden. Die Anwendung von Verbesserungsheuristiken wird durch die Existenz von Zeitfenstern eingeschränkt, gute Einsatzmöglichkeiten bieten Verfahren, die innerhalb einer Route einzelne Kunden oder Sequenzen von Kunden vertauschen (Potvin u. Rousseau 1995, S. 1434). Hierzu gehören insbesondere Verfahren wie Or-Opt und 2-Opt, welches zur Klasse der k-Opt-Verfahren gehört (Psaraftis 1983, S. 392). Beide Verfahren lassen sich getrennt oder nacheinander anwenden, als hybrides Verfahren, wie von Potvin u. Rousseau (1995, S. 1437) vorgestellt. In dem hier vorgestellten Testaufbau wird zuerst das Or-Opt-Verfahren und anschließend das 2-Opt-Verfahren durchgeführt. Zur Bewertung werden in beiden Verbesserungsheuristiken nicht alle Kriterien der Eröffnungsheuristik übernommen. Die Unterschreitung der Lagerbestände und die Verschiebung der Startzeitpunkte werden vernachlässigt, da sie für die Auswahl der Routen (Maschinen) relevant waren. Zur Bewertung einer bestehenden Route reicht die Reise- und Verweildauer (Rüstzeit und Produktionszeit) sowie die Unterschreitung der Sicherheitsbestände aus. Die Unterschreitung der Lagerbestände wird vernachlässigt, da dieser immer eine Unterschreitung der Sicherheitsbestände vorangeht.

Das 2-Opt-Verfahren basiert auf dem Austausch von zwei Kanten



(a) Route vor einer Iteration (b) Route nach einer Iteration

Abbildung 6.12: Eine Iteration des 2-Opt-Verfahrens

in einer Route. Der Teil der Route, der zwischen diesen Kanten liegt, wird in der Orientierung umgekehrt. Diese Iteration wird für jede Kombination zwischen zwei Kanten i und j durchgeführt. Die Ordnung dieser Heuristik liegt bei $O(n^2)$ (Psaraftis 1983, S. 392). Das Beispiel auf dieser Seite zeigt einen Austausch der Kanten zwischen den Knoten (2,3) und (8,9). Nach Durchführung der Iteration ist die Teilroute von Knoten 3 bis Knoten 8 umgekehrt, so daß die Route in der Reihenfolge $D-1-2-8-7\ldots-4-3-9\ldots$ durchlaufen wird.

Das Or-Opt-Verfahren benutzt hingegen Segmente der Größe $1-3$, die innerhalb einer Route an einer anderen Stelle wieder eingefügt werden (vgl. Or 1976, S. 105 und Or u. Pierskalla 1979, S. 91). Im Gegensatz zum 2-Opt-Verfahren bleibt die Orientierung der Segmente erhalten. Eine detaillierte Übersicht dieser und anderer Algorithmen zur Lokalen-Suche bei Tourenplanungsproblemen mit Zeitfenstern wird von Savelsbergh (1990, S. 76) vorgestellt.

Kombination aus Eröffnungs- und Verbesserungsheuristik Bei der Erstellung der Routen durch die Eröffnungsheuristik kann es vorkommen, daß einige Aufträge aufgrund von Kapazitätsbeschränkungen

nicht eingeplant werden können. Die Menge dieser Aufträge wird nach Durchführung der Eröffnungsheuristik zurückgegeben. Zur erneuten Einplanung wird, nachdem die Verbesserungsheuristik die Routen verändert hat, die Eröffnungsheuristik auf die so entstandene Lösung angewendet. Durch die veränderte Auftragsfolge kann sich die freie Kapazität erhöht haben, so daß wieder Aufträge eingefügt werden können. Dieser Zyklus aus Eröffnungsheuristik und Verbesserungsheuristik wird über drei Runden ausgeführt. Aufträge, die am Ende der dritten Runde nicht eingeplant werden konnten, werden verworfen.

Erweiterung des Lösungsraums Eine weitere Möglichkeit, den Lösungsraum für die Eröffnungsheuristik und Verbesserungsheuristik zu erweitern, ist die Nutzung der gleichen Route in umgekehrter Reihenfolge. Ein ähnliches Vorgehen wird von Muyldermans u. a. (2005, S. 984) zur Vergrößerung des Lösungsraums vorgeschlagen. Technisch wird das Verfahren so umgesetzt, daß zu einer Vorwärts-Route immer parallel eine Rückwärts-Route erzeugt oder abgeleitet wird.

Die Eröffnungsheuristik wird so modifiziert, daß nach der Berechnung vom besten Einfügepunkt der Vorwärtsroute die gleiche Berechnung für die Rückwärtsroute durchgeführt wird. Anschließend werden die so entstandenen Kosten miteinander verglichen und die Route mit den geringsten Kosten wird zur neuen Vorwärtsroute. Kann eine Rückwärtsroute nicht gebildet werden, weil die Kapazität zu gering ist, so bekommt sie einen Strafkostensatz zugewiesen, der dem eines nicht zugewiesenen Auftrags entspricht.

Bei der Durchführung der Verbesserungsheuristiken wird diese Umkehr nur für das 2-Opt-Verfahren durchgeführt, da es am meisten profitiert. Wie aus Abbildung 6.12b auf der gegenüberliegenden Seite ersichtlich ist, wird die Teilroute $3-4-\ldots7-8$ umgekehrt zu $8-7-\ldots-4-3$. Dieser umgekehrte Teilabschnitt existiert aber bereits in der Rückwärtsroute, so daß er nicht erneut berechnet werden muß. Durch einen Austausch der Segmente zwischen der Vorwärts- und Rückwärtsroute kann die Neuberechnung auf die Kanten und die Zusatzinformationen (Überschreitungen von Zeitfenstern) ab der ersten modifizierten Kante

beschränkt werden. Gegenüber einer klassischen Implementierung mit vollständiger Neuberechnung sinkt die Rechenzeit.

Die vollständige Umkehr der Routen tritt bei der Erzeugung meist während der ersten Iterationen auf, da hier die Auftragsfolge noch nicht sehr groß ist und so viel Potential zu Verbesserung besteht. Die Umkehr kann nicht nur einmal, sondern auch mehrmals eintreten, so daß während der ersten Schritte die Richtung teilweise mit jeder Iteration wechselt. Innerhalb der Verbesserungsheuristik läßt sich dieses Verhalten am 2-Opt-Verfahren beobachten, wenn die zu vertauschenden Kanten weit auseinander liegen. Die Richtung der Knoten, die zwischen den Kanten liegen, bleibt erhalten, während die der umgebenden Knoten vertauscht wird.

Ausblick auf Weiterentwicklungen Die eingesetzten Heuristiken besitzen aufgrund der Zeitfenster und ihrer Berechnung eine strukturelle Schwäche. Die Heuristiken übernehmen aus der Monatsplanung die Lose und versehen diese mit Zeitfenstern. Die Reihenfolge, in der die Lose übernommen werden, entspricht der Reihenfolge, in der die Produktion angenommen wird und anhand derer die Zeitfenster berechnet werden. Die Festlegung der Zeitfenster geht somit schon von einer Reihenfolge aus, die ja das Ziel der Planung sein soll. Daher ist es möglich, daß durch eine Veränderung der Bearbeitungsreihenfolge, nach Berechnung der Zeitfenster, eine bessere Lösung gefunden wird. Die Auswirkungen sind in den meisten Fällen jedoch gering. Die Einschränkungen sind meßbar, sobald der Sicherheits- oder Lagerbestand unterschritten wird. Die Strafkosten dominieren die anderen Kosten, so daß eine andere Anordnung nicht mehr realisiert wird.

Für spätere Untersuchungen kann durch eine Neuentwicklung dieses Problem umgangen werden. Ausgehend von der Menge der bekannten Aufträge, dem Sicherheitsbestand sowie den Produktionslosen kann jede von der Heuristik erzeugte Kombination alleine durch Berechnung des Bestandsverlaufes bewertet werden. Somit kann auch auf den Einsatz von Zeitfenstern verzichtet werden, da die Dauer der Unterschreitung des Sicherheitsbestandes aus dem Bestandsverlauf ge-

wonnen wird. Ein weiterer Vorteil ist die Möglichkeit, kontinuierliche Produktionsprozesse in der Heuristik korrekt abzubilden. Dies ist mit dem bisherigen Algorithmus nicht möglich.

6.3.6 Kundennachfrage

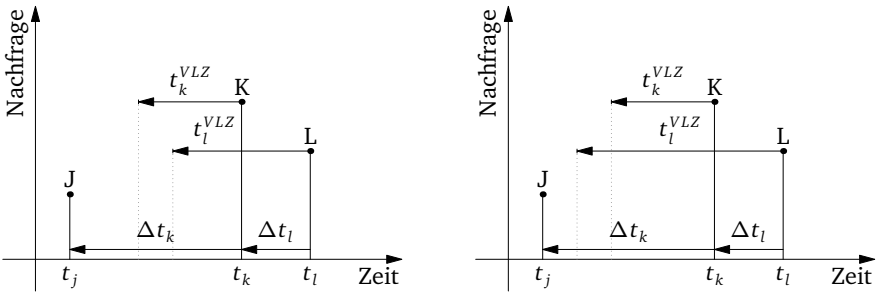
Die Einbindung der Unsicherheit in die hierarchische Produktionsplanung erfolgt durch die Kundenaufträge. Für die Jahres- und Monatsplanung wird angenommen, daß die Nachfrage deterministisch ist. In der Realität setzen sich die Daten aus Prognosen unterschiedlicher Güte zusammen und sind somit Schwankungen unterworfen. Die Feinplanung hingegeben muß in jedem Zeitpunkt der Planung eine neue Umweltsituation berücksichtigen, die durch die Kundennachfrage entsteht.

Die Modellierung der stochastischen Nachfrage erfolgt über einzelne Kunden, die nur jeweils ein Produkt bestellen. Andere Kombinationen, wie z. B. ein Kunde, der mehrere Produkte bestellt oder mehrere Kunden, die das gleiche Produkt bestellen, sind ebenfalls denkbar und können Gegenstand weiterer Untersuchungen sein.

Ein Nachfrageereignis wird über drei Eigenschaften charakterisiert, die aus einer stochastischen Verteilung ermittelt werden. Die Verteilungen sind für jede Kunde - Produkt Beziehung definiert.

- Höhe der Nachfrage
- Zeitpunkt der Nachfrage - Abstand zwischen zwei Nachfrageereignissen
- Vorlaufzeit der Information - Zeitraum zwischen der Bekanntgabe im System und dem Bedarfszeitpunkt

Die Erzeugung einer Nachfrage verläuft immer in folgender Reihenfolge: Das Planungssystem fragt zu einem Zeitpunkt t_i ab, welche Aufträge in der Zeit seit t_{i-1} eingetroffen sind. Um diese zu ermitteln, werden so viele Nachfrageereignisse erzeugt, bis sichergestellt ist, daß mit maximaler Vorlaufzeit alle Ereignisse erzeugt wurden, die bis spätestens t_i bekannt gegeben werden müssen. Die Erzeugung eines



(a) Normalfall (b) Überholung von Aufträgen

Abbildung 6.13: Erzeugung von stochastischer Kundennachfrage

einzelnen Ereignisses erfolgt dabei immer in der gleichen Reihenfolge. Zuerst wird der Zeitpunkt der nächsten Nachfrage bestimmt, danach die Höhe der Nachfrage. Anschließend wird das Ereignis in eine Liste einsortiert, die aufsteigend nach dem Zeitpunkt der Bekanntgabe im System geordnet ist. Aus dieser Liste werden die Ereignisse bei der Abfrage durch das Planungssystem entnommen.

Abbildung 6.13a zeigt diesen Standardfall. Auf einen Auftrag J folgen zwei weitere Aufträge K und L. Die Abstände zwischen den Aufträge J, K und L betragen Δt_k und Δt_l. Die Vorlaufzeit von Auftrag K liegt bei t_k^{VLZ}, er wird somit bei $t_k - t_k^{VLZ}$ im System bekannt. Der Zeitpunkt der Bekanntgabe im System für den Auftrag L liegt bei $t_l - t_l^{VLZ}$. Die Zeiten der Bekanntgabe beider Aufträge liegen hier auch in der Reihenfolge vor, in der die Aufträge später erfüllt werden sollen.

In Abhängigkeit von den Verteilungen kann jedoch auch der Fall eintreten, daß sich die Aufträge überholen, wie in Abbildung 6.13b gezeigt wird. Auftrag L wird zuletzt erzeugt, ist jedoch zu einem früheren Zeitpunkt im System bekannt, als Auftrag K. Dieser Fall zeigt, daß für die Bereitstellung aller Informationen im Planungssystem ein separates Objekt sicherstellen muß, daß zu jedem beliebigen Zeitpunkt alle Aufträge erfaßt werden, die im System bekannt sind. Während der Erzeugung der Aufträge durch ein Kundenobjekt wird überprüft, ob

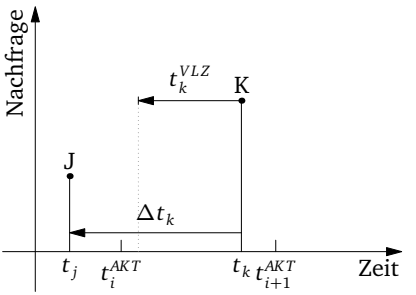


Abbildung 6.14: Risikozeitraum durch Planaktualisierung

der Zeitpunkt der Bekanntgabe eines Auftrags die maximale Vorlaufzeit überschreitet. Erst wenn dieser Auftrag erzeugt wurde, ist sichergestellt, daß keine weiteren Aufträge im System erzeugt werden müssen.

Ein besonderer Fall kann bei der intervallbezogenen Planung innerhalb eines hierarchischen Planungssystems auftreten. In Abbildung 6.14 wird zum Zeitpunkt t_i^{AKT} der Zustand des Systems abgefragt. Die nächste Abfrage findet bei t_{i+1}^{AKT} statt. Zwischen diesen beiden Zeitpunkten wird ein Kundenauftrag K erzeugt, der eine so geringe Vorlaufzeit t_k^{VLZ} hat, daß diese zwischen den beiden Zeitpunkten liegt. Dieser Auftrag kann von der Planung in t_{i+1}^{AKT} nicht mehr berücksichtigt werden, da er zu diesem Zeitpunkt bereits ausgeliefert sein sollte. Solche kurzfristigen Aufträge stellen ein nicht planbares Risiko dar, daß unter anderem durch Sicherheitsbestände abgedeckt werden muß. Die Häufigkeit des Auftretens hängt sowohl von der Charakteristik der Kundennachfrage, als auch von der Zeit zwischen zwei Aktualisierungen ab. Somit beeinflußt die Konfiguration des Planungssystems direkt die Höhe der Sicherheitsbestände. Ein weiteres Problem ergibt sich aus der hier verwendeten Antizipation der Umweltbedingungen. Da die Feinplanung nur eine Reihenfolgeplanung vornimmt, kann sie die Produktionsmenge innerhalb eines Monats nicht an die Kundennachfrage anpassen („frozen horizon“). Dies ist erst zu Beginn des folgenden Monats möglich, so daß ein weiterer Risikozeitraum von maximal einem Monat entsteht, in dem die Mengen nicht angepaßt werden können.

Kapitel 7

Ein Testdatengenerator für Losgrößenprobleme

Für die Simulation und Analyse des Verhaltens eines hierarchischen Produktionsplanungssystems werden Testdaten benötigt, die vielfältige Anforderungen in Größe und Detailgrad erfüllen müssen. Die bisher in der Literatur vorgestellten Testdaten sind nur auf jeweils eine Modellformulierung zugeschnitten und beinhalten meist auch keinen Algorithmus, der eine Erzeugung anhand von unterschiedlichen Anforderungen erlaubt. Da im Rahmen dieser Arbeit auch die Einflüsse bei der Veränderung von einzelnen Parametern untersucht werden sollen, wurde ein neuer Testdatengenerator für einstufige Losgrößenprobleme entwickelt. Die Entwicklung orientiert sich an den in Kapitel 3.4 und Tabelle 3.1 auf Seite 45 vorgestellten Schritten.

Das Ziel der Entwicklung ist ein Testdatengenerator, der umfangreiche Möglichkeiten bietet, den Charakter der Testdaten zu beeinflussen, sowie gleichzeitig die Anforderungen aus Abschnitt 3.3.1 erfüllt. Die Daten sollen für ein breites Spektrum an deterministischen einstufigen Losgrößenmodellen geeignet sein, sowie in Kombination mit der Simulation die Charakteristika von stochastischen Kundenbedarfen so nachbilden, daß die deterministische Nachfrage und die stochastische Nachfrage in ihren Kennzahlen übereinstimmen.

7.1 Definition der Modellgruppen - Problemformulierung

Der Testdatengenerator soll eingesetzt werden, um das Verhalten von einstufigen Losgrößenmodellen zu untersuchen. Innerhalb dieser Produktionsstufe können beliebig viele Maschinen parallel arbeiten. Die Anzahl an Produkten ist ebenfalls nicht begrenzt. Die Zuweisung von Produkten an Maschinen kann beliebig erfolgen - es ist möglich, Produkte nur einer Maschine zuzuweisen oder mehrere mögliche Maschinen anzugeben. Die Maschinen müssen in ihren Eigenschaften nicht identisch sein. Die Produkt-Maschine Kombinationen können sich in ihren Eigenschaften voneinander unterscheiden. Jede Maschine besitzt eine Kapazitätsbeschränkung, die nicht für jede Periode identisch sein muß. Die Kundennachfrage erfolgt am Ende einer Periode und muß nicht sofort erfüllt werden, sondern kann ohne zeitliche Beschränkung zurückgestellt werden (Backlog, Zangwill 1966, S. 105).

Die so erzeugten Testdaten sind geeignet für den Einsatz in „big bucket“ Modellen (Eppen u. Martin 1987, S. 832), sowie mit einigen Einschränkungen auch für „small bucket“ Modelle. Der bekannteste Vertreter der big bucket Modelle ist das CLSP, wie es z. B. von Dixon u. Silver (1981, S. 24) beschrieben wird. Weitere Einsatzmöglichkeiten sind Varianten des CLSP mit parallelen Maschinen, Erhaltung der Rüstzustände und periodenübergreifendem Rüsten (Dillenberger u. a. 1994). Einen Überblick der hier beschriebenen und anderen Erweiterungen des CLSP findet sich in Pochet u. Wolsey (2006, S. 273), Quadt u. Kuhn (2008) sowie in Sung u. Maravelias (2008).

Eine weitere Anforderung betrifft die Länge der Rüstzeiten - es wird davon ausgegangen, daß die Länge eines Rüstvorgangs eine Periodenlänge nicht überschreitet. Erweiterungen des CLSP und PLSP, wie sie von Suerie (2006, S. 878) vorgestellt werden, enthalten Rüstvorgänge, die über Periodengrenzen hinweggehen und auch mehr als eine Periode umfassen können. Ein Einsatz der Testdaten zur Untersuchung dieser Erweiterung wird nicht empfohlen. Durch die Art der Erzeugung der Rüstzeiten (vgl. Abschnitt 7.2.11) wird davon ausgegangen, daß exakt

ein Rüstvorgang pro Produkt und Periode stattfindet. Eine Verlängerung der Rüstzeiten ohne Berücksichtigung der periodenübergreifenden Rüstvorgänge führt zu strukturellen Fehlern, die denen in Abschnitt 7.2.11 beschriebenen ähnlich sind.

Die in Abschnitt 6.3.4 beschriebene Aufteilung in Mikroperioden berücksichtigt der Testdatengenerator. Davon betroffen ist die Berechnung der Rüstzeiten, da diese im Gegensatz zum Standard CLSP jetzt mehrmals innerhalb einer (Makro-) Periode anfallen. Die Verwendung der Daten in small bucket Modellen vom Typ PLSP (Drexl u. Haase 1995) oder GLSP (Fleischmann u. Meyr 1997) ist durch die Mikroperioden auch möglich. Da die Modelle jedoch auch die Rüstzeiten und die Fertigungsreihenfolge berücksichtigen, kommt es zu systematischen Fehlern. Der Testdatengenerator berücksichtigt diese und erzeugt zulässige Daten, er kann den strukturellen Fehler jedoch nicht kompensieren, so daß die Qualität der Daten im Gegensatz zum CLSP abnimmt (vgl. Abschnitt 7.3).

Entsprechend den hier aufgeführten Modellgruppen ergeben sich die in den Tabellen 7.1 bis 7.6 auf den Seiten 248–253 aufgeführten Merkmale, die der Testdatengenerator erzeugen muß, bzw. als Eingabeparameter gegeben sein müssen. Die Abfolge der Tabellen entspricht auch der Reihenfolge, in der die Daten erzeugt oder verarbeitet werden. Die bei den Einträgen angegebenen Variablen entsprechen den im Versuchsaufbau in Abschnitt 6.3.2 auf Seite 211 angegebenen Bezeichnern.

Bei der Wahl der Parameter, die Einfluß auf die Mikroperioden und die deterministische/stochastische Nachfrage haben, sind die in Abschnitt 7.2.12 aufgeführten Empfehlungen zu beachten.

7.2 Modellierung

Die Einordnung anhand der in Abschnitt 3.3.2 vorgestellten Klassifikationskriterien zeigt, daß es sich um einen Generator mit loser Datenbindung handelt. Als Eingabevektoren werden die Kennziffern und Verteilungen aus den Tabellen 7.1 bis 7.5 auf den Seiten 248–252

Indexmenge	Beschreibung
Produkte (J)	Die Anzahl an Produkten kann zwischen 1 und 50 liegen. Die Beschränkung auf 50 Elemente ist durch die Implementierung festgelegt, da jedem Produkt ein Namen aus einem Feld zugewiesen wird. Die Beschränkung kann durch ein größeres Feld oder einen Generator für Produktnamen aufgehoben werden.
Maschinen (M)	Die Anzahl an parallelen Maschinen kann zwischen 1 und 10 festgelegt werden. Die Beschränkung auf zehn Maschinen ist durch die Implementierung vorgegeben, kann jedoch geändert werden.
Perioden (T)	Anzahl an Perioden für die Daten erzeugt werden sollen. Dieser Wert muß nicht der Länge des Planungshorizonts entsprechen, sondern kann mit Hilfe der Zeitfensterfunktionen davon abweichen.

Tabelle 7.1: Indexmengen

Parameter	Beschreibung
Transformation der Zeitleiste	Zur Übertragung der Zeitleiste des deterministischen Jahresplans auf ein rollierendes Monatsmodell, sowie der Simulation sind mehrere Kennzahlen notwendig. Der Aufbau der rollierenden Zeitfenster sowie alle relevanten Parameter werden in Abschnitt 7.2.2 vorgestellt.
Startwert des Zufallszahlengenerator	Der Startwert eines Zufallszahlengenerators bestimmt zusammen mit der Postion des Teilstroms die Werte, die aus einer Verteilung gezogen werden. Zur Unterscheidung zwischen Testdatengenerator und Simulation, wird der Startwert auf die ersten 48 Nachkommastellen von π gesetzt. Die Vektoren IV_1 und IV_2 beschreiben die Initialisierungsvektoren des Zufallszahlengenerators von L'Ecuyer u. a. (2002b). $$IV_1 = \begin{pmatrix} 14159265 \\ 35897932 \\ 38462643 \end{pmatrix} IV_2 = \begin{pmatrix} 38327950 \\ 28841971 \\ 69399375 \end{pmatrix}$$
Nummer des ersten Teilstroms im Zufallszahlengenerator	Durch die Verwendung eines Zufallszahlengenerators mit Teilströmen ist es möglich anzugeben, welche Konfiguration und Stichprobe erzeugt werden soll. Ausgehend von der ersten Nummer des Teilstroms wird jeder Verteilung ein Teilstrom zugewiesen. Dieser Wert wird immer auf 0 gesetzt, da sich dieser Parameter auch durch einen anderen Startwert darstellen läßt. Der genaue Aufbau und die Verteilung der Teilströme auf die Parameter sind in Abschnitt 7.2.1 dargestellt.

Tabelle 7.2: Konfiguration

Typ	Beschreibung
Zuordnung Produkte - Maschinen (w_j^M, w_m^J)	Die Anzahl an Maschinen, auf denen ein Produkt hergestellt werden kann. Jedes Produkt muß auf mindestens einer Maschine und maximal allen Maschinen hergestellt werden können.
Auslastung	Die mittlere Auslastung der Anlagen als Mittelwert über den Planungshorizont und alle Maschinen bezogen auf die verfügbare Kapazität.
Produktionsanteil und Rüstanteil	Bezogen auf die Auslastung kann festgelegt werden, welcher Anteil für die Produktion und für das Rüsten verwendet werden können. Beide Anteile zusammen ergeben 100% der ausgelasteten Kapazität (nicht der verfügbaren Kapazität).
Sicherheitsbestand ($ss_{j,t}$)	Der Sicherheitsbestand wird durch den Anteil der mittleren Periodennachfrage für jedes Produkt berechnet. Ausgehend von einer Standardvorgabe kann die Höhe für jedes Produkt individuell festgelegt werden.
TBO - Kunden	Die TBO gibt die mittlere Zeit zwischen zwei Kundenaufträgen an. Diese Zeit ist die Standardvorgabe für alle Produkte, für die keine separaten Angaben gemacht wurden. Die TBO sollte größer als eins und kleiner als der Planungshorizont sein.
TBO - Produktion	Die TBO gibt die mittlere Zeit zwischen zwei Produktionsauflagen eines Produktes an. Die TBO wird für alle Produkte die keine separaten Angaben machen auf den gleichen Wert gesetzt. Die TBO sollte größer als eins und kleiner als der Planungshorizont sein.

Tabelle 7.3: Kennzahlen

Verteilung	Beschreibung
Nachfrage-charakteristik der Produkte	Jedem Produkt wird eine Nachfragecharakteristik zugewiesen, die als Kennzahl den Variationskoeffizienten benutzt. Die Verteilung der Produkte auf die Charakteristika wird über eine stochastische Verteilung in Form eines Histogramms vorgegeben.
Rüstzeiten für Rüstmatrix	In der Simulation der Feinplanung werden Rüstzeiten für jede Maschine-Produkt-Produkt Kombinationen benötigt. Die Verteilung gibt hier die Art und den Variationskoeffizienten vor, da die genauen Werte unter Berücksichtigung von Nebenbedingungen berechnet werden.
Lagerkosten-satz (h_j)	Die Lagerkostensätze werden für jedes Produkt individuell aus dieser Verteilung gezogen. Die Verteilung gibt sowohl die Art, als auch die genauen Lageparameter vor.
Lageranfangs-bestände ($I_{j,0}$)	Die Höhe der Lageranfangsbestände wird über den Anteil an der mittleren monatlichen Nachfrage ermittelt. Der aus dieser Verteilung für jedes Produkt gezogene Wert entspricht diesem Anteil.
Mittleren Periodennach-frage	Jedes Produkt besitzt eine mittlere Periodennachfrage bezogen auf den gesamten Planungshorizont. Die aus dieser Verteilung für jedes Produkt gezogenen Werte werden unverändert übernommen.
Produktions-rate ($\kappa_{m,j}$)	Für jede Produkt-Maschine Kombination wird aus dieser Verteilung eine Stichprobe gezogen, um anschließend eine Produktionsrate zu berechnen.
Vorlaufzeit der Kunden-nachfrage	Die Differenz zwischen dem Zeitpunkt, an dem der Auftrag im System bekannt wird und dem Zeitpunkt der Lieferung. Die stochastische Verteilung ist für alle Produkte identisch.

Tabelle 7.4: Verteilungen

Vorlage	Beschreibung
Deterministische Nachfrage	Nach Berechnung der Lageparameter wird aus dieser Vorlage einer Verteilung erzeugt, um die deterministische Nachfrage für jedes Produkt zu bestimmen.
Rüstzeiten in der Rüstmatrix	Nach Berechnung der Lageparameter wird aus dieser Vorlage eine Verteilung erzeugt, um für jede Maschine-Produkt-Produkt Kombination eine Rüstzeit zu bestimmen.
Kundennachfrage in der Simulation	Für jeden Kunden wird eine eigene Verteilung nach dieser Vorlage erstellt. Die mittlere Nachfrage entspricht der mittleren deterministischen Nachfrage die für dieses Produkt ermittelt wurde. Die Streuung wird über den Variationskoeffizienten berechnet, der durch die Charakteristik des Produktes gegeben ist.
Vorlaufzeit in der Simulation	Für jeden Kunden wird eine eigene Verteilung nach dieser Vorlage erstellt. Die Lageparameter ergeben sich aus dem vorher berechneten Mittelwert (der aus einer Verteilung gezogen wird) und der Streuung, die über den Variationskoeffizienten ermittelt wird. Es wird angenommen, daß die Vorlaufzeit den gleichen Schwankungen unterliegt wie auch die Nachfrage, so daß der Variationskoeffizient von der gegebenen Charakteristik des Produktes übernommen wird.
Zeit zwischen zwei Kundennachfragen in der Simulation	Der Abstand zwischen zwei Kundennachfragen wird aus der TBO für die Kunden bestimmt. Die Verteilung wird auf diesen Mittelwert hin verschoben und so skaliert, daß der Variationskoeffizient der Charakteristik des Produktes entspricht.

Tabelle 7.5: Vorlagen für stochastische Verteilungen

Merkmale	Beschreibung
Produktionskapazität ($c_{m,t}$)	maschinenspezifisch - innerhalb des Planungshorizonts ist die Kapazität konstant.
Produktionsrate ($\kappa_{m,j}$)	Die Produktionsrate ist definiert durch die Anzahl an Produkteinheiten, die innerhalb einer Zeiteinheit gefertigt werden können.
Rüstzeitenmatrix	Detaillierte Rüstzeiten für jede vorkommende Maschine-Produkt-Produkt Kombination.
Rüstzeiten für CLSP ($st_{m,j}$)	Maximale Rüstzeit für jede Maschine-Produkt Kombination. Wird aus der Rüstzeitenmatrix abgeleitet.
Lagerendbestand (i_j^T)	Der Ziel-Lagerbestand am Ende des Planungshorizonts.
deterministische Nachfrage ($d_{j,t}$)	für jedes Produkt und jede Periode innerhalb des Planungshorizonts.
Rüstkosten ($sc_{m,j}$)	für jede Produkt-Maschine Kombination.

Tabelle 7.6: Abhängige Merkmale

verwendet. Die lose Bindung der Ausprägungen der Merkmale an die Eingabevektoren erzeugt ein Ebenenmodell (vgl. Abschnitt 7.2.1).

Nach der Festlegung aller Merkmale (abhängige und unabhängige) sind noch die Methoden zu bestimmen, nach denen die Ausprägungen der Merkmale erzeugt werden. Die Grundlagen für die Entwicklung der Methoden wurden bereits in Kapitel 3 vorgestellt. Die folgenden Abschnitte stellen weitere darauf aufbauende Verfahren vor, die speziell für diesen Generator entwickelt wurden, in ihrer Anwendung jedoch nicht auf diesen Generator beschränkt sind.

Die Abbildung auf der nächsten Seite zeigt den Ablaufplan des Testdatengenerators. Die Erzeugung der Testdaten ist im Wesentlichen ein linearer Prozeß, in dem zuerst die Ausprägungen der unabhängigen Merkmale festgelegt werden und danach die der abhängigen Merkmale. Innerhalb des Ablaufplans sind Querverweise angebracht, die auf die detaillierte Beschreibung der angewendeten Algorithmen verweisen. Programmschritte ohne Querverweis bestehen nur aus dem Ziehen von Zahlen aus einer Verteilung.

Der Testdatengenerator verwendet den in Abschnitt 2.3 vorgestellten Zufallszahlengenerator von L'Ecuyer. Die Einteilung der Ebenen wurde gegenüber dem Referenzmodell nicht verändert. Den Einsatz und die Abbildung der Merkmale auf die Ebenen wird in Abschnitt 7.2.1 erläutert.

7.2.1 Anwendung des Ebenenmodells

Das in Abschnitt 3.5.3 auf Seite 53 vorgestellte Ebenenmodell für Testdatengeneratoren wird auf den bisher vorgestellten Merkmalen angewendet. Der im Testdatengenerator verwendete Zufallszahlengenerator besitzt in der Referenzimplementierung drei Ebenen, die hier unverändert eingesetzt werden. Weitere Ebenen werden, falls nötig, durch die Aufspaltung von Teilströmen gewonnen.

Ebene 1 enthält die Konfigurationsnummer der Testinstanz. Ihr untergeordnet sind auf Ebene 2 die strukturellen Merkmale. Vor der Erzeugung der Ausprägungen wird jedem Merkmal ein Teilstrom der Ebene 2 fest zugewiesen. Auf der dritten Ebene befindet sich ein verän-

	Rüstanteil ← 100% − Produktionsanteil	
	Zuweisung von Namen für alle Produkte und Maschinen mit Hilfe von „Ziehen ohne Zurücklegen“ (vgl. Abschnitt 3.5.4)	
	Verteilung der Produkte auf die Nachfrageklassen (vgl. Abschnitt 7.2.4)	
	Verteilung der Produkte auf den Maschinen (Abschnitt 7.2.3)	
	Registrierung der nachfolgenden Verteilungen als strukturelle Verteilungen (vgl. Abschnitt 3.5.3 auf Seite 53): mittlere Periodennachfrage, Maschine-Produkt-Produkt Rüstzeiten, Lagerkostensätze, Lageranfangsbestände, mittlere Maschine-Produkt Rüstzeiten, Zuweisung der Nachfragecharakteristik zu den Produkten, Produktionsrate und Rüstzustand.	
	Rüstzustand für jede Maschine aus einer Verteilung ziehen	
	Lagerkostensatz für jedes Produkt mit einer Genauigkeit von einer Nachkommastelle aus einer Verteilung ziehen.	
	Mittlere Periodennachfrage für jedes Produkt aus einer Verteilung ziehen.	
	Rüstkosten berechnen (vgl. Abschnitt 7.2.6)	
	Maximale und verfügbare Kapazität für jede Maschine anhand des Rüstanteils berechnen	
	Erstellung einer Verteilung für jedes Produkt anhand der Nachfragecharakteristik	
	Deterministischen Nachfrage aus dieser Verteilung ziehen	
	Produktionsrate berechnen und an die verfügbare Kapazität anpassen (vgl. Abschnitt 7.2.10)	

Abbildung 7.1: Algorithmus zur Erzeugung der Testdaten

Ebene	Verwendung
1	Konfiguration des Modells
2	Registrierung aller strukturellen Merkmale, Zuweisung von Teilströmen an einzelne Verteilungen
3	Nachfragereihe, veränderliche Merkmale

Tabelle 7.7: Zuweisung der Merkmale im Ebenenmodell

derliches Merkmal, die deterministische Nachfrage. Die strukturellen Merkmale der zweiten Ebene umfassen folgende Verteilungen:

- mittlere Periodennachfrage
- Maschine-Produkt-Produkt Rüstzeiten
- Lagerkostensätze
- Lageranfangsbestände
- mittlere Maschine-Produkt Rüstzeiten
- Zuweisung der Nachfragecharakteristik zu den Produkten
- Produktionsrate und Rüstzustand

Die Merkmale der zweiten Ebene benutzen die ihnen untergeordneten Teilströme der dritten Ebenen zur strukturierten Erzeugung von Teilströmen, indem z. B. jedes Produkt einen separaten Teilstrom der dritten Ebene zugewiesen bekommt. Somit ist es möglich, Testinstanzen mit einer unterschiedlichen Anzahl an Produkten, Maschinen und Perioden zu erzeugen, unter Beibehaltung der Ausprägungen oder der Verhältnisse zwischen den Ausprägungen aller Merkmale.

Diese Verteilung der Merkmale ermöglicht die Erzeugung unterschiedlicher Testdatensätze. Nach Festlegung der ersten Ebenen werden automatisch die Verteilungen auf die zweite Ebene verteilt. Somit ist die Struktur der Anlage gegeben und durch Auswahl von Teilströmen der

dritten Ebene kann die Struktur mit unterschiedlichen Nachfragereihen getestet werden. Zur Untersuchung der Einflüsse von einzelnen Merkmalen (z. B. der Höhe der Sicherheitsbestände) kann eine Testreihe erzeugt werden, in der das zu untersuchende Merkmal verändert wird. Wenn dieses Merkmal unabhängig ist und auch nicht andere Merkmale in ihren Ausprägungen beeinflußt, so ergibt sich für die anderen Merkmale eine identische Konfiguration. Die systematische Untersuchung der Einflüsse dieses Merkmals wird somit schon durch die Konfiguration der Testdaten ermöglicht.

Zur Auswahl einer anderen Konfiguration der Anlagen muß ein neuer Teilstrom der ersten Ebene ausgewählt werden. Die Position der strukturellen Verteilungen in der zweiten Ebene ist während der Registrierung festgelegt worden, so daß diese ihre Position relativ zur ersten Ebene einnehmen. Der dritten Ebene ist ein eigenes Element der zweiten Ebene zugeordnet, welches von keinem anderen Merkmal benutzt werden darf. Dieses aggregierte Merkmal verhindert, daß Merkmale der zweiten Ebene die gleichen Zufallszahlen benutzen wie die Nachfragereihen. Andernfalls würde eine Korrelation zwischen Merkmalen der zweiten und dritten Ebene hergestellt, was negative Auswirkungen auf die Qualität der Daten und Ergebnisse haben kann.

7.2.2 Rollierende Zeitfenster

Für die rollierende Planung werden zwei Zeitfenster erzeugt. Das Erste wird über die Eingabedaten aus einer Datenbank gelegt und bestimmt die Dimensionen und die Lage der Daten für den Jahresplan im Verhältnis zu den von außen vorgegebenen Daten. Das zweite Zeitfenster wird über das Erste gelegt und ergibt die Daten, aus denen der Monatsplan erstellt wird. Die Lage und Ausdehnung des zweiten Zeitfensters kann ebenfalls frei festgelegt werden. Der Aufbau der Zeitfenster wird in Abbildung 7.2 auf der nächsten Seite beschrieben. Die Abbildung zwischen Datenbank ↔ Jahresplan ↔ Monatsplan verwendet folgenden Zusammenhang:

$$t' = \Big(\big(t + T_{Offset} - 1\big) \mod T_{WS}\Big) + 1$$

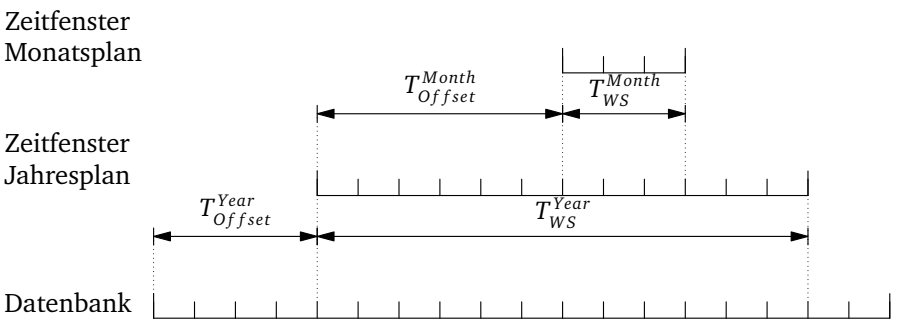


Abbildung 7.2: Aufbau der rollierenden Zeitfenster

t	transformierte Periode
t	Periode
T_{WS}	Größe des Zeitfenster
T_{Offset}	Verschiebung des Zeitfensters

Die Korrekturfaktoren ± 1 sind hier notwendig, um die Abbildung von der 1-basierten Periodeneinteilung t und der 0-basierten Modulo-Operation auf die wiederum 1-basierten Periodeneinteilung t' durchzuführen. Die Abbildung erfolgt von einer Periode t (z. B. eine Periode des Jahresplans) auf eine Periode t' (z. B. ein Eintrag in der Datenbank). Ist die Periode $t > T_{WS}$, so wird über den Modulo-Operator daß das Zeitfenster zyklisch fortgesetzt. Damit ist auch sichergestellt, daß während der Rollierung der Monatsplan am Ende des Jahres Werte erhält, die denen vom Anfang des Jahres - und somit dem Anfang des nächsten Jahres - entsprechen.

Für die Abbildung der Daten im Monatsplan wird die Transformation in zwei Schritten durchgeführt: zuerst von der Datenbank auf das Zeitfenster des Jahresplans und anschließend auf den Monatsplan. Da sich alle Parameter separat einstellen lassen, ist eine sehr flexible Möglichkeit gegeben, unterschiedliche Szenarien zu erstellen.

7.2.3 Verteilung der Produkte auf die Maschinen

Bei der Verteilung der Produkte auf die einzelnen Maschinen wird der Algorithmus auf der folgenden Seite ausgeführt. Die Produkte werden gleichmäßig auf die Maschinen verteilt, bis eine vorgegebenen Dichte d erreicht ist. d entspricht der Anzahl an Maschinen, auf denen ein Produkt p gefertigt werden kann. Durch den Algorithmus ist sichergestellt, daß jedes Produkt auf genau d Maschinen gefertigt werden kann. Problematisch ist, daß die Verteilung der Produkte auf die Maschinen nicht zufällig ist, wie Tabelle 7.8a auf Seite 261 zeigt. Der Algorithmus arbeitet in zyklischer Folge die Positionen in der Liste mit den Maschinen ab. Hierdurch ergibt sich bei gleicher Kombination aus Anzahl an Produkten, Maschinen und Dichte eine identische Zuordnung, da kein Zufallselement vorhanden ist. Eine Möglichkeit zur stochastischen Verteilung ist die Verwendung einer zufällig ermittelten Permutation der Indizes in der Produkt- oder Maschinenliste. Die interne Vergabe der Zuordnungen bleibt erhalten, erst durch die Abbildung auf der Permutation wird ein stochastisches Element indirekt eingeführt.

Ein weiteres Problem stellt die ungleiche Verteilung der Anzahl an Produkten, die auf einer Maschine produziert werden können, dar. Tabelle 7.8b auf Seite 261 ist das Ergebnis der Verteilung von Tabelle 7.8a aus Sicht der Maschinen. Maschine 1 wird mit vier Produkten belastet, während die Maschinen 2 und 3 nur mit jeweils drei Produkten belastet werden.

Für das Ergebnis der Verteilung durch den eingesetzten Algorithmus gilt folgender Satz:

Werden n Produkte auf $m \leq n$ Maschinen verteilt, so daß jedes Produkt auf genau auf $d \leq m$ Maschinen produziert werden kann, ergibt sich folgende Verteilung der Produkte:

- $m - ((n \cdot d) \mod m)$ Maschinen können $\lfloor \frac{n \cdot d}{m} \rfloor$ Produkte herstellen.
- $(n \cdot d) \mod m$ Maschinen können $\lfloor \frac{n \cdot d}{m} \rfloor + 1$ herstellen.

Ein Beispiel findet sich ebenfalls in Tabelle 7.8b auf Seite 261. Die Anzahl der Produkte (A bis E) beträgt $n = 5$, die Anzahl an Maschinen

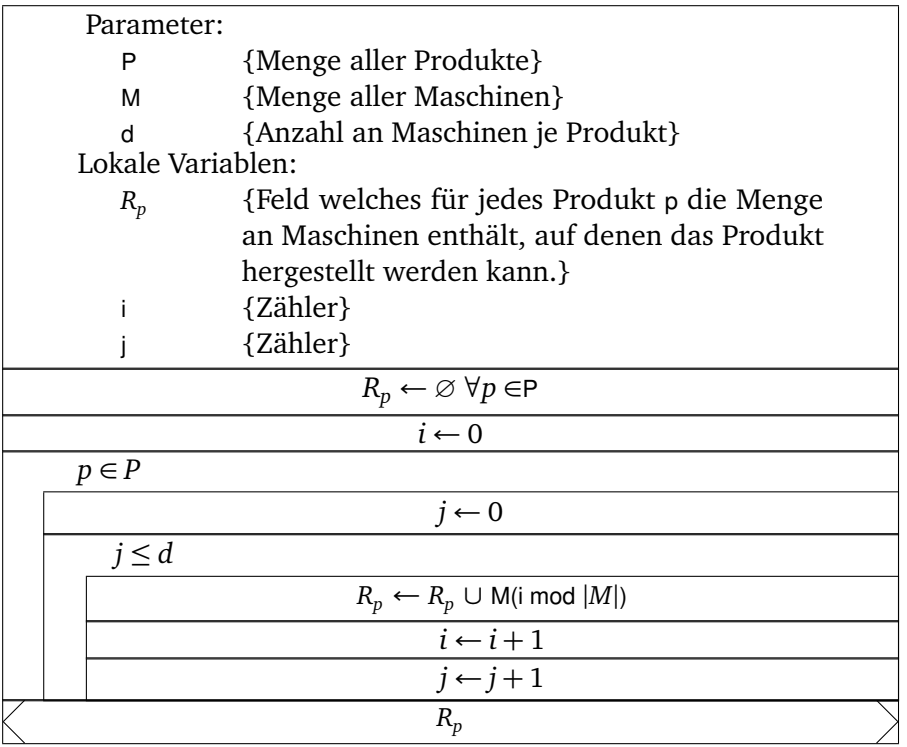


Abbildung 7.3: Algorithmus zur Verteilung von Produkten auf Maschinen

(1 bis 3) beträgt $m = 3$. Um eine Dichte von $d = 2$ zu erreichen müssen $3 - ((5 \cdot 2) \mod 3) = 2$ Maschinen $\lfloor \frac{5 \cdot 2}{3} \rfloor = 3$ Produkte herstellen und eine Maschine muß 4 Produkte herstellen.

7.2.4 Verteilung der Produkte auf die Nachfrageklassen

Zur Darstellung unterschiedlicher Produktcharakteristika wurden Nachfrageklassen erstellt, denen die Produkte zugewiesen werden. Die Zuweisung erfolgt nicht durch eine feste Zuordnung, sondern durch die

Produkt	Maschinen	
A	1	2
B	3	1
C	2	3
D	1	2
E	3	1

(a) Zuordnung aus Sicht der Produkte

Maschine	Produkte			
1	A	B	D	E
2	A	C	D	
3	B	C	E	

(b) Zuordnung aus Sicht der Maschinen

Tabelle 7.8: Aufteilung der Produkte auf die Maschinen

Vorgabe einer Gewichtung. Diese drückt die Häufigkeit aus, mit der ein Produkt dieser Nachfrageklasse vorkommt.

Als Eingabedaten gibt der Benutzer die Gewichtung der einzelnen Nachfrageklassen vor. Aus der Gewichtung erstellt der Testdatengenerator eine diskrete Verteilung. Zu dieser Häufigkeitsverteilung gibt es auch eine Verteilungsfunktion, welche eine eindeutige Umkehrfunktion besitzt. Für jedes Produkt wird anschließend aus einer Gleichverteilung eine Zufallszahl gezogen, die mit Hilfe der Umkehrfunktion auf einer Nachfrageklasse abgebildet wird.

Jeder Nachfrageklasse ist ein Variationskoeffizient zugewiesen. Dieser dient als charakteristisches Merkmal der Nachfrage, so daß unterschiedliche Verhalten simuliert werden können. Die Verwendung des Variationskoeffizienten bietet sich an, da er im Gegensatz zur Varianz unabhängig vom Mittelwert ist und auch bei Transformationen, wie sie in Abschnitt 3.5.5 auf Seite 55 vorgestellt werden, erhalten bleibt.

7.2.5 Lagerkostensatz

Jedes Produkt bekommt einen Lagerkostensatz (in Geldeinheiten je Mengeneinheit und Periode) zugewiesen, die aus einer Verteilung gezogen werden. Der Lagerkostensatz ist der zweiten Ebene zugeordnet, benutzt jedoch die dritte Ebene für die Zuordnung von Teilströmen auf

einzelne Produkte. Der Lagerkostensatz ist somit invariant gegenüber Änderungen der Produktanzahl.

7.2.6 Rüstkosten

Die Rüstkosten sind fixe Kosten, die bei jedem Rüstvorgang anfallen. Sie werden für jedes Produkt separat bestimmt und sind unabhängig von der Zeit. Für die Berechnung der Rüstkosten wird von einem sägezahnartigen Bestandsverlauf ausgegangen, sowie von einer konstanten Nachfragerate. Weiterhin wird angenommen, daß die Zeit zwischen zwei Produktionsaufträge vorgegeben ist. Unter diesen Voraussetzungen läßt sich die EOQ-Formel anwenden (vgl. Harris 1913, Harris 1990, Silver u. a. 1998, S. 150). Zur Anwendung der TBO ist eine modifizierte Form einzusetzen (Silver u. a. 1998, S. 155):

$$T_{EOQ} = TBO = \sqrt{\frac{2 \cdot A}{h \cdot D}}$$

Durch Umformung ergibt sich

$$A = TBO^2 \cdot \frac{h \cdot D}{2}$$

A	Rüstkosten
D	Nachfragerate in Mengeneinheiten je Zeiteinheit
h	Lagerkostensatz je Mengen- und Zeiteinheit
TBO	Time Between Orders = Zeit zwischen zwei Produktionsaufträgen

Der Lagerkostensatz h, TBO und mittlere Periodennachfrage D sind bekannt, so daß die Rüstkosten A hieraus berechnet werden können. Die gleichen Zusammenhänge werden auch von Reith-Ahlemeier (2002, S. 184) zur Berechnung von Bestellfixkosten benutzt.

7.2.7 Deterministische Nachfrage

Die Aufteilung der unterschiedlichen Verteilungen und Stichproben auf das Ebenenmodell ist komplexer, da die deterministische Nachfrage als veränderliches Merkmal der dritten Ebene zugeordnet ist. Die erste Ebene bestimmt die Auswahl einer Konfiguration, die zweite Ebene enthält die Verteilungen der strukturellen Merkmale (vgl. Abschnitt 7.2.1). Die mittlere Periodennachfrage wird dieser Ebene zugewiesen, so daß sie sich nur zusammen mit der Konfiguration ändert. Die darunterliegende dritte Ebene wird eingesetzt, um jedem Produkt einen eigenen Teilstrom zuzuweisen. Die mittlere Nachfrage ist somit invariant gegenüber Änderungen der Produktanzahl. Ebenfalls auf der zweiten Ebene wird eine leere Vorlage registriert, die als Platzhalter für die zu erstellende Verteilung dient. Technisch wird dies über ein Null-Pattern gelöst (Fowler 1999, S. 260), daß als Stellvertreter für die später zu erstellende tatsächliche Verteilung dient.

Die Verteilung der Produkte auf die Nachfragecharakteristik ist bereits erfolgt, so daß für jedes Produkt ein Mittelwert sowie eine Varianz bekannt ist. Die Art der Verteilung, aus der die Nachfrage generiert werden soll, ist von außen durch den Benutzer über eine Vorlage gegeben. Aus diesen Daten wird eine stochastische Verteilung auf der dritten Ebene erzeugt. Die Position in der zweiten Ebene bestimmt eine leere Vorlage, die der dritten Ebene das Produkt.

Da die Nachfrage als veränderliches Merkmal der dritten Ebene zugewiesen wurde, muß diese weiter aufgeteilt werden, um jedem Produkt einen eigenen Teilstrom zuzuweisen. Durch die Zuweisung der unabhängigen Teilströme ist die Nachfrage invariant gegenüber Änderungen an der Produktanzahl oder der Länge des Planungshorizonts. Zur Anwendung kommt das in Abschnitt 3.5.2 vorgestellte Verfahren. Die Dimensionen der Matrix entsprechen der durch die Konstruktion des Testdatengenerators vorgegebenen maximalen Anzahl an Produkten sowie der sich hieraus ergebenden maximalen Länge bezogen auf die Länge eines Teilstroms der dritten Ebene. Für jedes Produkt steht daher ein Teilstrom der Länge $\frac{2^{25}}{50} \approx 671088$ zur Verfügung. Diese Länge ist ausreichend groß für einen Testdatengenerator, auch wenn Accep-

tance-Rejection-Verfahren (wie sie für die Gamma-Verteilung eingesetzt werden) mehrere Elemente aus dem Zufallszahlenstrom zur Erzeugung einer Zufallszahl benötigen. Die Invarianz gegenüber einer Änderung der Periodenlänge ergibt sich durch die sequentielle Anwendung der Verteilung auf das Periodenraster. Eine Verlängerung oder Verkürzung hat keine Auswirkungen auf vorher gezogene Zahlen des gleichen Teilstroms.

Die Konstruktion besitzt mehrere Vorteile: Die Länge des Planungshorizonts und die Anzahl der Produkte lassen sich unabhängig voneinander verändern, ohne daß die erzeugten Nachfragereihen beeinflußt werden. Für systematische Untersuchungen kann die Problemstruktur erhalten bleiben, und die zu beobachtenden Effekte lassen sich einzelnen Merkmalen besser zuweisen.

7.2.8 Stochastische Nachfrage

Außer der für das Modell benötigten deterministischen Nachfrage wird für die Simulation auch eine stochastische Nachfrage erzeugt. Die Anforderungen und die Einbettung innerhalb der hierarchischen Planung wurden bereits in Abschnitt 6.3.6 beschrieben. Die Erzeugung der Daten erfolgt auf Basis der deterministischen Nachfrage. Es wird davon ausgegangen, daß die Mittelwerte der deterministischen und stochastischen Nachfrage übereinstimmen.

Für jedes Produkt wird ein Kundenobjekt erzeugt, daß nur dieses Produkt nachfragt. Das Kundenobjekt ist durch drei Eigenschaften charakterisiert: Höhe der Nachfrage, Abstand zwischen zwei Nachfrageereignissen und Vorlaufzeit zur Bekanntgabe der Nachfrage im System (vgl. Abschnitt 6.3.6). Alle drei Eigenschaften werden als stochastische Verteilungen modelliert. Die Verteilung der Nachfragehöhe wird aus der deterministischen Nachfrage abgeleitet. Die Art der Verteilung wählt der Benutzer aus. Die Variationskoeffizienten stimmen mit der Verteilung überein, die für die Erzeugung der deterministischen Nachfrage verwendet wurden.

Die Verteilung der Zwischenankunftszeiten wird abgeleitet aus der mittleren stochastischen Nachfrage und einer Vorgabe durch den Be-

nutzer. Die Werte werden so angepaßt, daß die sich ergebende mittlere stochastische Nachfrage mit der mittleren deterministischen Nachfrage übereinstimmt. Die Verteilung der Vorlaufzeit ist für alle Produkte identisch und wird anhand der Vorgaben durch den Benutzer modelliert.

Alle Veränderungen, die an den Lageparametern vorgenommen werden, behalten den Variationskoeffizienten der gegebenen Verteilung bei (vgl. Abschnitt 3.5.5 auf Seite 55).

7.2.9 Lageranfangs- und -endbestand

Der Lageranfangsbestand ist ein Merkmal der zweiten Ebene, dessen Verteilung vom Benutzer festgelegt wird. Für jedes Produkt wird ein eigener Teilstrom der dritten Ebene verwendet, so daß auch der Lageranfangsbestand gegenüber Änderungen der Produktanzahl invariant ist. Der Endbestand wird hingegen aus dem Anfangsbestand und weiteren Daten bestimmt.

Für die Berechnung wird ein idealtypischer Verlauf des Lagerbestandes unterstellt. Dieser hat die Form einer Sägezahnkurve, wie sie Abbildung 7.4 auf der folgenden Seite zeigt. Es gelten die Prämissen des EOQ-Modells (Silver u. a. 1998, S. 150), die hier nur kurz zusammengefaßt werden:

- Der Sicherheitsbestand wird während des gesamten Planungshorizonts nicht unterschritten und ist somit nicht entscheidungsrelevant.
- Die Nachfrage erfolgt mit einer konstanten Rate kontinuierlich während des gesamten Planungshorizonts.
- Jedes gefertigte Los hat die gleiche Losgröße q_{opt} und trifft regelmäßig im Abstand t_{opt} ein.
- Jede Lieferung erfolgt zeitpunktgeballt.
- Die Größe der Lieferung kann beliebig teilbar sein.
- Lager- und Rüstkosten sind zeit- und mengeninvariant.

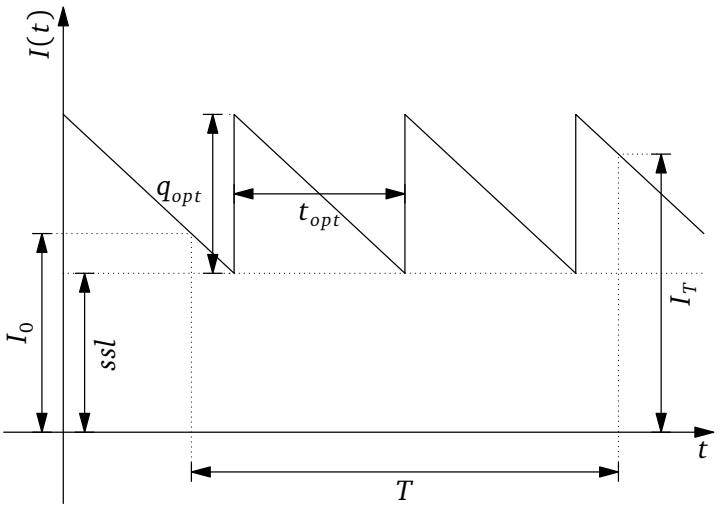


Abbildung 7.4: Idealisierter Lagerbestandsverlauf

Zur Bestimmung des Lageranfangsbestandes muß aus einer vom Benutzer vorgegebenen Verteilung eine $[0;1]$ verteilte Zahl gezogen werden. Diese gibt den Bruchteil von q_{opt} (optimale Losgröße, entspricht EOQ) an, der im Lager zusätzlich zum Sicherheitsbestand vorhanden ist. Damit ist gleichzeitig die Position der Sägezahnkurve aus Abbildung 7.4 festgelegt. Falls ein Lagerendbestand vorgegeben werden soll, kann dieser ebenfalls mit Hilfe der Sägezahnkurve aus Formel (7.1) ermittelt werden.

$$I_T = frac\left(\frac{T - \frac{I_0 - ssl}{\overline{d}}}{t_{opt}}\right) \cdot q_{opt} + ssl \tag{7.1}$$

Formel (7.1) korrigiert den Lageranfangsbestand um den Sicherheitsbestand *ssl* und ermittelt mit Hilfe der mittleren Nachfrage $\overline{d}$ den Zeitpunkt, an dem der Lagerbestand den Sicherheitsbestand erreicht. Die Differenz zum Planungshorizont, bezogen auf die Reichweite t_{opt} eines Loses der Größe q_{opt}, ergibt den Verbrauch als Vielfaches der

optimalen Losgröße. Die Anwendung der Funktion *frac* (welche den nicht-ganzzahligen Teil einer reellen Zahl berechnet) berechnet den prozentualen Anteil der optimalen Losgröße q_{opt}, der am Ende des Planungshorizonts noch im Lager vorhanden sein soll. Da der Lagerendbestand über dem Sicherheitsbestand *ssl* liegen muß, erfolgt noch eine Korrektur.

7.2.10 Produktionsrate

Zur Berechnung der Produktionsraten und der Rüstzeiten ist die Anwendung mehrerer der bisher vorgestellten Verfahren notwendig. Da sowohl die Nachfrage, als auch die verfügbare Kapazität als Eingabeparameter gegeben ist, muß die Produktionsrate gemeinsam mit der Rüstzeit so angepaßt werden, daß die Lösbarkeit des Modells gewährleistet ist. Die hierfür verantwortliche Nebenbedingung (7.2) (diese entspricht Nebenbedingung 6.6 auf Seite 212) beschränkt die Produktionsmenge und die Rüstzeit auf die zur Verfügung stehende Kapazität.

$$\sum_{j \in w_m^J} \frac{X_{m,j,t}}{\kappa_{m,j}} + \sum_{j \in w_m^J} st_{m,j} \cdot Y_{m,j,t} \leq c_{m,t} \qquad \forall \begin{matrix} m = 1 \ldots M, \\ t = 1 \ldots T \end{matrix} \qquad (7.2)$$

Die Kapazität wird von der Produktion und von den Rüstvorgängen verbraucht. Der Anteil der Produktion, der Rüstanteil und die zur Verfügung stehende Kapazität werden als Eingabeparameter festgelegt. Diese beziehen sich jedoch auf die aggregierte Größe der mittleren Kapazität pro Periode innerhalb des Planungshorizonts. Da auch keine Aussage über die Produktionsmenge $X_{m,j,t}$ getroffen werden kann, muß stellvertretend eine aggregierte Größe verwendet werden. Die Aggregation erstreckt sich über die Summe der Kapazität und Nachfrage des gesamten Planungshorizonts.

$$\sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{m\in w_j^M}\frac{X_{m,j,t}}{\kappa_{m,j}}$$

$$+\sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{m\in w_j^M} st_{m,j}\cdot Y_{m,j,t} \leq r_{load}\cdot\sum_{t=1}^{T}\sum_{m\in w_j^M} c_{m,t} \tag{7.3}$$

Aus der Beziehung $r_{prod}+r_{su}=1$ folgt

$$\sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{m\in w_j^M}\frac{X_{m,j,t}}{\kappa_{m,j}} \leq r_{load}\cdot r_{prod}\cdot\sum_{t=1}^{T}\sum_{m\in w_j^M} c_{m,t} \tag{7.4}$$

$$\sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{m\in w_j^M} st_{m,j}\cdot Y_{m,j,t} \leq r_{load}\cdot r_{su}\cdot\sum_{t=1}^{T}\sum_{m\in w_j^M} c_{m,t} \tag{7.5}$$

Die Berechnung der Rüstzeiten kann ausgehend von Formel (7.5) direkt mit Hilfe der in Abschnitt 3.5.5 auf Seite 57 vorgestellten Methoden erfolgen. Die Annahmen sowie Sonderfälle, die sich aus der Aggregation ergeben, werden in Abschnitt 7.2.11 vorgestellt.

Zunächst wird der Produktionsanteil in Gleichung (7.4) mit der Produktionsmenge $X_{m,j,t}$ und der Produktionsrate $\kappa_{m,j}$ betrachtet. Die Produktionsraten sind der Teil, der an die anderen Daten so angepaßt werden soll, daß eine zulässige Lösung entsteht. Für die Produktionsmenge $X_{m,j,t}$ muß eine Annahme getroffen werden: Es wird davon ausgegangen, daß während des Planungszeitraums nur soviel produziert werden soll, wie auch nachgefragt wird. Zusätzlich zur Nachfrage muß auch die Differenz zwischen dem Lagerbestand zu Beginn und am Ende des Planungshorizonts berücksichtigt werden. Es ergibt sich folgende Beziehung zwischen der Gesamtnachfrage und Produktionsmenge über den gesamten Planungshorizont:

$$\sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{m\in w_j^M} X_{m,j,t} = \sum_{t=1}^{T}\sum_{j=1}^{J} d_{j,t} + \sum_{j=1}^{J}(i_j^T - I_{j,0}) \tag{7.6}$$

für jedes Produkt gilt somit

$$\sum_{t=1}^{T}\sum_{m\in w_j^M} X_{m,j,t} = \sum_{t=1}^{T} d_{j,t} + (i_j^T - I_{j,0}) \qquad \forall\, j = 1\ldots J \tag{7.7}$$

Unter der Annahme, daß jedes Produkt auf jeder hierfür geeigneten Maschine in jeder Periode in der gleichen Menge $\overline{X_j}$ hergestellt wird, ergibt sich:

$$T \cdot |w_m^J| \cdot \overline{X_j} = \sum_{t=1}^{T} d_{j,t} + (i_j^T - I_{j,0}) \qquad \forall\, j = 1\ldots J \tag{7.8}$$

$$\overline{X_j} = \frac{1}{|w_m^J|}\left(\frac{\sum_{t=1}^{T} d_{j,t}}{T} + \frac{1}{T}\cdot(i_j^T - I_{j,0})\right) \qquad \forall\, j = 1\ldots J \tag{7.9}$$

Wird die mittlere Periodennachfrage ersetzt durch den Erwartungswert für die mittlere Nachfrage $\overline{d_j}$, so ergibt sich

$$\overline{X_j} = \frac{1}{|w_m^J|}\left(\overline{d_j} + \frac{1}{T}\cdot(i_j^T - I_{j,0})\right) \qquad \forall\, j = 1\ldots J \tag{7.10}$$

Wird dieser Term in Formel (7.4) für $X_{m,j,t}$ eingesetzt, so müssen nur noch die $\kappa_{m,j}$ so bestimmt werden, daß die Ungleichung erfüllt ist. Ein entsprechendes Verfahren wurde bereits in Abschnitt 3.5.5 auf Seite 57 vorgestellt. Die Werte für die Produktionskoeffizienten vor der Skalierung werden aus einer Verteilung gezogen, deren Parameter von außen vorgegeben sind. Nach der Skalierung bleiben die Form der Verteilung und der Variationskoeffizient erhalten.

Problematisch bei dieser letzten Ersetzung ist, daß der Erwartungswert für die mittlere Nachfrage nicht identisch ist zum gemessenen Mittelwert der Nachfrage. Diese stimmen nur ungefähr überein, in Abhängigkeit von der Länge des Planungshorizonts, d. h. der Anzahl der Stichproben aus der stochastischen Verteilung. Wenn - wie in diesem Fall - Backlog zugelassen sind, stellt dies kein Problem dar, da nicht erfüllte Nachfrage zurückgestellt werden kann. Jedoch nimmt die Qualität des Datensatzes ab, da das Modell gar nicht die Kapazität hat, die Nachfrage vollständig zu erfüllen. Bei einer ausreichenden Modellgröße ist der Fehler gering, so daß die Abweichungen zwischen den einzelnen Produkten sich gegenseitig kompensieren. Solange die Auslastung der Maschinen unter 100 % liegt, kann die Abweichung ebenfalls kompensiert werden.

Eine Alternative zu dieser Approximation ist die Verwendung der exakten mittleren Nachfrage, berechnet aus der Nachfragereihe. Da die Nachfragereihe aber ein Element der dritten Ebene ist, hat jede Änderung direkte Auswirkungen auf die Produktionskoeffizienten, die Merkmale der zweiten Ebene. Aufgrund der Konstruktion des Testdatengenerators ist dies jedoch nicht gewünscht.

7.2.11 Rüstzeiten

Die Berechnung der Rüstzeit verläuft analog zur Berechnung der Produktionskoeffizienten. Da die Rüstzeiten einen mit den Produktionskoeffizienten vergleichbaren Einfluß auf die Lösbarkeit der Jahres- und Monatspläne sowie auf die Feinplanung besitzen, muß bei der Berechnung darauf geachtet werden, daß die Daten für alle Planungsebenen eine zulässige Lösung erzeugen. Die Produktionskoeffizienten wurden nur unter Berücksichtigung der Kapazität erzeugt, da die Reihenfolge nicht im Jahres- und Monatsplan abgebildet wird. Somit müssen die Rüstzeiten zur Berücksichtigung der verfügbaren Kapazität auch die mögliche Reihenfolge der Lose auf der Feinplanungsebene berücksichtigen, damit eine zulässige Lösung auf dieser Ebene erzeugt werden kann.

Die Berechnung der Rüstzeiten erfolgt in mehreren Stufen. Im ersten Schritt wird die Rüstmatrix für jede vorkommende Maschine-Produkt-Produkt Kombination mit den Stichproben aus der vorgegebenen Verteilung belegt. Die Rüstzeiten sind ein Merkmal der zweiten Ebene, so daß die dazugehörige dritte Ebene für die Verteilung benutzt werden kann. Jede Maschine bekommt einen eigenen Teilstrom der dritten Ebene, der weiter aufgeteilt wird, indem die maximale Anzahl an Produkten, die vom Testdatengenerator erzeugt werden kann, der Dimension der Produkt-Produkt Matrix gleichgesetzt wird. Auf die entstandene Matrix kann das in Abschnitt 3.5.2 auf Seite 48 vorgestellte Verfahren anwendet werden. Die durch dieses Verfahren erzeugten Rüstzeiten sind invariant gegenüber Änderungen der Anzahl von Produkten oder Maschinen.

Aus dieser nicht normierten Matrix wird für jede Produkt-Maschine Kombination die maximale Rüstzeit für das Umrüsten auf ein Produkt ermittelt. Die maximale Rüstzeit entspricht der Zeit, die im ungünstigsten Fall auf der Feinplanungsebene auftreten kann. Dieser Fall kann z. B. eintreten, wenn in einer Periode nur zwei Lose produziert werden, deren Reihenfolge zu diesem Extrempunkt führt. Es wird somit angenommen, daß in jeder Periode für jedes Los, welches auf einer Maschine hergestellt wird, der ungünstigste Fall eintritt. Die Summe der sich hieraus ergebenden verbrauchten Kapazität muß summiert über alle Maschinen dem zugewiesenen Rüstanteil bezogen auf die verfügbare Kapazität entsprechen $r_{load} \cdot r_{su} \cdot \sum_{t=1}^{T} \sum_{m \in w_j^M} c_{m,t}$. Für den Fall, daß Mikroperioden existieren, muß die Kapazität gleichmäßig auf die Mikroperioden innerhalb einer Makroperiode aufgeteilt werden. Es wird angenommen, daß in jeder Mikroperiode jedes Produkt auf jeder Maschine einmal produziert wird. Für den hieraus folgenden Rüstvorgang muß ebenfalls die maximale Rüstzeit benutzt werden.

Das Verhältnis zwischen der verfügbaren Kapazität und dem Kapazitätsverbrauch der nicht normierten Matrix ergibt den Normierungsfaktor, mit dem die gesamte Rüstmatrix zu multiplizieren ist, um den vorgegebenen Rüstanteil zu erfüllen. Die Rüstzeiten für den Jahres- und Monatsplan werden aus der normierten Rüstmatrix ermittelt. Hierfür

werden wie bei der Normierung die maximalen Rüstzeiten für jede Produkt-Maschine Kombination ausgewählt. Nur so ist es möglich, daß der Jahres- und Monatsplan Vorgaben erstellen, die von der Feinplanungsebene auch umsetzbar sind.

Die vorgestellte Normierung und Auswahl der maximalen Rüstzeit ist das einzige Verfahren, daß sicherstellt, daß jeder Plan, der auf der mittelfristiger Ebene erstellt wurde, auch realisierbar ist. Problematisch ist die systematische Unterschätzung der verfügbaren Kapazität. Da für das Rüsten die maximale Zeit angenommen wird, kann die mittlere Rüstzeit - die auf operativer Ebene als Planungsergebnis entsteht - geringer sein. Somit wird Kapazität frei und die mittlere Auslastung sinkt unter den vorgegebenen Wert. Dieser strukturelle Fehler ist unvermeidbar, wenn die Pläne unmittelbar an die Feinplanungsebene übergeben werden. Soll hingegen ausschließlich ein Modell vom Typ CLSP betrachtet werden, so kann die durchschnittliche Rüstzeit aus der Feinplanung übernommen werden, da das CLSP keine Reihenfolgeplanung durchführt.

7.2.12 Abstimmung der Parameter

Bei der Erzeugung von Testdaten gibt es zwei Parametergruppen, die einen wesentlichen Einfluß auf die Qualität der Lösung haben. Werden diese bei der Konstruktion der Daten nicht berücksichtigt, treten Abbildungsfehler auf, die ihre Ursache in der Abstimmung der unterschiedlichen Zeitleisten haben. Weitere Auswirkungen ergeben sich für die hierarchische Planung, wenn die Daten auf mehreren Ebenen verwendet werden. Wird nur eine Ebene betrachtet, entfallen alle Probleme, die aufgrund der ersten Parametergruppe auftreten, so daß nur die zweite Gruppe beachtet werden muß.

- TBO und Synchronisation der Ereignisse
  - der mittlere Abstand zwischen zwei Bestellungen
  - der Abstand zwischen zwei Produktionsaufträgen = Anzahl der Losauflagen

  - die Anzahl an Mikroperioden

- Parameter der Lösungsstruktur

  - Lageranfangsbestand, Lagerendbestand
  - der Abstand zwischen zwei Produktionsaufträgen
  - freie Kapazität

Die erste Parametergruppe betrifft die Synchronisation der Losauflagen zwischen der deterministischen und stochastischen Nachfrage sowie dem auf mittelfristiger Ebene verwendeten Modell (Production Planning). Da das eingesetzte CLSP nur einen Rüstvorgang pro Periode erlaubt, entspricht die Anzahl an Mikroperioden der maximalen Anzahl an Losauflagen innerhalb einer Periode. Wenn diese zwischen den einzelnen Produkten wenig variiert, ist dies unproblematisch, da die unterschiedlichen Losauflagen innerhalb einer Periode sich auf die Mikroperioden verteilen. Liegen die Werte hingegen weit auseinander, so kann es bei der Erzeugung der Daten zu einer Unterschätzung der verfügbaren Kapazität kommen, da die Rüstzeiten zu hoch angesetzt werden.

Die Anzahl an Mikroperioden begrenzt die Anzahl an Losauflagen im Modell, die nicht überschritten werden darf. Andernfalls kommt es zu teilweise gravierenden Abbildungsfehlern der Testdaten im CLSP. Wird sie dennoch überschritten, steigt die Losgröße und die Anzahl an Rüstvorgängen sinkt. Mit sinkender Anzahl an Losauflagen kommt es zu Problemen bei der Erfüllung der stochastischen Kundennachfrage. Liegt die Anzahl an Nachfrageereignissen über der Anzahl an Losauflagen, so kann bei einer geschlossenen Produktion ein Teil der Nachfrage erst nach Abschluß eines (zu großen) Loses erfüllt werden. Die Häufigkeit von Nachlieferungen nimmt somit zu und der gemessenen Servicegrad sinkt.

Für die erste Parametergruppe gelten folgende Empfehlungen:

- Die Anzahl an Mikroperioden darf nicht wesentlich von der maximalen Anzahl an Losauflagen abweichen.

- Die Zwischenankunftszeit zwischen zwei Kundenaufträgen darf nicht länger als eine Periode sein, da sonst Abbildungsfehler bei der Erzeugung der Testdaten sowie bei der Planung entstehen.
- Die vom Testdatengenerator erzeugte mittlere Losgröße für ein Produkt darf nicht mehr als die Hälfte der verfügbaren Kapazität einer Mikroperiode verbrauchen, da sonst die Losgrößen der anderen Produkte stark beeinträchtigt werden.

Die zweite Parametergruppe betrifft die Lösungsstruktur der Lagerbestände. Durch die Vorgabe von Anfangs- und Endbeständen und deren Berechnung mit Hilfe der EOQ-Formel wird angenommen, daß die Losgröße der EOQ entspricht. Dieser Wert wird vom Modell auch näherungsweise gewählt, wenn ausreichend Kapazitäten vorhanden sind. Die Lösungsstruktur ist somit schon durch die Daten vorgegeben. Abweichungen von dieser Struktur sind erst mit steigender Auslastung möglich. Diese Aspekte müssen bei der Konstruktion von Testdatensätzen mit geringer Kapazitätsauslastung beachtet werden.

7.3 Ausblick

Der hier vorgestellte Testdatengenerator bietet im Gegensatz zu den in der Literatur verwendeten Generatoren die Möglichkeit, durch die Einstellung von Kennzahlen einen Testdatensatz zu erzeugen. Durch die hohe Anzahl an Einstellungen lassen sich viele stark unterschiedliche Situationen abbilden und testen. Aufgrund einiger Designschwächen können sich in Grenzsituationen starke Differenzen zwischen den eingestellten und später gemessenen Kennzahlen ergeben. Im Folgenden sollen die Problemfelder noch einmal kurz zusammengefaßt werden, um einen Ausgangspunkt für zukünftige Weiterentwicklungen zu geben.

Rüstzeiten Die Erzeugung von reihenfolgeabhängigen Rüstzeiten, die sowohl für die mittelfristige Planung, als auch für die Feinplanung eingesetzt werden können, ohne daß unzulässige Pläne entstehen, ist ein kritischer Punkt (vgl. Abschnitt 7.2.11). Je nach Spannweite zwischen der mittleren und höchsten Rüstzeit entsteht ein

Abbildungsfehler, der die Auslastung der Maschinen verzerrt. Wird der Testdatengenerator nur für die lang-/mittelfristige Planung verwendet, darf ausschließlich die mittlere Rüstzeit eingesetzt werden, um Abbildungsfehlern zu vermeiden.
Die Verwendung von mittleren Rüstzeiten innerhalb der rollierenden Planung ist nur zulässig, wenn die Feinplanung eine höhere Flexibilität hat, um die auftretenden längeren Rüstzeiten zu kompensieren. Eine Möglichkeit wäre die Anpassung der Losgrößen und der Einsatz von Planungsverfahren, die Lose auch zwischen den Maschinen austauschen können (vgl. Abschnitt 6.3.5 auf Seite 230).

TBO Durch die Konstruktion des Testdatengenerators wird davon ausgegangen, daß in jeder Periode ein Rüstvorgang stattfindet. Produkte, die eine TBO länger als eine Periode besitzen, wird die Rüstzeit somit überschätzt. Eine Möglichkeit, diesen Abbildungsfehler zu beheben, ist die anteilige Berücksichtigung der Rüstzeiten. Bei einer TBO von 3 Perioden wird die Rüstzeit zu einem Drittel in jeder Periode berücksichtigt. Entsprechend müssen diese Modifikationen auch auf die Rüstzeiten der Feinplanung übertragen werden.

Lageranfangs-/endbestand Der Testdatengenerator erzeugt für jedes Produkt einen individuellen Anfangs- und Endbestand anhand der Produktcharakteristik, die aus der EOQ abgeleitet wird (vgl. Abschnitt 7.2.9). Für Modelle, die Anfangs-/Endbestände nicht abbilden, lassen sich die Testdaten nicht einsetzen, da die Nachfragereihe in der Kapazität nur über durchschnittlich Werte abgebildet wird (vgl. Abschnitt 7.2.7).
Eine Möglichkeit, gültige Daten für diese Modelle zu erzeugen, wird in den Daten von Trigeiro u. a. (1989, S. 356) umgesetzt. Indem ein ansteigender Trend unterstellt wird, kann die Nachfrage in den ersten Perioden reduziert oder auf null gesetzt werden. Die zu Beginn verminderte Nachfrage wird in späteren Perioden aufgeschlagen, um die mittlere Nachfrage konstant zu halten.

Eine andere Möglichkeit wäre, bereits bei der Datenerzeugung mit Hilfe einer Heuristik zu überprüfen, ob es eine gültige Lösung gibt oder die Kapazitäten in den ersten Perioden an die Nachfrage anzupassen und zusätzliche Kapazitäten bereitzustellen. Letzteres entspricht dem Konzept von Trigeiro u. a. übertragen auf Kapazitäten.

Backlog Die Daten wurde unter der Annahme erzeugt, daß Nachlieferungen ohne Einschränkung zulässig sind. Entsprechend wurden für die Nachfrage nur Durchschnittswerte verwendet. Wenn das Modell jedoch keine Nachlieferungen vorsieht, kann es bei einer stark schwankenden Nachfrage dazu kommen, daß nicht ausreichend Kapazität vorhanden ist und die Daten keinen gültigen Produktionsplan zulassen. Das Problem und die möglichen Lösungsalternativen entsprechen denen, die im vorangehenden Absatz zu den Anfangs-/Endbeständen vorgestellt wurden.

Kapitel 8

Analyse der Testreihen

Das folgende Kapitel stellt die Ergebnisse vor, die mit dem Versuchsaufbau von Kapitel 6 berechnet wurden. Die Eingabedaten stammen aus dem in Kapitel 7 vorgestellten Testdatengenerator.

Ausgehend von einer Standardkonfiguration wird eine Referenztestreihe simuliert. Nach Vorstellung der Ergebnisse werden verschiedene Variationen berechnet und die Veränderungen in den Ergebnissen erläutert. Alle Testrechnungen wurden auf einem Rechner mit zwei Intel Xeon Quadcores (d. h. acht Kerne), einer Taktfrequenz von 2,33 GHz und einem Arbeitsspeicher von 16 GB durchgeführt. Als Betriebssystem kam die 64 bit Version von OpenSuSE 11.0 zum Einsatz. Zur Lösung der gemischt-ganzzahligen Modelle wurde die Software Xpress-MP der Firma dash optimization in der Version 2008A.2 verwendet. Alle Java-Programme wurden in einer 64 bit JVM der Sun JDK 1.6.0 (Update 13) ausgeführt. Der Server-Modus war hier aufgrund der Betriebssystemumgebung automatisch eingeschaltet. Die Datenaufzeichnung erfolgte in einer MySQL-Datenbank, Version 5.0.51a, 64 bit, Standardkonfiguration.

Die gemischt-ganzzahligen Modelle wurden in ihrer Laufzeit durch eine Zeitschranke begrenzt. Für Modelle der Jahresplanung beträgt diese 120 Sekunden, für Modelle der Monatsplanung 10 Sekunden. Die Software hat zum Lösen der Modelle alle acht Kerne parallel genutzt,

wobei die Zuweisung der Knoten zu den Threads deterministisch erfolgt und die Auslastung des Gesamtsystems nur bei ca. 70% liegt. Eine nicht deterministische Zuweisung führt zu einer höheren Auslastung, wurde jedoch aufgrund von Instabilitäten nicht eingesetzt.

Für die Auswertung der Testreihen wurden die Daten aus der MySQL-Datenbank auf einen zweiten Rechner dupliziert und über die ODBC-Schnittstelle mit R (R Development Core Team 2008) ausgelesen und verarbeitet. Sämtliche in dieser Arbeit durchgeführten statistischen Signifikanztests wurden mit Hilfe von R, Version 2.9.0 durchgeführt.

8.1 Einsatz des β-Servicegrades als Meßgröße

In Abschnitt 6.1.6 wurde bereits definiert, daß durch den Einsatz einer kontinuierlichen Zeitachse auch alle Kennziffern an diese Zeitachse angepaßt werden müssen. Zur Messung der Lieferfähigkeit eines Systems werden Servicegrade verwendet. In der Literatur werden drei grundlegende Arten von Servicegraden unterschieden: α, β und γ Servicegrad. Der α Servicegrad entspricht dem Anteil der Perioden, an denen die Nachfrage vollständig und unmittelbar geliefert werden konnte.

Der β-Servicegrad gibt an, welcher Anteil der Nachfrage unmittelbar erfüllt werden konnte. Als Letztes erweitert der γ Servicegrad den β-Servicegrad, indem nicht mehr die Fehlmenge, sondern der Fehlbestand verwendet wird. Somit wird auch die zeitliche Entwicklung berücksichtigt. Weitere Leistungskriterien werden detailliert in Tempelmeier (2006, S. 27) vorgestellt.

Im Rahmen dieser Untersuchungen soll der β-Servicegrad verwendet werden, da er in der Praxis weit verbreitet ist. Allgemein wird er wie folgt definiert (Tempelmeier 2006, S. 29, im engl. auch als „fill rate“ bezeichnet, vgl. Silver u. a. 1998, S. 244)

$$\beta = \frac{\text{Anteil der direkt erfüllten Nachfrage}}{\text{Gesamtnachfrage}} \tag{8.1}$$

Aufgrund der Erweiterung der Simulation durch eine kontinuierliche Zeitachse wird auch der β-Servicegrad nicht mehr periodisch,

sondern ereignisbezogen gemessen. Bei jedem Nachfrageereignis wird aufgezeichnet, welcher Anteil der Nachfrage direkt erfüllt werden konnte und welcher nachgeliefert wurde. Es wird davon ausgegangen, daß die Lieferungen beliebig teilbar sind.

Eine wesentliche Voraussetzung ist, daß die Nachfrage zeitpunktgeballt erfolgt. Für eine kontinuierliche Nachfrage (über ein Zeitintervall) ist der β-Servicegrad nicht geeignet. Erst durch den Einsatz des γ-Servicegrades können Kennzahlen unter kontinuierlicher Nachfrage erhoben werden. Der γ-Servicegrad mißt den Fehlbestand, der sich durch Integration der Fehlmengen über die Zeit ergibt.

8.2 Referenztestreihe

Als Ausgangspunkt der Untersuchung wurde eine Standardkonfiguration erstellt. In jeder weiteren Testreihe, erläutert in den folgenden Abschnitten, werden die Ausprägungen eines oder mehrerer Merkmale variiert und die Auswirkungen auf mehrere Meßgrößen vorgestellt.

Konfiguration

In Kapitel 6 wurden bereits die Erzeugnisstruktur sowie die auf den jeweiligen Planungsebenen eingesetzten Planungsalgorithmen und Modelle vorgestellt. Diese und alle folgenden Testreihen verwenden einen gemeinsamen Satz an Merkmalen (Konfiguration), die separat verändert werden, um die Reaktionen des Planungssystems auf die Veränderungen zu untersuchen. Die Konfigurationsdaten werden von einem Testdatengenerator, wie er in Kapitel 7 vorgestellt wurde, verarbeitet und dem Planungssystem als Eingabedaten zur Verfügung gestellt.

Die Untersuchungen beschränken sich auf eine Produktionsstufe mit einer Maschine. Hergestellt werden 10 Produkte, die sich nur in ihrer deterministischen Nachfrage und in einigen Testreihen auch in ihren Lagerkostensätzen unterscheiden. Bei einer geringeren Anzahl an Produkte wäre der Anteil der Kapazität, den ein Produkt belegt,

zu groß. Daraus würden sich Effekte ergeben, die nicht auf andere Problemgrößen übertragbar sind.

Die Kapazitätsdaten, sowie die Aufteilung der Simulationseinheiten auf die Perioden entspricht der Abbildung eines Jahres mit 12 Monaten (Perioden) und jeweils 4 Wochen (Mikroperioden). Jeder Monat besteht aus 30 Tagen (Simulationseinheiten je Periode), jeder Tag aus 24 Stunden (Kapazitätseinheiten je Simulationseinheit). Während der Rollierung wird der Produktionsplan für den ersten Monat festgehalten und an die Feinplanung übergeben. Der Abstand zwischen zwei Planungsläufen beträgt ein Monat, die Feinplanung aktualisiert jeden Tag ihre Pläne - beides entspricht den in der Praxis verwendeten Planungsabläufen. Eine Auslastung der Maschinen von 90% entspricht in etwa einer mittleren Auslastung (vgl. Trigeiro u. a. 1989, S. 362). Die Rüstzeiten werden in allen Testreihen konstant gehalten, um zusätzliche Einflüsse zu vermindern. Dies gilt auch für den Anteil der Rüstzeiten an der verfügbaren Kapazität - dieser ist deutlich geringer als bei Trigeiro u. a. (1989, S. 361).

Die Zeit zwischen zwei Kundenaufträgen wurde auf konstant eine Woche gesetzt, um zusätzliche Einflüsse zu minimieren. Die TBO der Produkte wurde auf den gleichen Wert gesetzt, um Abbildungsfehler durch nicht abgestimmte Produktionszyklen zu vermeiden. Die Höhe der Nachfrage (1000 ME je Periode) wurde zufällig ausgewählt. Der Variationskoeffizient der Nachfrage von 0,1 entspricht einer niedrigen Variabilität (vgl. Trigeiro u. a. 1989, S. 361). In einer späteren Testreihe wird diese auf 0,2 erhöht, was einer mittleren Variabilität entspricht. Der Basiswert der Lagerkostensätze wurde zufällig gewählt. Da alle anderen Kosten (Rüstkosten, Strafkosten) hiervon abgeleitet werden, kann er auch auf jeden anderen Wert gesetzt werden, da er so keinen Einfluß auf die Lösung der Modelle hat. Eine detaillierte Auflistung aller Parameter erfolgt in den Tabellen 8.1 bis 8.3 auf den Seiten 281–282.

Für die Untersuchungen werden die Ergebnisse einer oder mehrerer Testreihen miteinander verglichen. Jede Testreihe besteht aus 100 Testinstanzen, die sich in dem Startpunkt des Zufallszahlengenerators unterscheiden. Dieser wurde, wie in Abschnitt 7.2.1 beschrieben, um jeweils einen Teilstrom der ersten Ebene verschoben. Der Startvektor von

Parameter	Wert
Kapazitätseinheiten je Simulationseinheit	24
Simulationseinheiten je Periode	30
Länge des Planungshorizonts (Jahresplan)	12 Perioden
Länge des Planungshorizonts (Monatsplan)	3 Perioden
Anzahl der fixierten Perioden im Monatsplan	1 Periode
Anzahl der Mikroperioden	4
Auslastung der Maschinen	90%
Anteil der Produktionszeit	95%
Anteil der Rüstzeit	5%

Tabelle 8.1: Kapazitätsparameter der Referenztestreihe

Testinstanz 0 entspricht dem Startvektor des Zufallszahlengenerators in der Referenzimplementierung, wie sie in Abschnitt 7.2.1 beschrieben wurde.

Analyse

Die Durchführung der Simulationen, sowie die Auswahl der Testreihen und Konfigurationen erfolgte unter folgenden Fragestellungen:

- Wie verhalten sich die Servicegrade der Produkte innerhalb einer Testreihe untereinander, sowie im Vergleich zu anderen Testreihen? Worauf lassen sich diese Unterschiede zurückführen?
- Welche Zusammenhänge zwischen den beobachteten Größen lassen sich beobachten und worauf können diese zurückgeführt werden?

In diesem Abschnitt sollen einige Methoden vorgestellt werden, wie die durch die Simulation erhaltenen Daten interpretiert und ausgewertet werden können. Die Vorstellung erfolgt hier detaillierter, so daß später nur noch auf die Ergebnisse verwiesen wird.

Parameter	Wert
Simulationseinheiten zwischen zwei Kundenaufträge	7,5 (=0,25 Perioden)
TBO für alle Produkte	0,25 Perioden
Verteilung der mittleren Nachfrage	konstant bei $\mu = 1000$ ME je Periode
Schwankungen der stochastischen und deterministischen Nachfrage um die mittlere Nachfrage	Gammaverteilung mit einem Mittelwert von $\mu = 1000$ ME je Periode und einem Variationskoeffizienten von V=0,1
Vorlaufzeit der Kundennachfrage	keine

Tabelle 8.2: Nachfrageparameter der Referenztestreihe

Parameter	Wert
Verteilung der Lagerkostensätze	konstant mit $\mu = 16$
Verteilung der mittleren Rüstzeiten	konstant
Verteilung der Rüstzeiten in der Rüstmatrix	konstant
Verteilung des Lageranfangsbestandes	konstant bei 50% der mittleren Periodennachfrage
Verteilung der Produktionsraten	konstant

Tabelle 8.3: Weitere Parameter der Referenztestreihe

Für die Referenztestreihe läßt sich die Antwort auf die erste Frage anhand von Abbildung 8.1 auf der nächsten Seite zeigen. Die Abbildung enthält für jedes Produkt getrennt den Servicegrad in Abhängigkeit von der Nachfrage. Die Nachfrage entspricht der Summe der Kundennachfrage über den Beobachtungszeitraum von einem Jahr. Der Servicegrad wurde berechnet aus der Summe der gesamten Nachfrage und der unmittelbar erfüllten Nachfrage im Beobachtungszeitraum. Jeder Datenpunkt entspricht einer Testinstanz (einem Jahr), so daß für jedes Produkt 100 Datenpunkte gemessen wurden.

Es fällt auf, daß jedes Produkt eine hohe Spannweite an Servicegraden umfaßt und eine Punktwolke bildet, deren Schwerpunkt jeweils in der Mitte liegt. Für die gleiche Nachfrage kann der Servicegrad für ein Produkt zwischen ≈55% und ≈75% schwanken, jeweils bezogen auf den Beobachtungszeitraum von einem Jahr. Die genaue Lage der Punktwolke unterscheidet sich zwischen den Produkten teilweise signifikant. Damit unterscheiden sich auch die erreichten Servicegrade zwischen den Produkten. Abbildung 8.2 auf Seite 285 zeigt einen sog. Boxplot mit dem Mittelwert sowie den 25% und 75% Quantilen und den Ober-/Untergrenzen sowie Ausreißern. Die Unterschiede zwischen den Produkten sind hier deutlicher zu erkennen. Sie unterscheiden sich am stärksten in der Lage, sowie teilweise auch in der Form.

Zur Quantifizierung der Unterschiede können statistische Tests eingesetzt werden. Diese lassen sich unterteilen in Tests, die eine bestimmte Verteilung voraussetzen und verteilungsfreie Tests. Für die Verteilung der β-Servicegrade ist zuerst die Art der Verteilung zu bestimmen. Problematisch ist, daß mit steigendem Sicherheitsbestand auch der Servicegrad steigt und immer mehr Testinstanzen und Produkte einen Servicegrad von 100% erreichen. Die Verteilung wird deformiert und die Annahme der Verteilung kann für den unbereinigten Datensatz nicht mehr aufrecht erhalten werden. Daher wird auf verteilungsfreie Tests zurückgegriffen, hier der Wilcoxon-Rangsummentest (vgl. Hartung 2002, S. 513 und Duller 2008, S. 165). Dieser vergleicht die Lage zweier Verteilungen. Voraussetzungen für die Anwendung des Tests sind zwei Stichproben, die ordinale oder metrische skalierte Merkmale enthalten. Beide Stichproben müssen unabhängig voneinander sein und

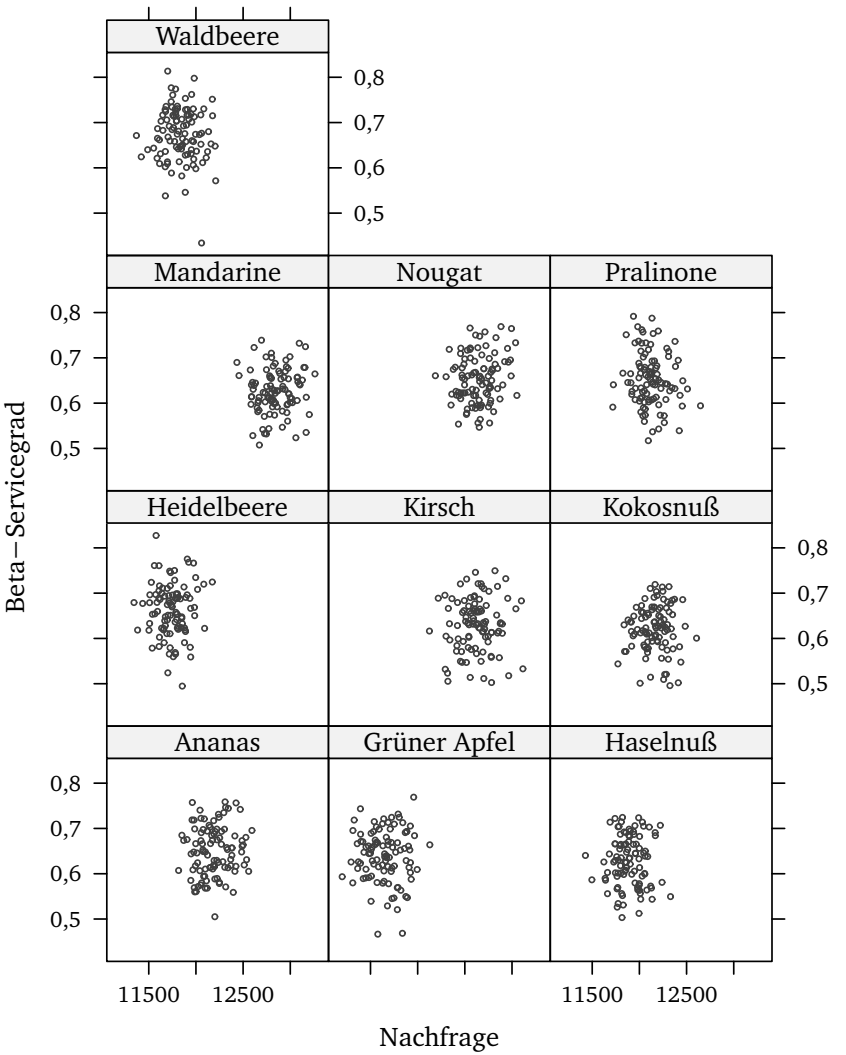


Abbildung 8.1: Vergleich zwischen Nachfrage und β-Servicegrad (geschlossene Produktion, Referenztestreihe)

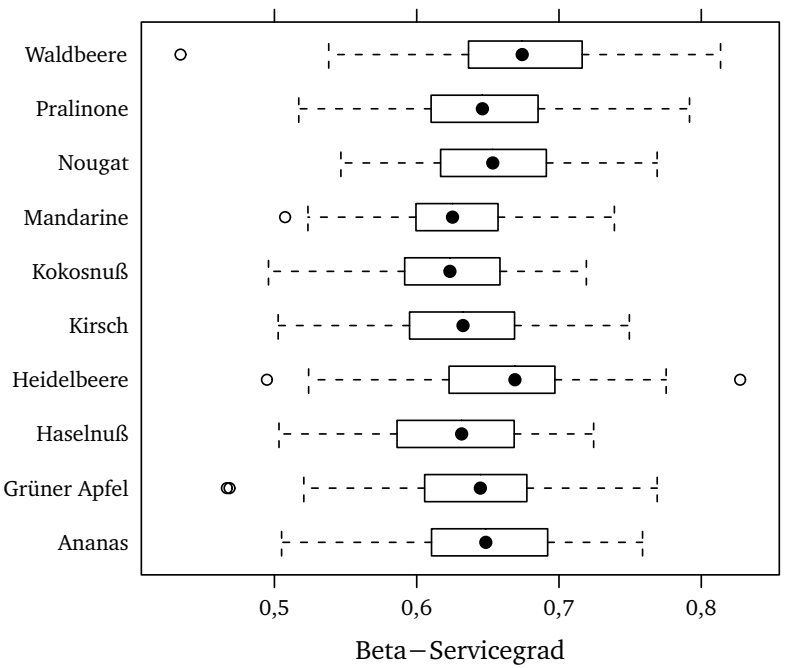


Abbildung 8.2: Produktbezogener Vergleich des β-Servicegrades (geschlossene Produktion, Referenztestreihe)

eine stetige Verteilungsfunktion besitzen (Duller 2008, S. 165). Alle Annahmen sind in diesem Fall erfüllt.

Die Nullhypothese des zweiseitigen Tests lautet, daß die Verteilungen um einen Betrag $\mu = 0$ verschoben sind. Zur Widerlegung der Nullhypothese muß gezeigt werden, daß die Verschiebung $\mu \neq 0$ ist, wobei die Richtung unbekannt ist, so daß zweiseitig getestet werden muß. Der Fehler erster Art (α-Fehler) besagt, daß mit dieser Wahrscheinlichkeit die Nullhypothese abgelehnt wird, obwohl die Verteilungen gegeneinander nicht verschoben sind.

Bei der Durchführung der Hypothesentests mit Hilfe von Computerprogrammen wie R oder SPSS wird kein Signifikanzniveau vorgegeben, sondern ein p-Wert für die Teststatistik berechnet. „Der p-Wert gibt die Wahrscheinlichkeit dafür an, unter der Nullhypothese die konkrete Stichprobe oder eine (in Bezug auf die Nullhypothese) noch seltenere Stichprobe zu beobachten. Anders ausgedrückt gibt der p-Wert das kleinste Testniveau an, auf dem die Stichprobe gerade noch signifikant ist. Ist der p-Wert kleiner oder gleich α wird zugunsten der Alternativhypothese entschieden, ansonsten wird die Nullhypothese beibehalten." (Duller 2008, S. 17). In den folgenden Abschnitten werden mehrere Tabellen vorgestellt, die p-Werte enthalten. Diese sind meist so formatiert, daß Werte, die ein bestimmtes Signifikanzniveau (z. B. $\alpha = 5\%$) unter-/überschreiten, hervorgehoben sind. Die Nullhypothese kann in diesen Fällen zum vorgegebenen Signifikanzniveau abgelehnt/nicht abgelehnt werden.

Da immer nur zwei Verteilungen gegeneinander getestet werden können, muß für einen vollständigen Vergleich aller Produkte für jede Produkt-Produkt Kombination der Signifikanztest durchgeführt werden. Tabelle 8.4 auf der nächsten Seite zeigt diesen Vergleich, wobei die p-Werte hervorgehoben sind, die über 5% liegen. Bei diesen Vergleichen kann die Nullhypothese zu einem Signifikanzniveau von $\alpha = 5\%$ nicht mehr verworfen werden, so daß angenommen wird, daß die Verschiebung bei $\mu = 0$ liegt. Die Tabelle ist symmetrisch, d. h. der Vergleich von Ananas mit Haselnuß ergibt den gleichen α-Fehler wie der Vergleich von Haselnuß mit Ananas. Die Werte der Hauptdiagonalen wurden ausgelassen.

	Ananas	Gr. Apfel	Haselnuß	Heidelbeere	Kirsch
Ananas		**0,31777**	0,01270	**0,07789**	0,03322
Grüner Apfel	**0,31777**		**0,14256**	0,00746	**0,24095**
Haselnuß	0,01270	**0,14256**		0,00006	**0,82918**
Heidelbeere	**0,07789**	0,00746	0,00006		0,00016
Kirsch	0,03322	**0,24095**	**0,82918**	0,00016	
Kokosnuß	0,00123	0,01834	**0,46662**	0,00000	**0,26531**
Mandarine	0,00415	**0,05179**	**0,77826**	0,00000	**0,45918**
Nougat	**0,70459**	**0,19619**	0,00643	**0,13991**	**0,01647**
Pralinone	**0,85396**	**0,26636**	0,00957	**0,12241**	0,02321
Waldbeere	0,00244	0,00005	0,00000	**0,15074**	0,00000

	Kokosnuß	Mandarine	Nougat	Pralinone	Waldbeere
Ananas	0,00123	0,00415	**0,70459**	**0,85396**	0,00244
Grüner Apfel	0,01834	**0,05179**	**0,19619**	**0,26636**	0,00005
Haselnuß	**0,46662**	**0,77826**	0,00643	0,00957	0,00000
Heidelbeere	0,00000	0,00000	**0,13991**	**0,12241**	**0,15074**
Kirsch	**0,26531**	**0,45918**	0,01647	0,02321	0,00000
Kokosnuß		**0,64230**	0,00028	0,00066	0,00000
Mandarine	**0,64230**		0,00128	0,00236	0,00000
Nougat	0,00028	0,00128		**0,96791**	0,00508
Pralinone	0,00066	0,00236	**0,96791**		0,00596
Waldbeere	0,00000	0,00000	0,00508	0,00596	

Tabelle 8.4: p-Werte des Wilcoxon-Rangsummentests im paarweisen Produktvergleich des β-Servicegrades (geschlossene Produktion, Referenztestreihe)

Gemeinsam mit dem Wilcoxon-Rangsummentest ist es auch möglich, ein entsprechendes Konfidenzintervall für die Verschiebung des Mittelwertes zu berechnen. Die Tabellen 8.5 bis 8.6 auf den Seiten 289–290 enthalten die untere bzw. obere Grenze des Konfidenzintervalls zu einem Testniveau von 95%. Die Tabelle muß so gelesen werden, daß bei einem Vergleich zwischen Ananas und Haselnuß die untere Grenze bei 0,0043 und die obere Grenze bei 0,0037 liegt. Da der Median der Verteilung von Ananas bei 0 liegt, ist der Servicegrad von Haselnuß höher als der von Ananas. Der entsprechende p-Wert liegt bei 0,0127 (1,27%). Auch diese Tabellen sind wieder symmetrisch, d. h. die Grenzen werden nur vertauscht, sobald die Reihenfolge des Vergleichs umgekehrt wird.

Durch den Einsatz von graphischer Darstellung und nichtparametrischen Tests läßt sich so zeigen, daß die Servicegrade der Produkte stark unterschiedlich sind. Da die Lagerkostensätze konstant sind, kann die Ursache hierfür nur noch in der unterschiedlichen Nachfrage liegen. Die Mittelwerte der Nachfrage sind identisch, doch die Werte für die deterministische Nachfrage werden aus einer Verteilung erstellt, die einen hierzu identischen Mittelwert besitzt und einen Variationskoeffizienten von 0,1. Da der Planungshorizont endlich ist, unterscheiden sich die gemessenen Mittelwerte. Die deterministische Nachfragereihe dient auch als Planungsgrundlage für die stochastische Kundennachfrage. Diese wird normiert, so daß der Erwartungswert der Kundennachfrage mit dem Erwartungswert der deterministischen Nachfragereihe übereinstimmt (vgl. Abschnitt 7.2.8). So kommt es zu den unterschiedlichen Erwartungswerten der Kundennachfrage, die sich in Abbildung 8.1 auf Seite 284 zeigen.

Alternative Methoden zur Auswertung sind die Bestimmung des Typs der Verteilung sowie der Lageparameter. Sobald diese statistisch abgesichert sind, können spezialisiertere Verfahren ein wesentlich genaueres Konfidenzintervall liefern, als die hier verwendeten nichtparametrischen Verfahren. Die Anwendung dieser Verfahren empfiehlt sich nach der Entwicklung eines mathematischen Modells zur Erklärung der Beobachtungen. Dieses liegt hier nicht vor, daher beschränken sich die folgenden Abschnitte auf graphische Darstellungen sowie nichtparametrische Testverfahren.

	Ananas	Gr. Apfel	Haselnuß	Heidelbeere	Kirsch
Ananas		-0,00818	0,00430	-0,03044	0,00122
Grüner Apfel	-0,02500		-0,00431	-0,03800	-0,00653
Haselnuß	-0,03704	-0,02803		-0,05142	-0,01942
Heidelbeere	-0,00178	0,00605	0,01715		0,01553
Kirsch	-0,03533	-0,02684	-0,01470	-0,04885	
Kokosnuß	-0,04190	-0,03400	-0,02211	-0,05606	-0,02412
Mandarine	-0,03775	-0,03023	-0,01841	-0,05138	-0,02044
Nougat	-0,01351	-0,00531	0,00685	-0,02798	0,00315
Pralinone	-0,01504	-0,00698	0,00555	-0,02934	0,00266
Waldbeere	0,00967	0,01746	0,02925	-0,00463	0,02675

	Ananas	Gr. Apfel	Haselnuß	Heidelbeere	Kirsch
Ananas		0,02500	0,03704	0,00178	0,03533
Grüner Apfel	0,00818		0,02803	-0,00605	0,02684
Haselnuß	-0,00430	0,00431		-0,01715	0,01470
Heidelbeere	0,03044	0,03800	0,05142		0,04885
Kirsch	-0,00122	0,00653	0,01942	-0,01553	
Kokosnuß	-0,00993	-0,00277	0,00998	-0,02487	0,00687
Mandarine	-0,00690	0,00019	0,01341	-0,02176	0,01014
Nougat	0,01836	0,02683	0,03898	0,00404	0,03715
Pralinone	0,01843	0,02624	0,03937	0,00409	0,03651
Waldbeere	0,04161	0,04993	0,06265	0,02772	0,05982

Tabelle 8.5: Untere und obere Grenzen des 95% Konfidenzintervalls im paarweisen Produktvergleich des β-Servicegrades (geschlossene Produktion, Referenztestreihe)

	Kokosnuß	Mandarine	Nougat	Pralinone	Waldbeere
Ananas	0,00993	0,00690	-0,01836	-0,01843	-0,04161
Grüner Apfel	0,00277	-0,00019	-0,02683	-0,02624	-0,04993
Haselnuß	-0,00998	-0,01341	-0,03898	-0,03937	-0,06265
Heidelbeere	0,02487	0,02176	-0,00404	-0,00409	-0,02772
Kirsch	-0,00687	-0,01014	-0,03715	-0,03651	-0,05982
Kokosnuß		-0,01785	-0,04359	-0,04320	-0,06680
Mandarine	-0,01074		-0,03948	-0,03870	-0,06288
Nougat	0,01252	0,00969		-0,01552	-0,03957
Pralinone	0,01183	0,00832	-0,01655		-0,04067
Waldbeere	0,03613	0,03313	0,00743	0,00708	

	Kokosnuß	Mandarine	Nougat	Pralinone	Waldbeere
Ananas	0,04190	0,03775	0,01351	0,01504	-0,00967
Grüner Apfel	0,03400	0,03023	0,00531	0,00698	-0,01746
Haselnuß	0,02211	0,01841	-0,00685	-0,00555	-0,02925
Heidelbeere	0,05606	0,05138	0,02798	0,02934	0,00463
Kirsch	0,02412	0,02044	-0,00315	-0,00266	-0,02675
Kokosnuß		0,01074	-0,01252	-0,01183	-0,03613
Mandarine	0,01785		-0,00969	-0,00832	-0,03313
Nougat	0,04359	0,03948		0,01655	-0,00743
Pralinone	0,04320	0,03870	0,01552		-0,00708
Waldbeere	0,06680	0,06288	0,03957	0,04067	

Tabelle 8.6: Untere und obere Grenzen des 95% Konfidenzintervalls im paarweisen Produktvergleich des β-Servicegrades (geschlossene Produktion, Referenztestreihe)

8.3 Fingerabdrücke eines Planungssystems

Bei den bisher durchgeführten Vergleichen wurde der Servicegrad in Abhängigkeit von der Nachfrage gezeigt, also eine Ausgabegröße mit einer Eingabegröße. Das Planungssystem erzeugt jedoch weitere Ausgabegrößen, wie die mittlere Losgröße, die auch ein Ergebnis der Planung ist. Ein entsprechender Vergleich der mittleren Losgröße mit dem β-Servicegrad wird in Abbildung 8.3 auf der nächsten Seite durchgeführt. Es zeigt sich, daß tendenziell mit kleineren Losgrößen ein höherer Servicegrad erreicht werden kann. Dies hängt u. a. damit zusammen, daß hier eine geschlossene Produktion betrachtet wird, bei der ein Los erst nach Abschluß der Produktion verfügbar ist.

Ein wesentlich differenzierteres Bild ergibt sich bei der Betrachtung einer offenen Produktion, wie sie in Abbildung 8.4 auf Seite 293 zu sehen ist. Für jedes Produkt bildet sich über alle 100 Testinstanzen eine gleichförmige Punktwolke heraus, die sich zwischen den Produkten nur in den Lageparametern (mittlere Losgröße, mittlerer Servicegrad) unterscheidet. Zu beobachten ist auch, daß die Dichte der Wolke zum Mittelpunkt hin zunimmt.

Weitere interessante Einsichten ergeben sich bei der Betrachtung der Abhängigkeit zwischen Nachfrage und mittlerer Losgröße in Abbildung 8.5 auf Seite 295. Es gibt eine Tendenz zu geringeren Losgrößen, wenn die Nachfrage geringer ist (die Umhüllende der Punktwolke verläuft von links oben nach rechts unten). Eine besondere Stellung nimmt jedoch die Struktur ein, die von den einzelnen Punkte gebildet wird. Die Form ist vergleichbar mit einer Kurvenschar. In den weiteren Abschnitten wird gezeigt werden, daß diese Struktur ein Merkmal ist, das ähnlich wie ein Fingerabdruck charakteristisch für dieses Planungssystem ist.

Die Form der Kurvenscharen kann zunächst als Geraden bezeichnet werden, aufgrund der vergleichsweise geringen Intervallgröße kann es sich jedoch auch um ein Polynom, eine gebrochen rationale Funktion oder auch Teile von Exponential-, Logarithmusfunktionen o.ä. enthalten. Die Kurvenscharen verlaufen nicht exakt parallel zueinander. Es wird vermutet, daß sie auf einen gemeinsamen Punkt zustreben, der

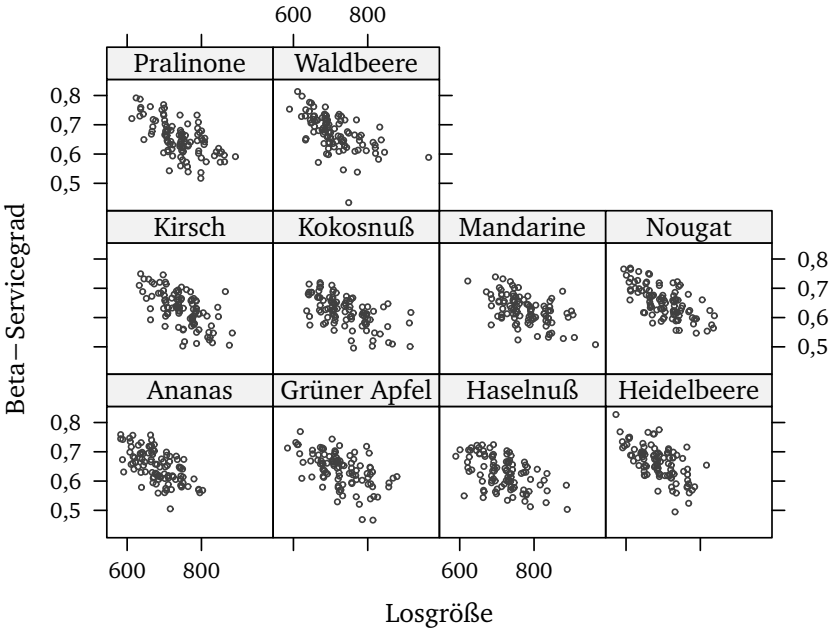


Abbildung 8.3: Vergleich zwischen mittlerer Losgröße und β-Servicegrad (geschlossene Produktion, Referenztestreihe)

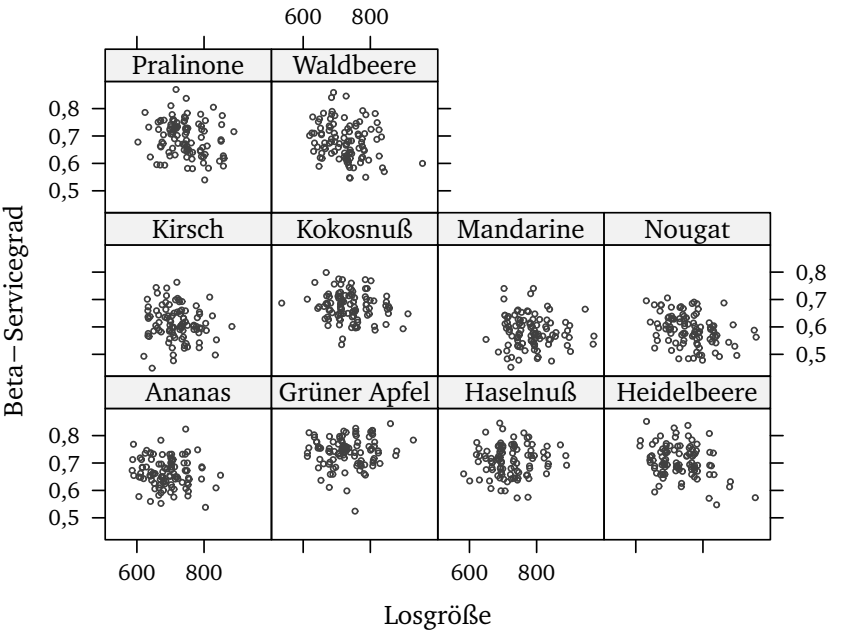


Abbildung 8.4: Vergleich zwischen mittlerer Losgröße und β-Servicegrad (offene Produktion, Referenztestreihe)

z. B. bei einer Nachfrage von 0 und einer Losgröße von 0 liegt (vgl. Abbildung 8.14 auf Seite 307).

Der Verlauf der Kurvenscharen zeigt, daß die gleiche Nachfrage in unterschiedlichen Testinstanzen zu verschiedenen mittleren Losgrößen führen kann. Der Raum zwischen den einzelnen Kurven enthält keine Datenpunkte, so daß das Planungssystem nicht mit diesen Parametern arbeitet. Die Kurvenscharen erfüllen Kriterien, die zu einer Stabilisierung des Systems führen. Eine mathematische Beschreibung dieser Kurvenscharen ist bisher nicht bekannt. In den folgenden Abschnitten sollen jedoch Veränderungen an unterschiedlichen Parametern untersucht werden, sowie deren Einflüsse auf die Lage der Kurvenscharen.

Die gemeinsame Darstellung der Informationen aus den Abbildungen 8.4 bis 8.5 auf den Seiten 293–295 führt zu den Abbildungen 8.6 bis 8.7 auf Seite 296. Die in Abbildung 8.5 auf der gegenüberliegenden Seite entdeckten Kurvenscharen sind auch hier zu erkennen als Punktwolken im Raum, der zwischen mittlerer Losgröße, Nachfrage und Servicegrad aufgespannt wird. Die Punktwolken bilden auch hier deutlich Strukturen aus, jedoch entsprechend der räumlichen Darstellung als Ebenen (gekrümmte Flächen?) zu erkennen sind. Mit steigender Losgröße nimmt auch der Abstand zwischen den Ebenen zu.

Beide Abbildungen enthalten die Datenpunkte aller Produkte für 100 Testinstanzen. Die durch jedes Produkt einzeln gebildeten Punktwolken überlagern sich zu einer gemeinsamen Struktur. Es ist jedoch nicht so, daß eine Ebene einem Produkt entspricht, sondern daß jedes Produkt mehrere Ebenen enthält, die sich zu einer gemeinsamen Schar an Ebenen überlagern.

Die Bezeichnung „Fingerabdruck“ ist insofern zutreffend, da die charakteristische Struktur auch in allen weiteren Testreihen für dieses Planungssystem nachzuweisen ist. In weiteren Untersuchungen kann die Hypothese untersucht werden, daß jedes Planungssystem - auch stochastische Lagerhaltungspolitiken - eine charakteristische Punktwolke/Struktur im Raum zwischen Losgröße, Nachfrage und Servicegrad ausbilden. Eine alternative Darstellung, die insbesondere bei stochastischen Lagerhaltungspolitiken vielleicht noch zu anderen Strukturen führt, entsteht durch den Austausch von Losgröße und Lagerbestand.

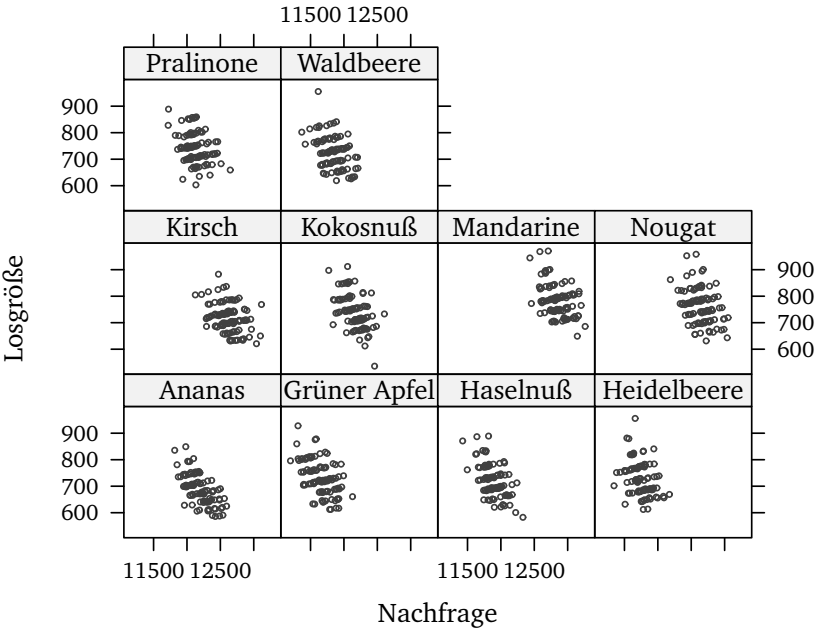


Abbildung 8.5: Vergleich zwischen Nachfrage und mittlerer Losgröße (offene Produktion, Referenztestreihe)

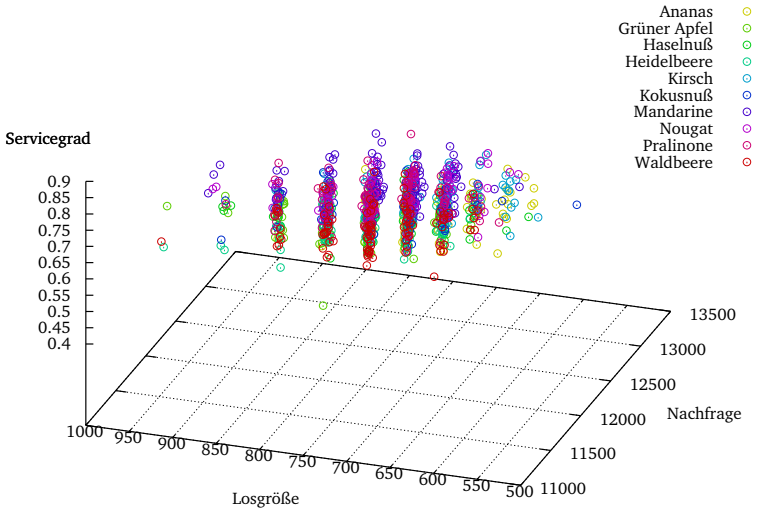


Abbildung 8.6: Fingerabdruck eines Planungssystems mit offener Produktion (produktspezifische Einfärbung)

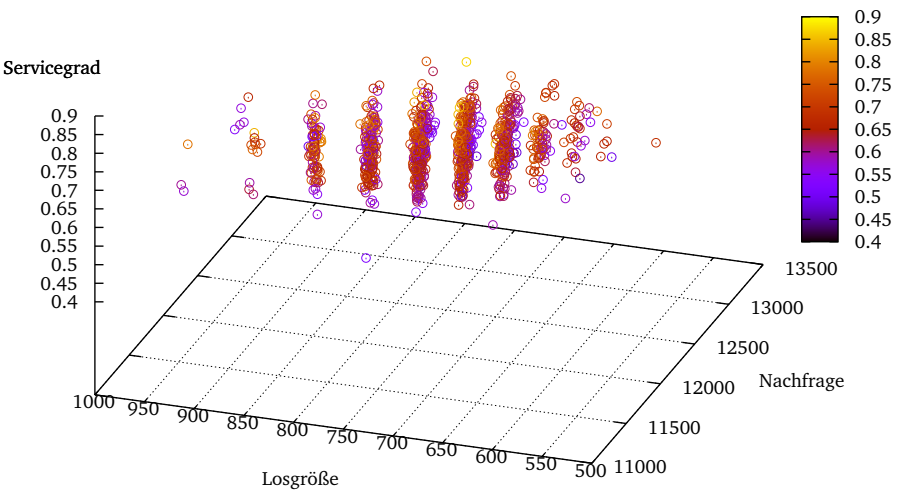


Abbildung 8.7: Fingerabdruck eines Planungssystems mit offener Produktion (Einfärbung gemäß β-Servicegrad)

8.4 Erhöhung der Sicherheitsbestände

In der Referenztestreihe wurden bisher keine Sicherheitsbestände betrachtet, so daß ein im Vergleich zur Praxis geringer β-Servicegrad gemessen wurde. Ein Mittel zur Erhöhung des Servicegrades ist die Einführung von Sicherheitsbeständen. Untersucht wird die Planung unter Berücksichtigung eines Sicherheitsbestandes für alle Produkte von 10%, 20% und 30% der mittleren Periodennachfrage. Die Sicherheitsbestände sind bereits zu Beginn der Simulation in den Anfangslagerbeständen enthalten. Die Daten werden mit denen der Referenztestreihe bei geschlossener Produktion verglichen. Die Abbildung auf der nächsten Seite zeigt den direkten Vergleich zwischen den einzelnen Testreihen getrennt nach Produkten. Wie erwartet, ist der Servicegrad für jedes Produkt mit zunehmendem Sicherheitsbestand gestiegen.

Bei der Form der Verteilungen ist mit steigendem Sicherheitsbestand bei einigen Produkten eine Änderung zu beobachten. Bei Haselnuß und Ananas nimmt die Spannbreite des β-Servicegrades mit steigendem Sicherheitsbestand ab. Das Planungssystem kann bei beiden Produkten besser auf eine höhere Nachfrage reagieren, so daß der Servicegrad steigt.

Eine weitere Deformation der Verteilungen ist zu beobachten, wenn der Sicherheitsbestand weiter erhöht wird (hier nicht abgebildet). Bereits bei der letzten Testreihe mit einem Sicherheitsbestand von 30% der mittleren Nachfrage wird in einigen Testinstanzen ein Servicegrad nahe 100% erreicht. Bei einer weiteren Erhöhung des Sicherheitsbestandes würde sich dieser Anteil erhöhen und die Verteilung entsprechend deformiert, da der β-Servicegrad nicht über 100% steigen kann. Für systematische Untersuchungen empfiehlt es sich daher, nur Sicherheitsbestände in einem Bereich zu betrachten, in dem keine oder nur wenige Testinstanzen einen β-Servicegrad von 100 % erreichen.

Wie auch in Abschnitt 8.3 zeigen sich auch hier erneut die bereits beschriebenen Strukturen. Abbildung 8.9 auf Seite 299 zeigt diese in einer Überlagerung aller durch die einzelnen Produkte aufgespannten Flächen zu einer gemeinsamen Schar an Flächen. Die Abbildung stammt

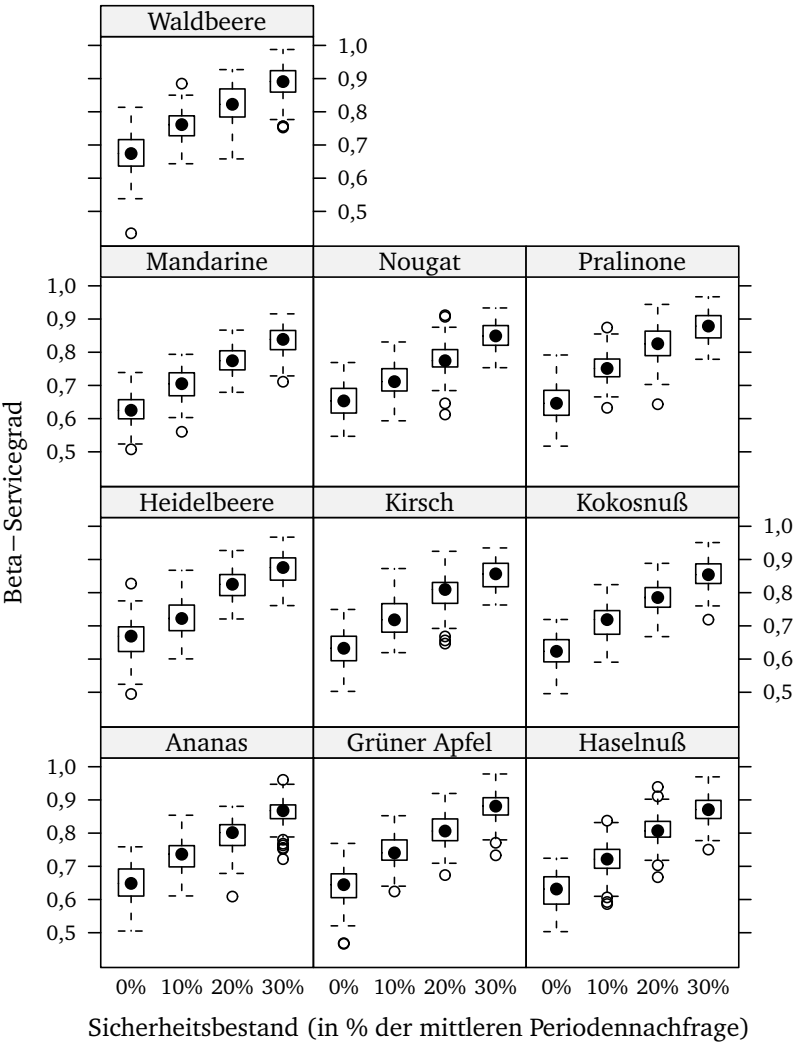


Abbildung 8.8: Produktbezogener Vergleich des β-Servicegrades bei steigendem Sicherheitsbestand (0% bis 30%, geschlossene Produktion, Referenztestreihe)

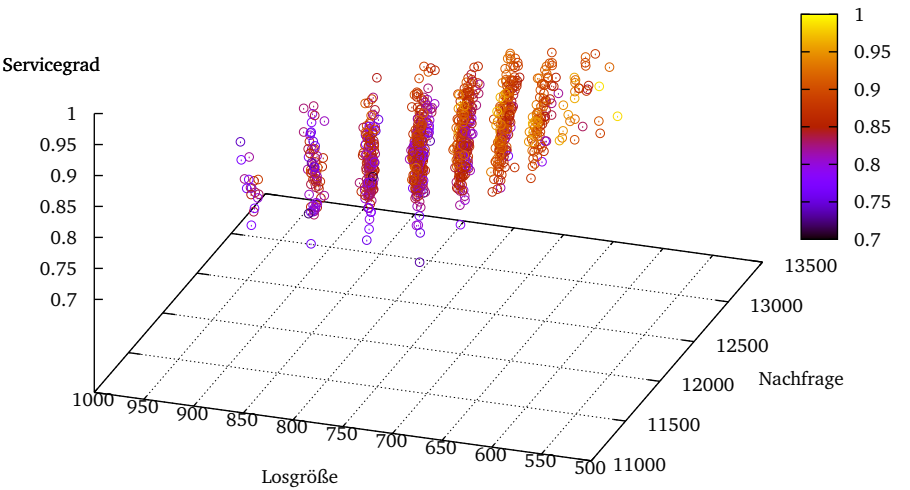


Abbildung 8.9: Fingerabdruck eines Planungssystems bei einem Sicherheitsbestand von 30% der mittleren Periodennachfrage (Einfärbung gemäß β-Servicegrad, geschlossene Produktion, Referenztestreihe)

aus einer Testreihe mit einem Sicherheitsbestand von 30% der mittleren Periodennachfrage bei geschlossener Produktion.

Von Interesse ist hier der direkte Vergleich zwischen der Referenztestreihe ohne Sicherheitsbestand und einer Testreihe mit Sicherheitsbestand. Abbildung 8.10 auf der nächsten Seite zeigt die bereits eingesetzte Überlagerung aller Produkte für die Referenztestreihe und die Testreihe aus Abbildung 8.9. Durch den höheren Sicherheitsbestand werden die Datenpunkte entlang der β-Servicegrad-Achse verschoben, während die Lage der Flächen im Raum gleich geblieben ist. Auch die Variabilität, d. h. die Spannbreite des β-Servicegrades in Abhängigkeit von der Nachfrage, ist weiterhin hoch.

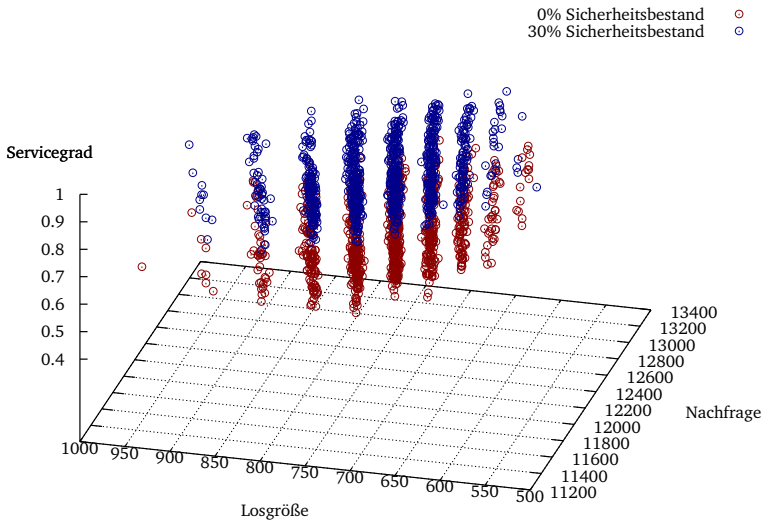


Abbildung 8.10: Vergleich der Fingerabdrücke des Planungssystems bei Sicherheitsbeständen von 0% und 30% der mittleren Periodennachfrage (geschlossene Produktion, Referenztestreihe)

8.5 Veränderung der Variabilität der Nachfrage

Die in den vorangehenden Abschnitten vorgestellten Testreihen haben mit identischen Nachfragereihen gearbeitet. Sowohl die deterministische Nachfrage, als auch die stochastische Kundennachfrage war für alle Testreihen identisch. Der Variationskoeffizient lag bei 0,1 und verursachte in der auf ein Jahr bezogenen Nachfrage nur geringe Schwankungen. Wird dieser heraufgesetzt auf 0,2, so erhöht sich die Spannbreite der Nachfrage und damit auch die Variabilität im gesamten System. Zu beachten ist, daß durch den höheren Variationskoeffizienten auch die deterministische Nachfrage verändert wurde, und somit auch die Berechnungsgrundlage für die Produktionskoeffizienten. Da sich der Mittelwert jedoch nicht verändert hat, wurde eine Skalierung der Stichproben aus der eingesetzten Verteilung durchgeführt. Der für die Berechnung des Produktionskoeffizienten verwendeter Mittelwert weist so nicht wesentlich von dem vorher berechneten Wert ab.

Geschlossene Produktion

Die Auswirkungen der Skalierung auf den β-Servicegrad zeigt die Abbildung auf der folgenden Seite. Eine eindeutige Wirkungsrichtung ist nicht festzustellen. Es gibt Produkte, bei denen der Servicegrad ansteigt (Grüner Apfel, Haselnuß), sinkt (Nougat, Waldbeere), sowie nahezu gleich bleibt. Ein Wilcoxon-Rangsummentest (vgl. Tabelle 8.7 auf Seite 303) bestätigt diese Ergebnisse. Die gleichen Beobachtungen lassen sich auch mit Sicherheitsbeständen machen.

Offene Produktion

Ein wesentlich einheitlicheres Bild ergibt sich bei einer offenen Produktion auf Seite 304. Hier zeigt der Vergleich zwischen der Referenztestreihe und der Testreihe mit einem höheren Variationskoeffizienten, daß der Servicegrad bei nahezu allen Produkten stark sinkt. Der p-Wert des Wilcoxon-Rangsummentests liegt für alle Produkte unter 2%, für 9 von

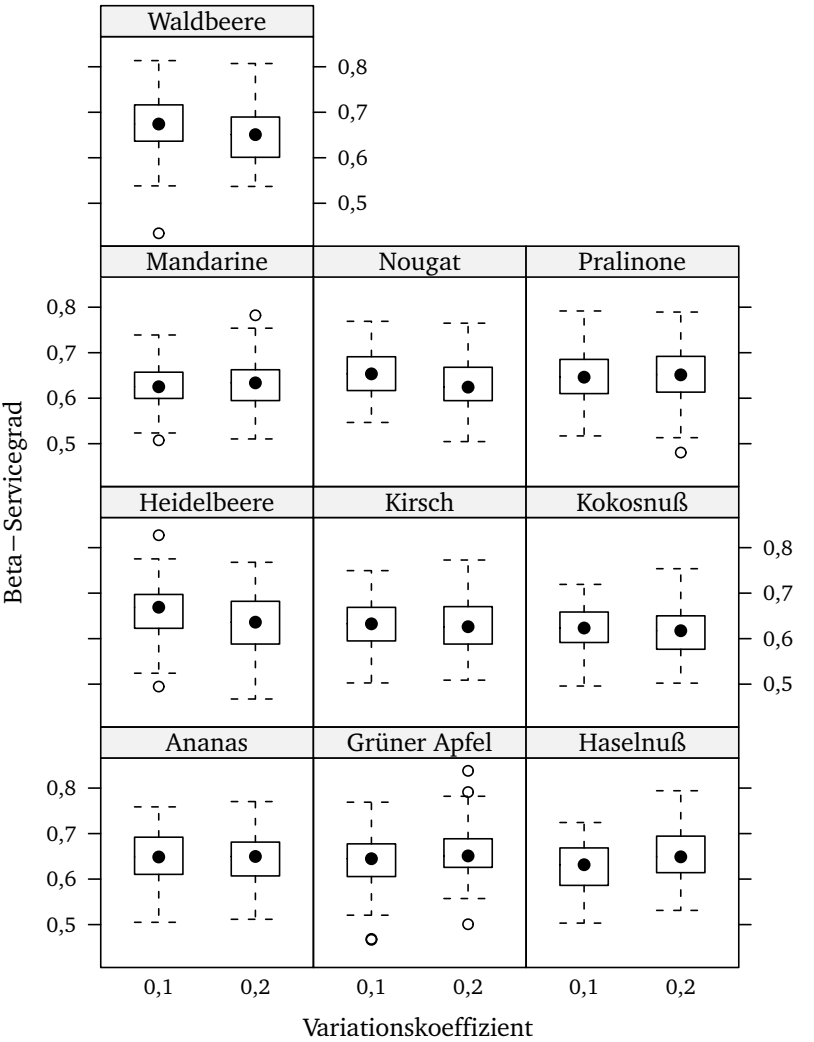


Abbildung 8.11: Produktbezogener Vergleich des β-Servicegrades bei steigender Variabilität der Nachfrage (geschlossene Produktion, Referenztestreihe)

Produkt	p-Wert
Ananas	0,61632
Grüner Apfel	0,04561
Haselnuß	0,00909
Heidelbeere	0,00128
Kirsch	0,84823
Kokosnuß	0,33954
Mandarine	0,99125
Nougat	0,00428
Pralinone	0,74851
Waldbeere	0,00261

Tabelle 8.7: p-Werte des Wilcoxon-Rangsummentests im Vergleich zwischen Referenztestreihe und einer Testreihe mit höherer Streuung der Nachfrage (geschlossene Produktion, $V = 0{,}1$ bzw. 0,2)

10 Produkten unter 0,1%. Die Obergrenzen der 95% Konfidenzintervalle zeigt Tabelle 8.8. Für nahezu alle Produkte kann eine Verringerung des Servicegrades um mehrere Prozente festgestellt werden. Nur bei einem Produkt liegt eine Erhöhung des Servicegrades vor (vgl. auch Abbildung 8.12 auf der nächsten Seite).

Fingerabdruck des Planungssystems

Abbildung 8.13 auf Seite 306 zeigt wiederum die bereits bekannte Struktur mit den Kurvenscharen/Flächen. Ein direkter Vergleich mit der Referenztestreihe bei geschlossener Produktion wurde in Abbildung 8.14 durchgeführt. Der höhere Variationskoeffizient führt, wie erwartet, zu einer größeren Spannbreite, die vom Planungssystem abgedeckt wird. Die Draufsicht zeigt, daß die Lage der Flächen in beiden Fällen identisch ist. Der höhere Variationskoeffizient hat dazu geführt, daß die Punktwolken der Referenztestreihen gestreckt wurden.

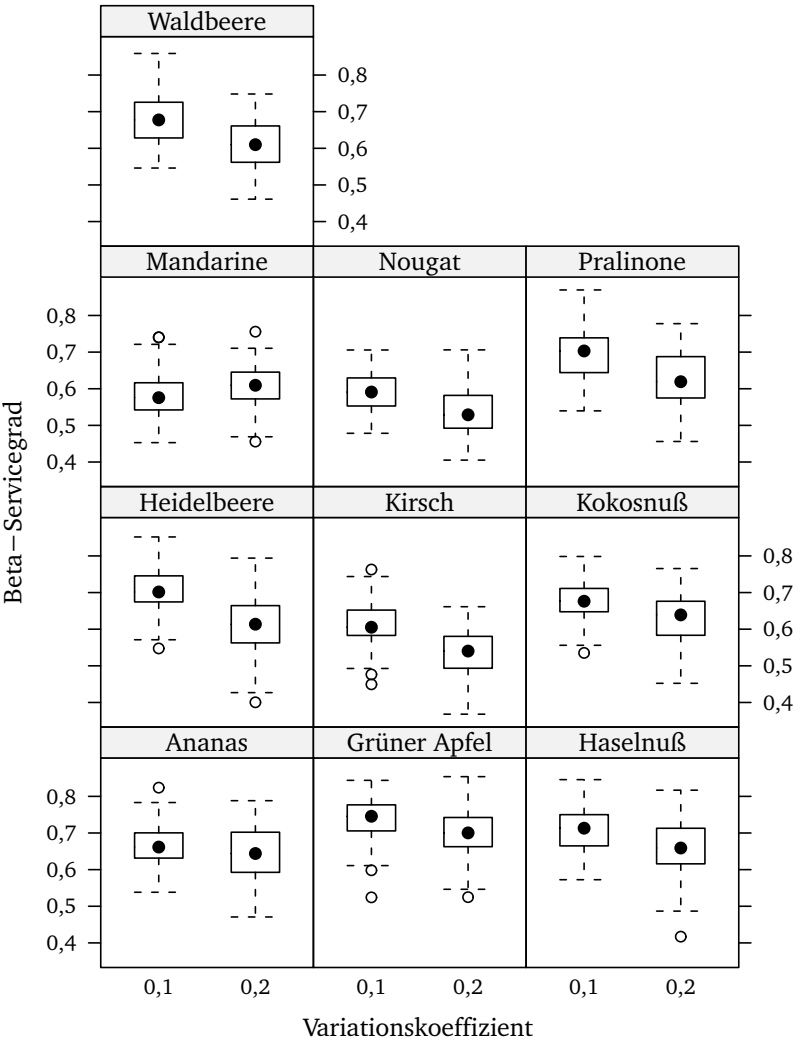


Abbildung 8.12: Produktbezogener Vergleich des β-Servicegrades bei steigender Variabilität der Nachfrage (offene Produktion, Referenztestreihe)

Produkt	Obergrenze
Ananas	-0,00483
Grüner Apfel	-0,02445
Haselnuß	-0,02967
Heidelbeere	-0,07484
Kirsch	-0,05889
Kokosnuß	-0,02657
Mandarine	0,04410
Nougat	-0,03835
Pralinone	-0,04493
Waldbeere	-0,04860

Tabelle 8.8: Obergrenze des 95% Konfidenzintervalls des Wilcoxon-Rangsummentests im Vergleich zwischen Referenztestreihe und einer Testreihe mit erhöhtem Variationskoeffizienten (offene Produktion, V=0,1 bzw. 0,2)

8.6 Veränderungen der Strafkosten

Eine weitere Größe, deren Einfluß auf die Planung untersucht werden soll, ist die Höhe der Strafkosten, die in der Planung für eine Unterschreitung des Sicherheitsbestandes und Lagerbestandes (Backlog) veranschlagt werden. Es wird vermutet, daß durch eine Veränderung der Höhe die Unterschiede zwischen den einzelnen Produkten ab- oder zunimmt und daß sich die Variabilität der erreichten Servicegrade verändert.

Für die Referenztestreihen wurden folgende Verhältnisse der Strafkostensätze angenommen $c_j^{BL} = 1000h_j$, $c_j^{SS} = 100h_j$ $c_j^{IT} = 50h_j$ (vgl. Tabelle 6.2 auf Seite 213). In einem zweiten Satz von Testreihen wurden die Strafkosten um den Faktor 10 reduziert auf $c_j^{BL} = 100h_j$, $c_j^{SS} = 10h_j$ $c_j^{IT} = 5h_j$. Für die Monatsmodelle wurde der Strafkostensatz für die Unterschreitung der Ziellagerbestände ein Satz von $10h_j$ und in der reduzierten Version von $5h_j$ angenommen.

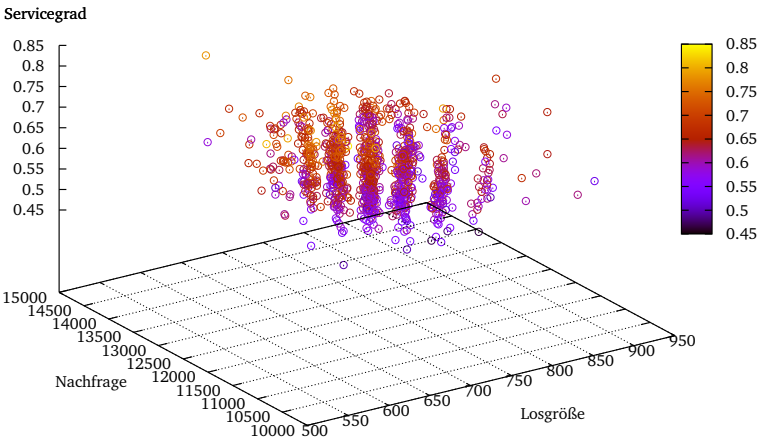


Abbildung 8.13: Fingerabdruck des Planungssystems bei einer höheren Variabilität der Nachfrage (geschlossene Produktion, Referenztestreihe)

Zur Untersuchung der Unterschiede zwischen den Testreihen gibt es mehrere Verfahren. Wenn davon ausgegangen wird, daß die Art der Verteilung nicht bekannt ist, können nur nichtparametrische Tests eingesetzt werden. Für die Untersuchung der Unterschiede in den Varianzen zwischen zwei unabhängigen Testreihen lassen sich der Siegel-Turkey-Test, Mood-Test oder Ansari-Bradley-Test einsetzen. Problematisch ist, daß diese Tests davon ausgehen, daß ein multiplikativer Faktor auf die Stichprobe einwirkt und somit sowohl den Mittelwert, als auch die Varianz verändert (Duller 2008, S. 181). Wenn hingegen der Mittelwert gleich bleibt und nur die Varianz sich verändert, können die Tests nicht eingesetzt werden.

Eine alternative Testmethode ist die Untersuchung, ob Verteilungen aus einer gemeinsamen Verteilung stammen. Als nichtparametrischen Test bietet sich hier der Kolmogorov-Smirnov-Test an (Duller 2008, S. 156). Getestet wird zweiseitig, d. h. die Identität der Verteilungen wird überprüft. Die Nullhypothese lautet daher, daß beide Verteilungen identisch sind. Die Durchführung des Tests erfolgt für jedes Produkt getrennt, durch einen paarweisen Vergleich der Daten beider Testreihen.

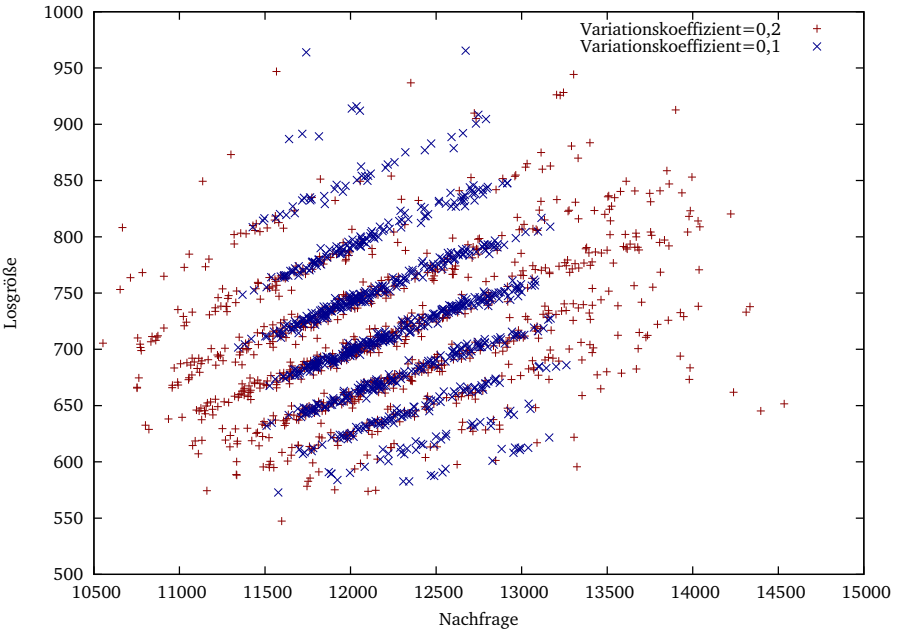


Abbildung 8.14: Abhängigkeit der Losgröße von der Nachfrage bei einer höheren Variabilität der Nachfrage (geschlossene Produktion, Referenztestreihe)

Geschlossene Produktion

Die Tabellen 8.9a bis 8.9b auf der gegenüberliegenden Seite zeigen die p-Werte des Kolmogorov-Smirnov-Tests für eine geschlossene Produktion bei identischen Lagerkostensätzen. Ein Signifikanzniveau $\alpha = 5\%$ wird nur in wenigen Fällen erreicht (diese sind in den Tabellen **fett** markiert). In den meisten Fällen liegt der p-Wert höher, so daß zu einem Signifikanzniveau von $\alpha = 5\%$ die Nullhypothese nicht abgelehnt werden kann. Die Verteilungen zeigen somit mit wenigen Ausnahmen keinen signifikanten Unterschied, unabhängig von der Höhe der Strafkosten. Sie stimmen sowohl in der Lage, als auch in der Form überein. Auch bei geschlossener Produktion mit nicht identischen Lagerkostensätzen zeigen sich vergleichbare Ergebnisse (vgl. Tabelle 8.10 auf Seite 310). Die Höhe der Strafkosten beeinflußt den erreichten Servicegrad nur in einigen wenigen Fällen signifikant. Eine mögliche Ursache ist die Gestaltung der Strafkosten. Die Sätze wurden so hoch gewählt, daß bei geringen Unterschreitungen von Sicherheitsbeständen oder Lagerbeständen ein Rüsten für das Modell vorteilhafter ist, als die Fortführung der Fehlbestände. Werden die Kostensätze geringer gewählt, so daß der Kostenvorsprung des Rüstens schrumpft, ergeben sich wahrscheinlich auch signifikantere Unterschiede in den erreichten Servicegraden.

Offene Produktion

Die Untersuchungen der geschlossenen Produktion werden für die offene Produktion wiederholt. Eingesetzt werden die gleichen Testverfahren, wie bei der geschlossenen Produktion. Erhalten bleiben die Nullhypothese und die Erwartung, daß sich die Verteilung in ihrer Form oder Lage ändert.

Tabelle 8.11 auf Seite 311 zeigt die p-Werte der Testgruppen mit offener Produktion und identischen/gleichverteilten Lagerkostensätzen. Bei gleichverteilten Lagerkostensätzen ist unabhängig von der Variabilität der Nachfrage kein signifikanter Unterschied meßbar. Das Verhalten bei identischen Lagerkostensätzen unterscheidet sich hiervon grundlegend. Bei geringer Variabilität ist noch kein signifikanter Unterschied

	Sicherheitsbestand			
Produkt	0%	10%	20%	30%
Ananas	0,96707	0,36673	0,58062	0,36673
Gr. Apfel	0,07832	0,15454	0,69937	0,36673
Haselnuß	0,21055	0,99376	0,46756	0,69937
Heidelbeere	0,58062	**0,00079**	0,36673	0,07832
Kirsch	0,69937	0,81275	0,15454	0,58062
Kokosnuß	0,81275	0,46756	**0,00630**	0,69937
Mandarine	0,81275	0,46756	0,28096	0,58062
Nougat	**0,00386**	0,90621	0,58062	0,58062
Pralinone	0,36673	0,28096	0,96707	0,58062
Waldbeere	0,96707	0,21055	0,90621	0,21055

(a) Variationskoeffizient=0,1

	Sicherheitsbestand			
Produkt	0%	10%	20%	30%
Ananas	0,21055	0,46756	0,99963	0,28096
Gr. Apfel	0,46756	0,81275	0,96707	0,69937
Haselnuß	0,05410	0,69937	0,36673	0,99376
Heidelbeere	$< \mathbf{10^{-5}}$	0,46756	0,81275	0,11113
Kirsch	0,69937	0,90621	0,58062	0,07832
Kokosnuß	0,69937	0,81275	0,36673	0,90621
Mandarine	0,46756	0,96707	0,58062	0,90621
Nougat	0,69937	0,36673	0,69937	0,96707
Pralinone	0,81275	0,36673	0,58062	0,81275
Waldbeere	0,69937	0,05410	**0,01581**	0,69937

(b) Variationskoeffizient=0,2

Tabelle 8.9: *p*-Werte des Kolmogorov-Smirnov-Test für den Vergleich zwischen Referenztestreihe und einer Testreihe mit geringeren Strafkosten bei steigenden Sicherheitsbeständen (geschlossene Produktion, identische Lagerkostensätze, Referenztestreihe)

Produkt	Sicherheitsbestand 0%	10%	20%	30%
Ananas	0,90621	0,28096	0,99963	0,46756
Gr. Apfel	0,81275	0,21055	0,21055	0,99963
Haselnuß	0,69937	0,90621	0,11113	0,58062
Heidelbeere	**0,01581**	0,36673	0,28096	0,28096
Kirsch	$< \mathbf{10^{-5}}$	0,81275	0,90621	0,28096
Kokosnuß	0,15454	0,58062	0,90621	0,81275
Mandarine	**0,00386**	0,58062	0,81275	0,69937
Nougat	**0,01581**	0,07832	**0,03663**	0,90621
Pralinone	0,69937	0,46756	0,90621	0,15454
Waldbeere	0,96707	0,99376	0,81275	0,81275

(a) Variationskoeffizient=0,1

Produkt	Sicherheitsbestand 0%	10%	20%	30%
Ananas	0,58062	0,99376	0,69937	0,58062
Gr. Apfel	**0,01581**	0,81275	0,81275	0,36673
Haselnuß	0,15454	0,96707	0,11113	**0,01008**
Heidelbeere	$< \mathbf{10^{-4}}$	0,99963	0,28096	0,96707
Kirsch	0,90621	0,46756	0,69937	0,36673
Kokosnuß	$< \mathbf{10^{-4}}$	0,46756	0,99376	**0,00232**
Mandarine	0,81275	0,99376	0,15454	0,90621
Nougat	0,58062	0,69937	0,99376	0,58062
Pralinone	0,36673	0,58062	0,69937	0,69937
Waldbeere	0,69937	0,58062	0,21055	0,90621

(b) Variationskoeffizient=0,2

Tabelle 8.10: *p*-Werte des Kolmogorov-Smirnov-Test für den Vergleich zwischen Referenztestreihe und einer Testreihe mit geringeren Strafkosten bei steigenden Sicherheitsbeständen (geschlossene Produktion, gleichverteilte Lagerkostensätze, Referenztestreihe)

Identische Lagerkostensätze

Produkt	Sicherheitsbestand 0%	10%
Ananas	0,36673	0,81275
Gr. Apfel	0,58062	**0,00630**
Haselnuß	0,05410	0,69937
Heidelbeere	0,90621	**0,01008**
Kirsch	0,81275	0,99376
Kokosnuß	0,05410	0,46756
Mandarine	0,21055	0,90621
Nougat	0,81275	0,58062
Pralinone	0,58062	0,58062
Waldbeere	0,96707	0,69937

(a) Variationskoeffizient=0,1

Produkt	Sicherheitsbestand 0%	10%
Ananas	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$
Gr. Apfel	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$
Haselnuß	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$
Heidelbeere	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$
Kirsch	$< \mathbf{10^{-3}}$	$< \mathbf{10^{-2}}$
Kokosnuß	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$
Mandarine	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$
Nougat	$< \mathbf{10^{-3}}$	$< \mathbf{10^{-3}}$
Pralinone	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$
Waldbeere	$< \mathbf{10^{-5}}$	$< \mathbf{10^{-5}}$

(b) Variationskoeffizient=0,2

Gleichverteilte Lagerkostensätze

Produkt	Sicherheitsbestand 0%	10%
Ananas	0,46756	0,58062
Gr. Apfel	0,36673	0,69937
Haselnuß	0,15454	0,28096
Heidelbeere	0,90621	0,58062
Kirsch	0,46756	0,46756
Kokosnuß	0,28096	0,36673
Mandarine	0,58062	**0,00004**
Nougat	0,81275	0,36673
Pralinone	0,11113	0,46756
Waldbeere	0,90621	0,90621

(c) Variationskoeffizient=0,1

Produkt	Sicherheitsbestand 0%	10%
Ananas	0,46756	0,69937
Gr. Apfel	0,81275	0,15454
Haselnuß	0,15454	0,81275
Heidelbeere	0,36673	**0,00232**
Kirsch	0,05410	0,69937
Kokosnuß	**0,00045**	0,46756
Mandarine	0,46756	0,36673
Nougat	0,81275	0,69937
Pralinone	0,90621	0,90621
Waldbeere	0,58062	0,28096

(d) Variationskoeffizient=0,2

Tabelle 8.11: *p*-Werte des Kolmogorov-Smirnov-Test für den Vergleich zwischen Referenztestreihe und einer Testreihe mit geringeren Strafkosten bei steigenden Sicherheitsbeständen (offene Produktion)

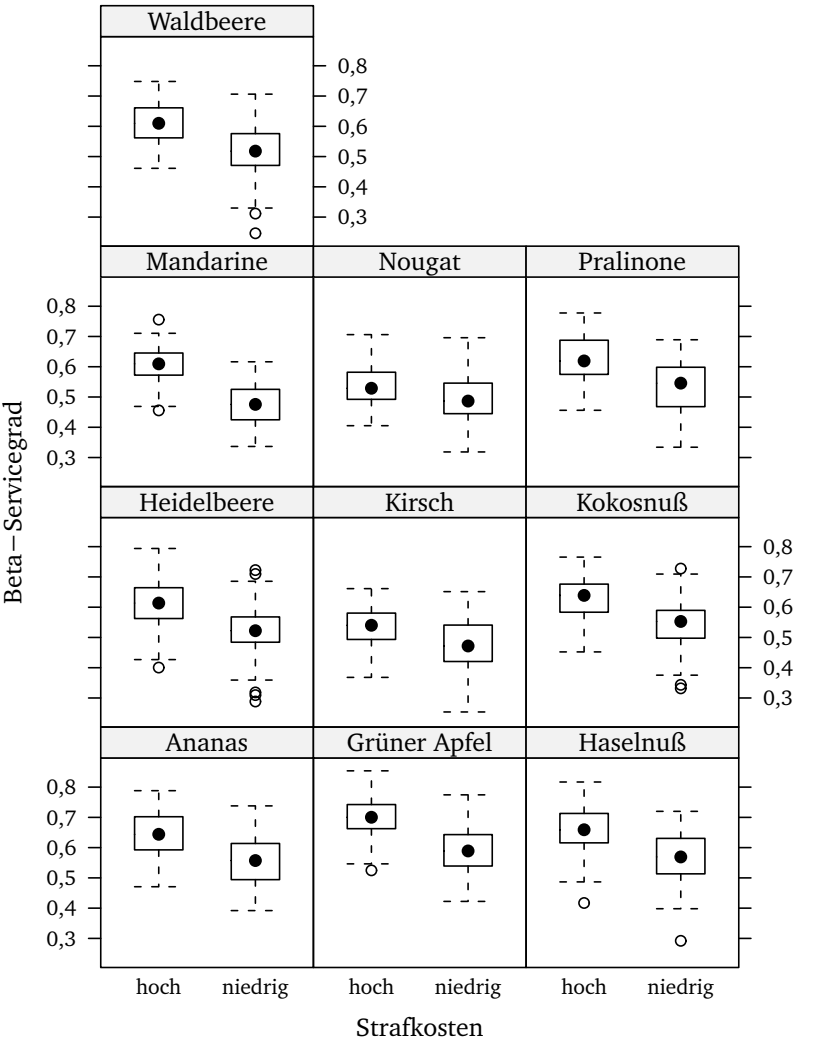


Abbildung 8.15: Produktbezogener Vergleich des β-Servicegrades bei einer Veränderung der Strafkosten (offene Produktion, identische Lagerkostensätze, $V=0{,}2$, $SB=0\%$)

zur Referenztestreihe meßbar. Mit steigender Variabilität der Nachfrage (vgl. Tabelle 8.11b) ist eine deutliche Veränderung sichtbar. Bei allen Produkten ist zu beobachten, daß der Servicegrad gegenüber der Referenztestreihe um ca. 5 bis 10% sinkt (vgl. Abbildung 8.15 auf der vorherigen Seite). Die geringeren Strafkosten haben hier eine negative Auswirkung auf das Planungssystem, so daß es die höhere Variabilität nicht mehr kompensieren werden kann.

Fingerabdruck des Planungssystems

In den vorangehenden Abschnitten wurde getestet, ob die Verteilung der β-Servicegrade bei einer Veränderung der Strafkosten sich verändert oder gleichbleibt. Die statistischen Tests betrachten die Verteilung der β-Servicegrade isoliert, ohne die Abhängigkeiten zu berücksichtigen, die sich aufgrund des Fingerabdrucks ergeben. Zur Betrachtung des Fingerabdrucks wird daher auf die graphische Darstellung zurückgegriffen.

Abbildung 8.16 auf Seite 315 zeigt die Überlagerung aller Produkte der Referenztestreihe und der äquivalenten Testreihe mit niedrigeren Strafkosten. Aus der Graphik ist ersichtlich, daß es auch keine Lokationsunterschiede in der Position der charakteristischen Kurvenscharen gibt. Anders sieht es aus, wenn die Testreihen mit offener Produktion verglichen werden, bei denen der Variationskoeffizient auf 0,2 zunimmt. Hier wurde bereits in Tabelle 8.11b auf Seite 311 gezeigt, daß es signifikante Unterschiede zwischen den Testreihen mit hohen und niedrigen Strafkosten gibt. Abbildung 8.18 auf Seite 317 zeigt auch, worauf sich die Veränderungen zurückführen lassen. Bei niedrigeren Strafkosten verlagern sich die Testreihen auf die Kurvenscharen, die bei gleicher Nachfrage eine höhere mittlere Losgröße besitzen. Da die Ressourcen jetzt im Mittel länger mit einem Produkt belegt sind, können sie sich nicht so schnell an die stärker schwankende Nachfrage anpassen.

Die gleiche Verschiebung hin zu höher gelegenen Kurvenscharen läßt sich auch bei niedrigen Variationskoeffizienten beobachten, jedoch nicht in der gleichen Intensität (vgl. Abbildung 8.17 auf Seite 316). Ein entsprechender Vergleich beider Verteilungen der Losgrößen mit Hilfe

des Kolmogorov-Smirnov-Tests ergibt einen p-Wert von 20%, der für eine fundierte Aussage zu hoch ist.

Aus der graphischen Darstellung lassen sich die Effekte sehr gut ablesen. Der wesentliche Unterschied ergibt sich aus der Spannbreite der Nachfrage und der damit verbundenen Spannbreite der mittleren Losgröße. Bei einem hohen Variationskoeffizienten der Nachfrage überdeckt ein Teil der Kurvenschar eine Differenz in der Losgröße von ≈ 200 Mengeneinheiten (vgl. Abbildung 8.18 auf Seite 317). Eine geringe Variabilität in der Nachfrage führt auch zu geringeren Schwankungen in der mittleren Losgröße. Bei einem Variationskoeffizienten von 0,1 überdeckt eine Kurve nur noch eine Differenz in der Losgröße von ≈ 100 Mengeneinheiten (vgl. Abbildung 8.17 auf Seite 316).

Eine Verlagerung innerhalb der Kurvenscharen hin zu größeren Losgrößen führt bei einer stärker schwankenden Nachfrage und niedrigeren Strafkosten zu geringeren Servicegraden, da jetzt auch die Losgrößen und damit die Zeit zwischen zwei Losauflagen stärker schwanken. Das System wird unflexibler, da es durch die größeren Lose länger an eine Entscheidung gebunden ist und nur noch langsamer auf Änderungen reagieren kann. Diese Aussage ist nur für eine offene Produktion gültig. Bei geschlossener Produktion lassen sich die durch die Strafkosten verursachten Effekte noch nicht beobachten. Es kann nur vermutet werden, daß durch die geschlossene Produktion das System unabhängig von den Strafkosten insgesamt unflexibler geworden ist, da Kundennachfragen während der Produktion eines Loses erst nach Abschluß der Produktion bearbeitet werden können. Die Auswirkungen durch geringere Strafkostensätze lassen sich wahrscheinlich erst beobachten, wenn die Schwankungen in der Nachfrage einen Grenzwert überschritten haben. Im Gegensatz zu einer offenen Produktion, die stetig auf Änderungen reagiert, wird hier eine sprunghafte Reaktion erwartet.

8.7 Veränderungen der Auslastung

Ein anderer Faktor, welcher die Höhe der Sicherheitsbestände beeinflußt, ist die Auslastung des Systems. In der Referenztestreihe wurde

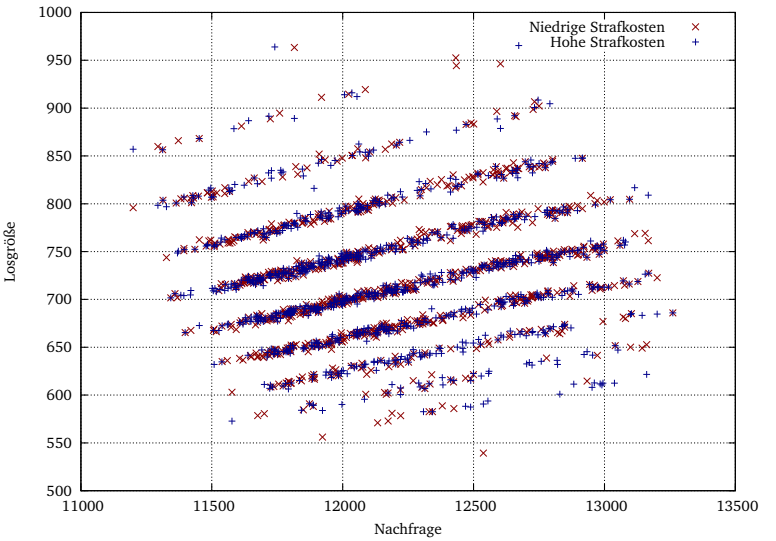


Abbildung 8.16: Fingerabdrücke des Planungssystems im Vergleich zwischen hohen und niedrigen Strafkosten (geschlossene Produktion, konstante Lagerkostensätze, V=0,1, SB=0%)

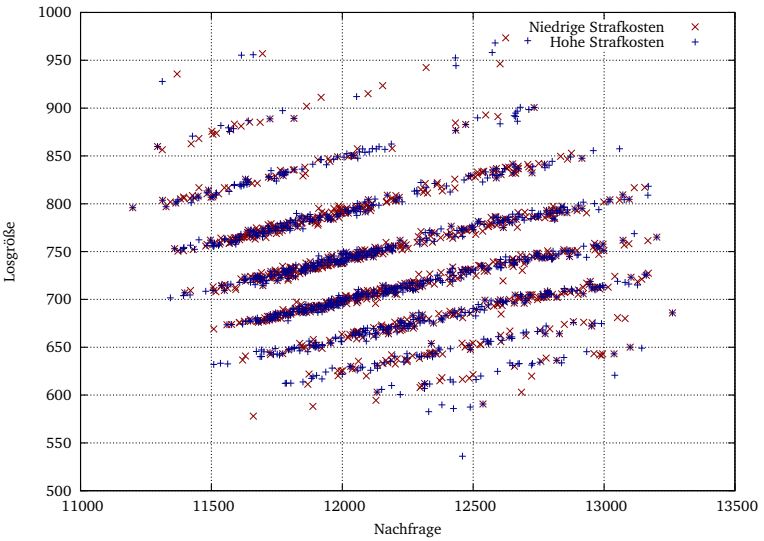


Abbildung 8.17: Fingerabdrücke des Planungssystems im Vergleich zwischen hohen und niedrigen Strafkosten (offene Produktion, konstante Lagerkostensätze, V=0,1, SB=0%)

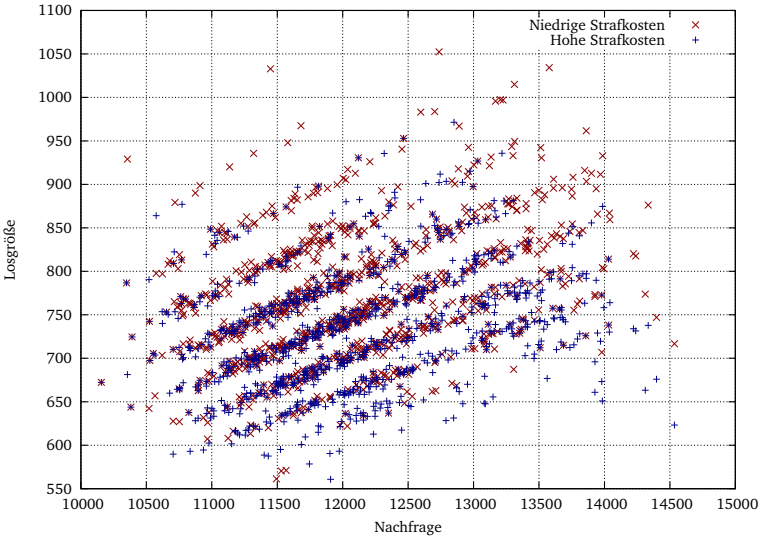


Abbildung 8.18: Fingerabdrücke des Planungssystems im Vergleich zwischen hohen und niedrigen Strafkosten (offene Produktion, konstante Lagerkostensätze, V=0,2, SB=0%)

eine Auslastung von 90% simuliert, wobei die Produktion einen Anteil von 95% der verfügbaren Kapazität besitzt (=85,5% der gesamten Kapazität). Sowohl für die geschlossene, als auch für die offene Produktion soll untersucht werden, welche Auswirkungen eine Veränderung der Zielauslastung hat. Erwartet wird, daß mit zunehmender Auslastung der Servicegrad abnimmt.

Durch die Konstruktion des Testdatengenerators verändert sich bei gleicher mittlerer Nachfrage (diese ist hier gegeben) mit der Auslastung auch der Produktionskoeffizient der einzelnen Produkte. Dieser wird so berechnet, daß auf den Maschinen die Zielauslastung erreicht und nicht überschritten wird. Wenn die Testreihen miteinander verglichen werden, muß somit immer berücksichtigt werden, daß die Produkte nicht identisch sind, sondern in ihren Relationen untereinander identisch geblieben sind.

Eine alternative Formulierung mit einer höheren Kapazität hätte bei gleichbleibenden Produktionskoeffizienten auch zu einer zeitlichen Verschiebung geführt. Eine höhere Kapazität bedeutet immer, daß die Maschinenlaufzeit ausgedehnt wird. Die Laufzeit der Lose bezogen auf die verfügbare Kapazität sinkt. Damit verbunden ist eine Veränderung der Produktionskoeffizienten. Die zeitlichen Auswirkungen wären vergleichbar.

Geschlossene Produktion

Die Analyse der Abbildung auf der gegenüberliegenden Seite zeigt die Quantile der β-Servicegrade nach Produkten und Testreihen getrennt. Ausgangspunkt ist die Referenztestreihe bei geschlossener Produktion mit einer Auslastung von 90%. Diese wurde in 5% Schritten variiert von 80% bis 95%.

Es ist zu erkennen, daß mit abnehmender Überschußkapazität auch der Servicegrad sinkt. Durch den geringeren Überschuß verringert sich auch die Flexibilität des Planungssystems und damit auch die Fähigkeit, sich an die Nachfrage besser anzupassen. Die gleichen Veränderungen lassen sich auch für die Testreihen mit Sicherheitsbeständen nachweisen, jedoch auf einem höheren Servicegrad-Niveau. Durch die

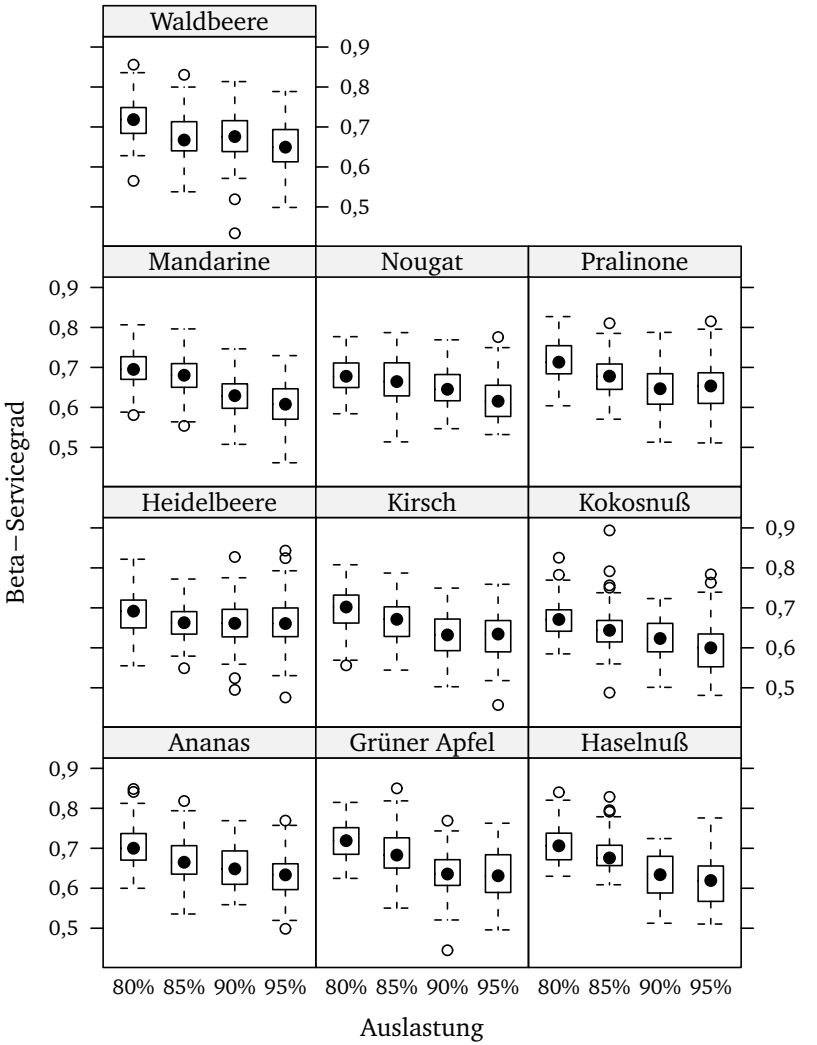


Abbildung 8.19: Produktbezogener Vergleich des β-Servicegrades unter verschiedenen Auslastungen (geschlossene Produktion, konstante Lagerkostensätze, V=0,1, SB=0%)

Sicherheitsbestände besitzt das Planungssystem eine Reserve, die es bei knappen Kapazitäten auch einsetzt.

Offene Produktion

Im Gegensatz zur geschlossenen Produktion zeigt sich das gleiche System bei einer offenen Produktion mit einem vollständig anderen Verhalten. Abbildung 8.20 auf der nächsten Seite zeigt den Vergleich der Referenztestreihe unter unterschiedlicher Auslastung. Bei allen Produkten läßt sich beobachten, daß der Servicegrad mit zunehmender Auslastung zuerst sinkt, dann steigt und anschließend sinkt. Die Auslastung von 85% ist für das Planungssystem insgesamt sehr ungünstig, da es selbst bei einer höheren Auslastung bessere Werte erreicht. Die Ursachen für dieses Verhalten sind nicht bekannt.

Fingerabdruck des Planungssystems

Außer den Unterschieden in den erreichten Servicegraden stellt sich auch die Frage, welche Auswirkungen eine Veränderung der Auslastung auf die Losgrößen im Planungssystem haben. Abbildung 8.21 auf Seite 322 zeigt für die Referenztestreihe bei geschlossener Produktion die Unterschiede zwischen einer Auslastung von 80% und 95%. Die charakteristischen Kurvenscharen sind auch hier erneut zu sehen und überlagern sich für alle Produkte unabhängig von der Auslastung. Es ist zu erkennen, daß mit steigender Auslastung die mittlere Losgröße steigt. Das Planungssystem muß in größeren Losen produzieren, da weniger Kapazität für zusätzliche Rüstvorgänge zur Verfügung steht. Durch die längeren Lose ist es aber auch für einen längeren Zeitraum an seine Entscheidung gebunden, so daß es an Flexibilität verliert. Die gleichen Beobachtungen lassen sich für die geschlossene Produktion auch bei den anderen untersuchten Auslastungen machen. Auch bei Einführung von Sicherheitsbeständen bleibt diese Struktur erhalten.

Die Anomalie, die bei einer offenen Produktion beobachtet wurde, findet sich auch beim Vergleich zwischen mittlerer Losgröße und Nachfrage wieder. Abbildung 8.22 auf Seite 323 zeigt den Vergleich

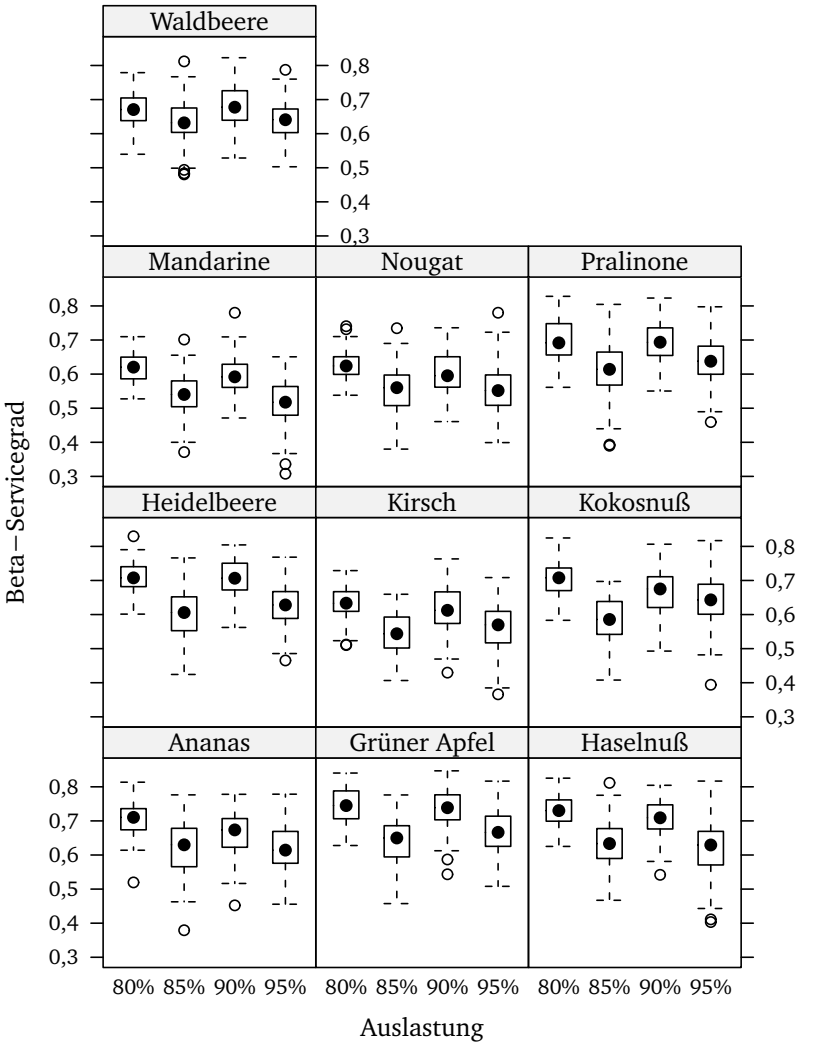


Abbildung 8.20: Produktbezogener Vergleich des β-Servicegrades unter verschiedenen Auslastungen (offene Produktion, konstante Lagerkostensätze, V=0,1, SB=0%)

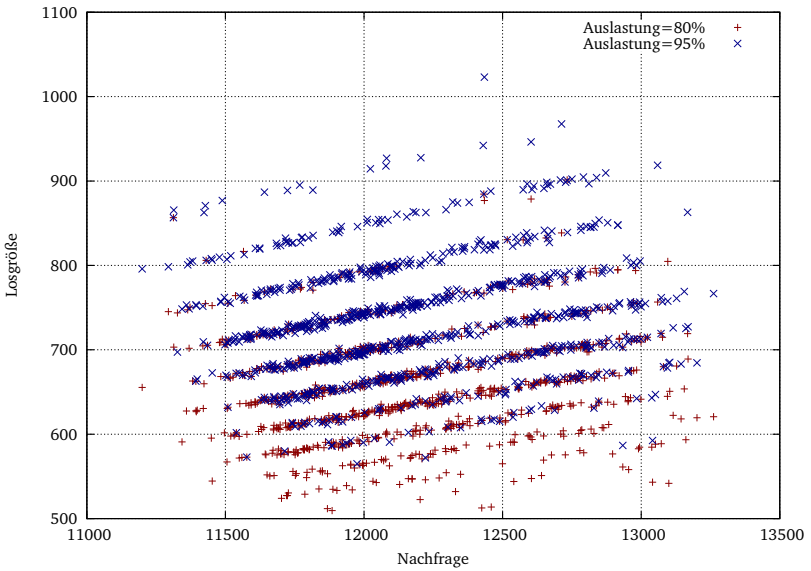


Abbildung 8.21: Abhängigkeit der Losgröße von der Nachfrage bei unterschiedlicher Auslastung (geschlossene Produktion, konstante Lagerkostensätze, $V=0{,}1$, $SB=0\%$)

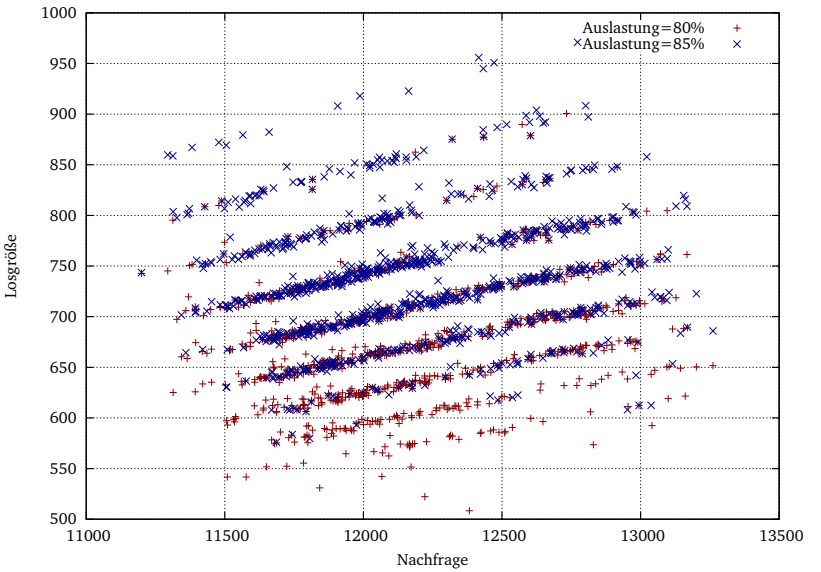


Abbildung 8.22: Abhängigkeit der Losgröße von der Nachfrage bei unterschiedlicher Auslastung (offene Produktion, konstante Lagerkostensätze, $V=0{,}1$, $SB=0\%$)

der Referenztestreihe bei offener Produktion zwischen einer Auslastung von 80% und 85%. Erkennbar ist die Verschiebung der mittleren Losgrößen aller Produkte in Richtung größerer Lose bei einer Auslastung von 85%. Diese starke Verschiebung läßt sich auch bei Einführung eines Sicherheitsbestandes beobachten. Wird die Auslastung weiter auf 90% erhöht, so sinken die mittleren Losgrößen, so daß höhere Servicegrade erreicht werden können. Dies stimmt auch mit den Beobachtungen aus dem vorangehenden Abschnitt überein.

8.8 Veränderungen der TBO/Lagerkostensatz

Als Letztes soll der Einfluß der TBO/Lagerkostensatz der Produkte untersucht werden. In allen vorangehenden Testreihen wurde davon

Produkt	h_j	sc_j	TBO
Ananas	8	1000	$\frac{1}{2}$
Grüner Apfel	18	1000	$\frac{1}{3}$
Haselnuß	2	1000	1
Heidelbeere	32	1000	$\frac{1}{4}$
Kirsch	8	1000	$\frac{1}{2}$
Kokosnuß	32	1000	$\frac{1}{4}$
Mandarine	18	1000	$\frac{1}{3}$
Nougat	18	1000	$\frac{1}{3}$
Pralinone	2	1000	1
Waldbeere	32	1000	$\frac{1}{4}$

Tabelle 8.12: Parameter bei Variation der TBO

ausgegangen, daß die TBO für alle Produkte identisch ist und bei $\frac{1}{4}$ der Periodenlänge liegt. Die Tabelle auf dieser Seite zeigt die Daten für die TBO, Lagerkostensätze und Fixkosten, die innerhalb dieser Testreihen verwendet werden. Die Rüstkosten sind für alle Produkte identisch geblieben. Durch Variation der TBO haben sich die Lagerkostensätze verändert, so daß es für einige Produkte jetzt günstiger ist, größere Lose zu fertigen, während andere Produkte in kleineren Losen gefertigt werden müssen. Da das Nachfragemuster und die stochastische Kundennachfrage identisch geblieben sind, lassen sich so die Einflüsse der unterschiedlichen Losgrößen auf den Servicegrad beobachten (vgl. Abbildung 8.24 auf Seite 327).

Die Abbildung auf der gegenüberliegenden Seite zeigt die Unterschiede zur Referenztestreihe bei geschlossener Produktion. Die Kurvenscharen sind erneut zu sehen, mit unterschiedlich starken Ausprägungen. Die neuen Kurvenscharen weisen eine Lage- und Formveränderung gegenüber der Referenztestreihe auf, die auf die veränderten Lagerkostensätze zurückzuführen sind. Ein Vergleich der Lageänderungen zeigt, daß diese für Produkte mit identischen Lagerkostensätzen auch in die gleiche Wirkungsrichtung gehen (z. B. Kokosnuß und Waldbeere

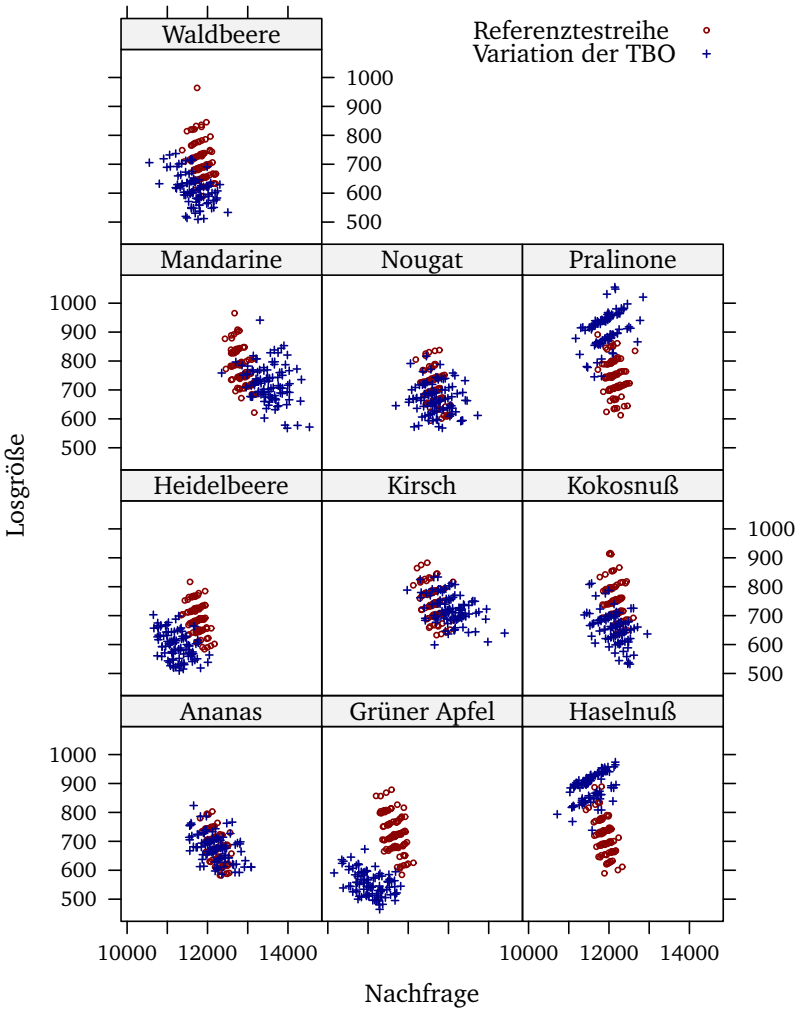


Abbildung 8.23: Abhängigkeit der Losgröße von der Nachfrage bei unterschiedlicher TBO (geschlossene Produktion, konstante Rüstkosten, $V=0{,}1$, $SB=0\%$)

bzw. Haselnuß und Pralinone). Die gleichen Auswirkungen lassen sich beobachten, wenn Sicherheitsbestände berücksichtigt werden. Auch der Übergang von der geschlossenen zur offenen Produktion zeigt, daß durch die Veränderung der Lagerkostensätze die Lage der Kurvenscharen verändert wird. Die Kostenstrukturen stellen somit der wesentliche Einflußfaktor für die Lage der Kurvenscharen dar.

Zusammen mit den Lageänderungen der Kurvenscharen verändern sich auch die mittleren Losgrößen je Produkt. Für Produkte wie Kokosnuß und Waldbeere verringern sich die Losgrößen, so daß bei der hier betrachteten geschlossenen Produktion die Flexibilität des Systems ansteigt, da die Ressourcen für eine kürzere Zeitspanne belegt werden. Der Servicegrad dieser Produkte steigt gegenüber der Referenztestreihe deutlich an. Sinkende Servicegrade lassen sich hingegen bei Haselnuß und Pralinone beobachten. Bei beiden Produkten steigt die mittlere Losgröße deutlich an und die damit auch die Zeit, bis ein Los zur Verfügung steht. Das System ist hier länger an die Planungsentscheidung gebunden, die Flexibilität sinkt und damit auch der Servicegrad.

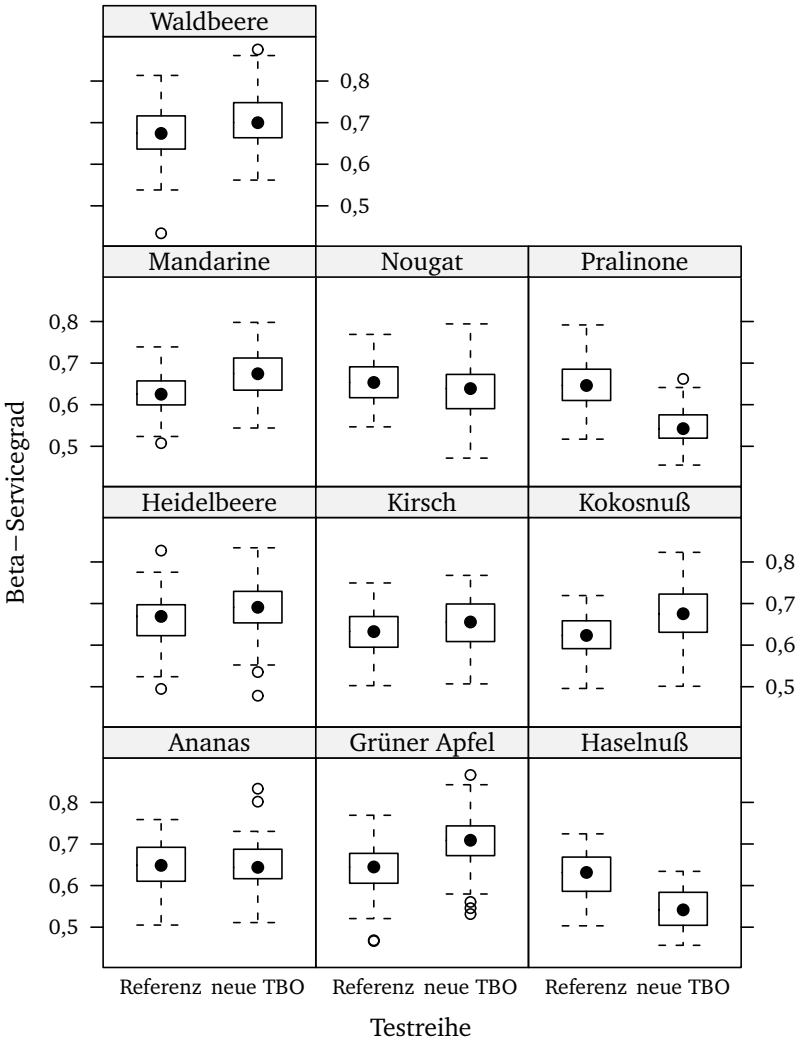


Abbildung 8.24: Produktbezogener Vergleich des β-Servicegrades unter verschiedenen TBO (geschlossene Produktion, konstante Rüstkosten, V=0,1, SB=0%)

Kapitel 9

Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Bereitstellung von Werkzeugen, um die Simulation kontinuierlicher Prozesse in der hierarchischen Produktionsplanung zu ermöglichen. Diese Aufgabe bestand aus drei aufeinander aufbauenden Teilen: Erzeugung von Testdaten, Simulation kontinuierlicher Prozesse, Verfahren zur Auswertung und Visualisierung der Daten.

Für den ersten Teil, die Erzeugung von Testdaten, wurden in dieser Arbeit standardisierte Methoden vorgestellt, die nicht nur für den Bereich der Losgrößenmodelle einsetzbar sind. Sowohl die Methoden, als auch die entwickelten Java-Klassen lassen sich allgemein auf unterschiedliche Arten von Testdaten anwenden. Speziell die Anwendung von Zufallszahlengeneratoren mit Teilströmen auf die Erzeugung von synthetische Testdaten läßt sich bei systematischen Untersuchungen sehr gut einsetzen. Ein entsprechender Generator für Losgrößenmodelle wurde in Kapitel 7 entwickelt. Die Möglichkeiten zur Erweiterung und Verbesserung der Qualität der Testdaten wurden detailliert in Abschnitt 7.3 vorgestellt.

Der zweite Bestandteil, die Simulation kontinuierlicher Prozesse, umfasste sowohl theoretische Grundlagen der Simulation, als auch die

Bereitstellung einer Referenzimplementierung in Java. Die Entwicklung der Algorithmen zur Simulation kontinuierliche Prozesse stellt eine Neuerung dar, die sowohl die Ansprüche an die Genauigkeit erfüllt, als auch eine geringe Rechenleistung benötigt. Im Bereich der Produktionsplanung, mit einem Planungshorizont von z. B. einem Jahr, kann so eine hohe Anzahl an Wiederholungen simuliert werden, bei geringem Einsatz von Rechenleistung. Eine universell gehaltene Klassenstruktur ermöglicht die Simulation von Ein-/Mehrproduktanlagen mit beliebiger Erzeugnisstruktur. Somit ist auch für zukünftige Untersuchungen sichergestellt, daß diese eine einheitliche und erprobte Simulationsbibliothek einsetzen können.

Im dritten Teil dieser Arbeit wurden Ergebnisse der ersten beiden Teile auf ein Problem der hierarchischen Produktionsplanung angewandt. Im Rahmen der Auswertung wurde eine andere Art der Visualisierung vorgestellt, welche die charakteristischen Merkmale eine Planungssystems als „Fingerabdruck“ zeigt. Dieser beschreibt die Zusammenhänge zwischen Eingabe- und Ausgabegrößen und ermöglicht eine anschauliche Betrachtung der Auswirkungen von Veränderungen in den Eingabedaten.

Die beobachteten Muster des β-Servicegrades wurden in dieser Form bisher noch nicht beschrieben. Es stellt sich die Frage, welche anderen Systemen entsprechende charakteristische Muster besitzen. Es ist wahrscheinlich, das diese nicht nur bei einstufigen Anlagen, sondern auch bei mehrstufigen Anlagen auftreten. Außer komplexeren Anlagen sollten auch einfache Lagerbestandssysteme (wie z. B. (s,S)-Politiken) untersucht werden. Eine entsprechende Klassifikation und Abbildung der Muster von einfachen Anlagen könnte helfen, die Muster der komplexeren Anlagen zu interpretieren.

Die Untersuchungen haben gezeigt, daß der Einsatz der neuen Simulationsumgebung zu neuen Fragestellungen führt, die auf Effekten beruhen, die in dieser Form noch nicht beobachtet werden konnten. Für zukünftige Untersuchungen bietet dies ausreichend Platz, um z. B. folgende Fragen zu beantworten:

- Was sind die Ursachen für die unterschiedlichen Servicegrade bei

den Produkten aus der Referenztestreihe (vgl. Abbildung 8.2 auf Seite 285)?

- Was sind die Ursachen für die Ausbildung der charakteristischen Strukturen, die im Fingerabdruck des Planungssystems zu beobachten sind (vgl. Abbildung 8.6 auf Seite 296)?
- Warum überlagern sich die Strukturen aller Produkte zu einer gemeinsamen Struktur?
- Weshalb reagiert der Servicegrad auf veränderte Strafkosten erst ab einer bestimmten Variabilität, abhängig von der Kostenstruktur (vgl. Tabelle 8.11b auf Seite 311)?
- Welche Parameter haben Einfluß auf die Anomalie bei Erhöhung der Kapazitätsauslastung (vgl. Abbildung 8.20 auf Seite 321)?

Die Antworten auf diese Fragen könnten nicht nur helfen, die Auswirkungen der Parameter auf die Planungssysteme besser zu verstehen, sondern diese auch gezielt einzusetzen, um die Servicegrade zu verbessern.

Literaturverzeichnis

[Absi u. Kedad-Sidhoum 2008] Absi, Nabil ; Kedad-Sidhoum, Safia: The multi-item capacitated lot-sizing problem with setup times and shortage costs. In: *European Journal of Operational Research* 185 (2008), Nr. 3, S. 1351–1374

[Achterberg u. a. 2009] Achterberg, Tobias ; Koch, Thorsten ; Martin, Alexander: *MIPLIB*. http://miplib.zib.de/. Version: 2009. – abgerufen am 27.10.2009

[Ahrens u. Dieter 1982] Ahrens, Joachim H. ; Dieter, Ulrich: Generating Gamma Variates by a Modified Rejection Technique. In: *Communications of the ACM* 25 (1982), Nr. 1, S. 47–54

[Ahrens u. Dieter 1988] Ahrens, Joachim H. ; Dieter, Ulrich: Efficient Table-Free Sampling Methods for the Exponential, Cauchy, and Normal Distributions. In: *Communications of the ACM* 31 (1988), Nr. 11, S. 1330–1337

[Anderson u. Lagodimos 1989] Anderson, E.J. ; Lagodimos, A.G.: Service levels in single-stage MRP systems with demand uncertainty. In: *Engineering Costs and Production Economics* 17 (1989), Nr. 1-4, S. 125–133

[Anthony 1965] Anthony, Robert N.: *Planning and Control Systems; a Framework for Analysis*. 9. Boston : Division of Research, Graduate School of Business Administration, Harvard University, 1965

[Axsäter 2006] Axsäter, Sven: *Inventory Control*. 2. Berlin : Springer, 2006

[Baker 1977] Baker, Kenneth R.: An Experimental Study of the Effectiveness of Rolling Schedules in Production Planning. In: *Decision Sciences* 8 (1977), Nr. 1, S. 19–27. http://dx.doi.org/10.1111/j.1540-5915.1977.tb01065.x. – DOI 10.1111/j.1540–5915.1977.tb01065.x

[Baker u. Peterson 1979] Baker, Kenneth R. ; Peterson, David W.: An Analytic Framework for Evaluating Rolling Schedules. In: *Management Science* 25 (1979), Nr. 4, S. 341–351

[Balzert 2000] Balzert, Helmut: *Lehrbuch der Softwaretechnik - Softwareentwicklung*. 2. Heidelberg : Spektrum Akademischer Verlag, 2000

[Balzert 2008] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Softwaremanagement*. 2. Heidelberg : Spektrum Akademischer Verlag, 2008

[Banks u. a. 2000] Banks, Jerry ; Carson II, John S. ; Nelson, Barry L. ; Nicol, David M.: *Discrete-Event System Simulation*. 4. Upper Saddle River, NJ : Prentice Hall, 2000

[Barr u. Haas 2009] Barr, Rimon ; Haas, Zygmund J.: *Java in Simulation Time - JiST*. http://jist.ece.cornell.edu/. Version: 2009. – abgerufen am 27.10.2009

[Belveaux u. Wolsey 2001] Belveaux, Gaetan ; Wolsey, Laurence A.: Modelling Practical Lot-Sizing Problems as Mixed-Integer Programs. In: *Management Science* 47 (2001), Nr. 7, S. 993–1007

[Best 1983] Best, D.J.: A Note on Gamma Variate Generators with Shape Parameter less than Unity. In: *Computing* 30 (1983), Nr. 2, S. 185–188

[Billington u. a. 1983] Billington, Peter J. ; McClain, John O. ; Thomas, L. J.: Mathematical Programming Approaches to Capacity-Constrained MRP Systems: Review, Formulation and Problem Reduction. In: *Management Science* 29 (1983), Nr. 10, S. 1126–1141

[Bixby 2002] Bixby, Robert E.: Solving real-world linear programs: a decade and more of progress. In: *Operations Research* 50 (2002), Nr. 1, S. 3–15

[Blackburn u. Millen 1982] Blackburn, Joseph D. ; Millen, Robert A.: The impact of a rolling schedule in a multi-level MRP system. In: *Journal of Operations Management* 2 (1982), Nr. 2, S. 125–135. http://dx.doi.org/10.1016/0272-6963(82)90028-6. – DOI 10.1016/0272–6963(82)90028–6

[Bloch 2008] Bloch, Joshua: *Effective Java: A Programming Language Guide*. 2. Upper Saddle River, NJ : Addison-Wesley, 2008

[Bloech u. a. 2007] Bloech, Jürgen ; Bogaschewsky, Ronald ; Buscher, Udo ; Daub, Anke ; Götze, Uwe ; Roland, Folker: *Einführung in die Produktion*. 6. Berlin, Heidelberg : Springer, 2007

[Blömer 1999] Blömer, Ferdinand ; Günther, H.O. (Hrsg.): *Produktionsplanung und -steuerung in der chemischen Industrie*. Wiesbaden : Deutscher Universitäts Verlag, 1999 (Gabler Edition Wissenschaft, Produktion und Logistik)

[Bodin u. a. 1983] Bodin, L. ; Golden, B. ; Assad, A. ; Ball, M.: Routing and Scheduling of Vehicles and Crews - The State of the Art. In: *Computers & Operations Research* 10 (1983), Nr. 2, S. 63–211

[Boulaksil u. a. 2009] Boulaksil, Youssef ; Fransoo, Jan ; Halm, Ernico van: Setting safety stocks in multi-stage inventory systems under rolling horizon mathematical programming models. In: *OR Spectrum* 31 (2009), Nr. 1, S. 121–140. http://dx.doi.org/10.1007/s00291-007-0086-3. – DOI 10.1007/s00291–007–0086–3

[Bronstein u. a. 2003] Bronstein, I.N. ; Semendjajew, K.A. ; Grosche, G. ; Ziegler, V. ; Ziegler, D. ; Zeidler, E. (Hrsg.): *Teubner-Taschenbuch der Mathematik*. 2. Stuttgart, Leipzig, Wiesbaden : B.G. Teubner, 2003

[Bronstein u. a. 1999] Bronstein, I.N. ; Semendjajew, K.A. ; Musiol, G. ; Mühlig, H.: *Taschenbuch der Mathematik*. Frankfurt am Main : Harri Deutsch, 1999

[Buchmann 2008] Buchmann, Johannes: *Einführung in die Kryptographie*. 4. Berlin, Heidelberg : Springer, 2008

[Buzacott u. Shanthikumar 1994] Buzacott, J.A. ; Shanthikumar, J.G.: Safety Stock Versus Safety Time in MRP Controlled Production Systems. In: *Management Science* 40 (1994), Nr. 12, S. 1678–1689

[Carlson u. a. 1982] Carlson, Robert C. ; Beckman, Sara L. ; Kropp, Dean H.: The Effectiveness of extending the Horizon in Rolling Production Scheduling. In: *Decision Sciences* 13 (1982), Nr. 1, S. 129–146. http://dx.doi.org/10.1111/j.1540-5915.1982.tb00136.x. – DOI 10.1111/j.1540–5915.1982.tb00136.x

[Cellier 1991] Cellier, François E.: *Continuous System Modeling*. New York, Berlin, Heidelberg : Springer, 1991

[CERN 2004] CERN: *Colt Project - Open Source Libraries for High Performance Scientific and Technical Computing in Java*. http://acs.lbl.gov/~hoschek/colt/. Version: 2004. – abgerufen am 27.10.2009

[Chang 1985] Chang, C. A.: The interchangeability of safety stocks and safety lead time. In: *Journal of Operations Management* 6 (1985), Nr. 1, S. 35–42. http://dx.doi.org/10.1016/0272-6963(85)90033-6. – DOI 10.1016/0272–6963(85)90033–6

[Chubanov u. a. 2008] Chubanov, Sergei ; Kovalyov, Mikhail Y. ; Pesch, Erwin: A single-item economic lot-sizing problem with a non-uniform resource: Approximation. In: *European Journal of Operational Research* 189 (2008), Nr. 3, S. 877–889. http://dx.doi.org/10.1016/j.ejor.2007.02.058. – DOI 10.1016/j.ejor.2007.02.058

[Clark u. Scarf 1960] Clark, Andrew J. ; Scarf, Herbert: Optimal Policies for a Multi-Echelon Inventory Problem. In: *Management Science* 6 (1960), Nr. 4, S. 475–490

[Cordeau u. a. 2002] Cordeau, J-F ; Gendreau, M. ; Laporte, G. ; Potvin, J-Y ; Semet, F.: A Guide to Vehicle Routing Heuristics. In: *Journal of the Operational Research Society* 53 (2002), Nr. 5, S. 512–522

[Cordeau u. a. 2001] Cordeau, J-F ; Laporte, G. ; Mercier, A.: A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows. In: *Journal of the Operational Research Society* 52 (2001), Nr. 8, S. 928–936

[Cormen u. a. 2007] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Algorithmen - Eine Einführung*. 2., korr. A. München, Wien : Oldenbourg Verlag, 2007

[Crowder u. a. 1978] Crowder, Harlan P. ; Dembo, Ron S. ; Mulvey, John M.: Reporting computational experiments in mathematical programming. In: *Mathematical Programming* 15 (1978), Nr. 1, S. 316–329. http://dx.doi.org/10.1007/BF01609036. – DOI 10.1007/BF01609036

[Dautelle 2007] Dautelle, Jean M.: *JScience*. http://www.jscience.org. Version: 2007. – abgerufen am 27.10.2009

[de Bodt u. a. 1982] de Bodt, Marc A. ; van Wassenhove, Luk N. ; Gelders, Ludo F.: Lot sizing and safety stock decisions in an MRP system with demand uncertainty. In: *Engineering Costs and Production Economics* 6 (1982), S. 67–75. http://dx.doi.org/10.1016/0167-188X(82)90039-8. – DOI 10.1016/0167–188X(82)90039–8

[Demeulemeester u. a. 2003] Demeulemeester, Erik ; Vanhoucke, Mario ; Herroelen, Willy: RanGen: A Random Network Generator for Activity-on-the-Node Networks. In: *Journal of Scheduling* 6 (2003), Nr. 1, S. 17–38. http://dx.doi.org/10.1023/A:1022283403119. – DOI 10.1023/A:1022283403119

[Dillenberger u. a. 1994] Dillenberger, Christof ; Escudero, Laureano F. ; Wollensak, Artur ; Zhang, Wu: On practical resource allocation for production planning and scheduling with period overlapping setups. In: *European Journal of Operational Research* 75 (1994), Nr. 2, S. 275–286. http://dx.doi.org/10.1016/0377-2217(94)90074-4. – DOI 10.1016/0377–2217(94)90074–4

[Dixon u. Silver 1981] Dixon, Paul S. ; Silver, Edward A.: A heuristic solution procedure for the multi-item, single-level, limited capacity, lot-sizing problem. In: *Journal of Operations Management* 2 (1981), Nr. 1, S. 23–39. http://dx.doi.org/10.1016/0272-6963(81)90033-4. – DOI 10.1016/0272–6963(81)90033–4

[Drexl u. Haase 1995] Drexl, Andreas ; Haase, Knut: Proportional lotsizing and scheduling. In: *International Journal of Production Economics* 40 (1995), Nr. 1, S. 73–87

[Duller 2008] Duller, Christine: *Einführung in die nichtparametrische Statistik mit SAS und R: Ein anwendungsorientiertes Lehr- und Arbeitsbuch*. 1. Heidelberg : Physica-Verlag, 2008

[Dyckhoff 2006] Dyckhoff, Harald: *Produktionstherorie*. 5. Berlin, Heidelberg : Springer, 2006

[Eker u. a. 2003] Eker, J. ; Janneck, J.W. ; Lee, E.A. ; Liu, Jie ; Liu, Xiaojun ; Ludvig, J. ; Neuendorffer, S. ; Sachs, S. ; Xiong, Yuhong: Taming heterogeneity - the Ptolemy approach. In: *Proceedings of the IEEE* 91 (2003), Januar, Nr. 1, S. 127–144. http://dx.doi.org/10.1109/JPROC.2002.805829. – DOI 10.1109/JPROC.2002.805829

[Entacher 1998] Entacher, Karl: Bad subsequences of well-known linear congruential pseudorandom number generators. In: *ACM Transactions on Modeling and Computer Simulation* 8 (1998), Nr. 1, S. 61–70. http://dx.doi.org/10.1145/272991.273009. – DOI 10.1145/272991.273009

[Eppen u. Martin 1987] Eppen, Gary D. ; Martin, R. K.: Solving Multi-Item Capacitated Lot-Sizing Problems Using Varibale Redefinition. In: *Operations Research* 35 (1987), Nr. 6, S. 832–848

[Evans 2004] Evans, Eric: *Domain-Driven Design*. 1. Boston, San Francisco : Addison-Wesley, 2004

[FairIsaac Corporation 2008] FairIsaac Corporation: *Xpress-Mosel, Reference Manual*. 2.4. Leamington Spa, UK, Juni 2008

[Federgruen u. Tzur 1994] Federgruen, Awi ; Tzur, Michal: Minimal Forecast Horizons and a New Planning Procedure for the General Dynamic Lot Sizing Model: Nervousness Revisited. In: *Operations Research* 42 (1994), Nr. 3, S. 456–468

[Fishman 1976] Fishman, George S.: Sampling from the Gamma Distribution on a Computer. In: *Communications of the ACM* 19 (1976), Nr. 7, S. 407–409

[Fishman 2001] Fishman, George S.: *Discrete-Event Simulation: Modeling, Programming, and Analysis: Modeling, Programming and Analysis*. 1. Berlin, New York : Springer, 2001 (Springer Series in Operations Research)

[Fleischmann 1990] Fleischmann, Bernhard: The discrete lot-sizing and scheduling problem. In: *European Journal of Operational Research* 44 (1990), Nr. 3, S. 337–348

[Fleischmann 1994] Fleischmann, Bernhard: The discrete lot-sizing and scheduling problem with sequence-dependent setup costs. In: *European Journal of Operational Research* 75 (1994), Nr. 2, S. 395–404

[Fleischmann u. Meyr 1997] Fleischmann, Bernhard ; Meyr, Herbert: The general lotsizing and scheduling problem. In: *OR Spectrum* 19 (1997), Nr. 1, S. 11–21. http://dx.doi.org/10.1007/BF01539800. – DOI 10.1007/BF01539800

[Fleischmann u. a. 2008] Fleischmann, Bernhard ; Meyr, Herbert ; Wagner, Michael: Advanced Planning. In: Stadtler, Hartmut (Hrsg.) ; Kilger, Christoph (Hrsg.): *Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies*. 4. Berlin : Springer, 2008, S. 81–106

[Fowler 1999] Fowler, Martin: *Refactoring: Improving the Design of Existing Code*. Boston : Addison-Wesley, 1999 (Object Technology Series)

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph E.: *Design Patterns. Elements of Reusable Object-Oriented Software.* 1, Reprint. Boston : Addison-Wesley, 1995

[Gau u. Wäscher 1995] Gau, T. ; Wäscher, G.: CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem. In: *European Journal of Operational Research* 84 (1995), Nr. 3, S. 572–579. http://dx.doi.org/10.1016/0377-2217(95)00023-J. – DOI 10.1016/0377–2217(95)00023–J. – ISSN 0377–2217

[Gomory 1958] Gomory, Ralph E.: Outline of an algorithm for integer solutions to linear programs. In: *Bulletin of the American Mathematical Society* 64 (1958), Nr. 5, S. 275–278

[Gopalakrishnan u. a. 2001] Gopalakrishnan, Mohan ; Ding, Ke ; Bourjolly, Jean-Marie ; Mohan, Srimathy: A Tabu-Search Heuristic for the Capacitated Lot-Sizing Problem with Set-up Carryover. In: *Management Science* 47 (2001), Nr. 6, S. 851–863

[Hall u. Posner 2001] Hall, Nicholas G. ; Posner, Marc E.: Generating experimental data for computational testing with machine scheduling applications. In: *Operations Research* 49 (2001), Nr. 7, S. 854–865

[Haramoto u. a. 2008] Haramoto, Hiroshi ; Matsumoto, Makoto ; Nishimura, Takuji ; Panneton, Francois ; L'Ecuyer, Pierre: Efficient Jump Ahead for F2-Linear Random Number Generators. In: *INFORMS Journal on Computing* 20 (2008), Nr. 3, S. 385–390. http://dx.doi.org/10.1287/ijoc.1070.0251. – DOI 10.1287/ijoc.1070.0251

[Harris 1913] Harris, Ford W.: How Many Parts to Make at Once. In: *Factory, The Magazine of Management* 10 (1913), Nr. 2, S. 135–136,152

[Harris 1990] Harris, Ford W.: How Many Parts to Make at Once. In: *Operations Research* 38 (1990), Nr. 6, S. 947–950

[Hartung 2002] Hartung, Joachim: *Statistik*. Bd. 13. München, Wien : Oldenbourg Verlag, 2002

[Hax u. Meal 1975] *Kapitel* Hierarchical integration of production planning and scheduling. In: Hax, Arnoldo C. ; Meal, Harlan C.: *Logistics*. Amsterdam : North-Holland Publishing Co., 1975 (North-Holland/TIMS studies in the management sciences), S. 53–70

[Heike u. Târcolea 2000] Heike, Hans-Dieter ; Târcolea, Constantin: *Grundlagen der Statistik und Wahrscheinlichkeitsrechnung, Statistik I*. 2. München, Wien : Oldenbourg Verlag, 2000

[Hellekalek 1998] Hellekalek, P.: Good random number generators are (not so) easy to find. In: *Mathematics and Computers in Simulation* 46 (1998), Nr. 5-6, S. 485–505

[Hillier u. Lieberman 2007] Hillier, Frederick S. ; Lieberman, Gerald J.: *Introduction to Operations Research*. 8. New York : McGraw-Hill, 2007

[Hooker 1994] Hooker, J. N.: Needed: An Empirical Science of Algorithms. In: *Operations Research* 42 (1994), Nr. 2, S. 201–212. – ISSN 0030364X

[Hopp u. Spearman 2001] Hopp, Wallace J. ; Spearman, Mark L.: *Factory Physics: Foundations of Manufacturing Management*. 2. Boston : Irwin/McGraw-Hill, 2001

[IEEE 1985] IEEE: *IEEE Standard for Binary Floating-Point Arithmetic*. 1985

[ISI 2009] ISI: *The network simulator - ns-2*. http://www.isi.edu/nsnam/ns/. Version: 2009. – abgerufen am 27.10.2009

[Jackson u. a. 1991] Jackson, Richard H. F. ; Boggs, Paul T. ; Nash, Stephen G. ; Powell, Susan: Guidelines for reporting results of computational experiments. Report of the ad hoc committee. In: *Mathematical Programming* 49 (1991), Nr. 1, S. 413–425. http://dx.doi.org/10.1007/BF01588801. – DOI 10.1007/BF01588801

[Jacobs u. a. 2002] Jacobs, P.H.M. ; Lang, N.A. ; Verbraeck, A.: D-sol; A distributed Java based discrete event simulation architecture. In: Yucesan, E. (Hrsg.) ; Chen, C.H. (Hrsg.) ; Snowdon, J.L. (Hrsg.) ; Charnes, J.M. (Hrsg.): *Proceedings of the 2002 Winter Simulation Conference, Vols 1 and 2*. New York : IEEE, 2002, S. 793–800

[Jung u. a. 2004] Jung, June Y. ; Blau, Gary ; Pekny, Joseph F. ; Reklaitis, Gintaras V. ; Eversdyk, David: A simulation based optimization approach to supply chain management under demand uncertainty. In: *Computers & Chemical Engineering* 28 (2004), Nr. 10, S. 2087–2106. http://dx.doi.org/10.1016/j.compchemeng.2004.06.006. – DOI 10.1016/j.compchemeng.2004.06.006

[Jung u. a. 2008] Jung, June Y. ; Blau, Gary ; Pekny, Joseph F. ; Reklaitis, Gintaras V. ; Eversdyk, David: Integrated safety stock management for multi-stage supply chains under production capacity constraints. In: *Computers & Chemical Engineering* 32 (2008), Nr. 11, S. 2570–2581. http://dx.doi.org/10.1016/j.compchemeng.2008.04.003. – DOI 10.1016/j.compchemeng.2008.04.003

[Kallehauge 2008] Kallehauge, Brian: Formulations and exact algorithms for the vehicle routing problem with time windows. In: *Computers & Operations Research* 35 (2008), Nr. 7, S. 2307–2330. http://dx.doi.org/10.1016/j.cor.2006.11.006. – DOI 10.1016/j.cor.2006.11.006

[Karmarkar u. Schrage 1985] Karmarkar, Uday S. ; Schrage, Linus: The Deterministic Dynamic Product Cycling Problem. In: *Operations Research* 33 (1985), Nr. 2, S. 326–345

[Kelton 2006] *Kapitel* Random Number Generation. In: Kelton, W. D.: *Implementing Representations of Uncertainty*. Amsterdam : Elsevier Science, 2006 (Elsevier Handbooks in Operations Research and Management Science), S. 181–191

[Kerievsky 2005] Kerievsky, Joshua: *Refactoring to Patterns*. 1. München : Addison-Wesley, 2005

[Kimms u. Müller-Bungart 2007] Kimms, Alf ; Müller-Bungart, Michael: Simulation of stochastic demand data streams for network revenue management problems. In: *OR Spectrum* 29 (2007), Nr. 1, S. 5–20. http://dx.doi.org/10.1007/s00291-005-0020-5. – DOI 10.1007/s00291–005–0020–5

[Knuth 1997] Knuth, Donald E.: *The Art of Computer Programming*. Bd. 2: *Seminumerical Algorithms*. 3. Reading, Massachusetts : Addison-Wesley, 1997

[Koh 2004] Koh, S. C. L.: MRP-Controlled Batch-Manufacturing Environment under Uncertainty. In: *Journal of the Operational Research Society* 55 (2004), Nr. 3, S. 219–232

[Kolisch u. Sprecher 1996] Kolisch, Rainer ; Sprecher, Arno: PSPLIB - A project scheduling problem library : OR Software - ORSEP Operations Research Software Exchange Program. In: *European Journal of Operational Research* 96 (1996), Nr. 1, S. 205–216. http://dx.doi.org/10.1016/S0377-2217(96)00170-1. – DOI 10.1016/S0377–2217(96)00170–1. – ISSN 0377–2217

[Kolisch u. a. 1995] Kolisch, Rainer ; Sprecher, Arno ; Drexl, Andreas: Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. In: *Management Science* 41 (1995), Nr. 10, S. 1693–1703

[Koskosidis u. a. 1992] Koskosidis, Yiannis A. ; Powell, Warren B. ; Solomon, Marius M.: An Optimization-Based Heuristic for Vehicle Routing and Scheduling with Soft Time Window Constratraints. In: *Transportation Science* 26 (1992), Nr. 2, S. 69–85

[Kuhn u. Gebhard 2008] Kuhn, Heinrich ; Gebhard, Marina: Hierarchische Produktions- und Logistikplanung bei unvollkommenen Informationen. In: *Zeitschrift für Betriebswirtschaft* 78 (2008), Nr. 4, S. S99–S124

[Kurz u. Askin 2001] Kurz, M. E. ; Askin, R. G.: Heuristic scheduling of parallel machines with sequence-dependent set-up times. In: *International Journal of Production Research* 39 (2001), Nr. 16, S. 3747–3769

[Laundy u. a. 2009] Laundy, Richard ; Perregaard, Michael ; Tavares, Gabriel ; Tipi, Horia ; Vazacopoulos, Alkis: Solving Hard Mixed-Integer Programming Problems with Xpress-MP: A MIPLIB 2003 Case Study. In: *INFORMS Journal on Computing* (2009), Nr. 2, S. 304–313. http://dx.doi.org/10.1287/ijoc.1080.0293. – DOI 10.1287/ijoc.1080.0293

[Law u. Kelton 2000] Law, Averill M. ; Kelton, W.David: *Simulation Modeling and Analysis*. 3. Boston, London : McGraw-Hill, 2000

[L'Ecuyer u. Côté 1991] L'Ecuyer, P. ; Côté, S.: Implementing A Random Number Package With Splitting Facilities. In: *ACM Transactions on Mathematical Software* 17 (1991), Nr. 1, S. 98–111

[L'Ecuyer 1988] L'Ecuyer, Pierre: Efficient and portable combined random number generators. In: *Communications of the ACM* 31 (1988), Nr. 6, S. 742–751. http://dx.doi.org/10.1145/62959.62969. – DOI 10.1145/62959.62969

[L'Ecuyer 1994] L'Ecuyer, Pierre: Uniform random number generation. In: *Annals of Operations Research* 53 (1994), Nr. 1-4, S. 77–120

[L'Ecuyer 1999] L'Ecuyer, Pierre: Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators. In: *Operations Research* 47 (1999), Nr. 1, S. 159–164

[L'Ecuyer 2006] *Kapitel* Random Number Generation. In: L'Ecuyer, Pierre: *Simulation*. Amstaerdam : Elsevier Science, 2006 (Elsevier

Handbooks in Operations Research and Management Science), S. 55–81

[L'Ecuyer 2009] L'Ecuyer, Pierre: *Webseite*. Online. http://www.iro.umontreal.ca/~lecuyer/myftp/streams00/java/. Version: 2009. – abgerufen am 01.08.2009

[L'Ecuyer u. Andres 1997] L'Ecuyer, Pierre ; Andres, Terry H.: A random number generator based on the combination of four LCGs. In: *Mathematics and Computers in Simulation* 44 (1997), Nr. 1, S. 99–107. http://dx.doi.org/10.1016/S0378-4754(97)00052-9. – DOI 10.1016/S0378–4754(97)00052–9

[L'Ecuyer u. a. 2002a] L'Ecuyer, Pierre ; Meliani, Lakhdar ; Vaucher, Jean: SSJ: a framework for stochastic simulation in Java. In: *WSC '02: Proceedings of the 34th conference on Winter simulation*, Winter Simulation Conference, 2002, S. 234–242

[L'Ecuyer u. a. 2002b] L'Ecuyer, Pierre ; Simard, Richard ; Chen, E. J. ; Kelton, W. D.: An object-oriented random-number package with many long streams and substreams. In: *Operations Research* 50 (2002), Nr. 6, S. 1073–1075

[Lee u. a. 1997] Lee, Chung-Yee ; Lei, Lei ; Pinedo, Michael: Current trends in deterministic scheduling. In: *Annals of Operations Research* 70 (1997), Nr. 1-4, S. 1–41

[Loos 1997] Loos, Peter ; Scheer, A.W. (Hrsg.): *Schriften zur EDV-orientierten Betriebswirtschaft*. Bd. 1: *Produktionslogistik in der chemischen Industrie*. Wiesbaden : Gabler, 1997

[Marchand u. a. 2002] Marchand, Hugues ; Martin, Alexander ; Weismantel, Robert ; Wolsey, Laurence: Cutting planes in integer and mixed Integer programming. In: *Discrete Applied Mathematics* 123 (2002), Nr. 1-3, S. 397–446

[Matsumoto u. Nishimura 1998] Matsumoto, Makoto ; Nishimura, Takuji: Mersenne twister: a 623-dimensionally equidistributed uniform

pseudo-random number generator. In: *ACM Transactions on Modeling and Computer Simulation* 8 (1998), Nr. 1, S. 3–30. http://dx.doi.org/10.1145/272991.272995. – DOI 10.1145/272991.272995

[McGeoch 1996] McGeoch, Catherine C.: Toward an Experimental Method for Algorithm Simulation. In: *INFORMS Journal on Computing* 8 (1996), Nr. 1, S. 1–15. – ISSN 10919856

[Méndez u. a. 2006] Méndez, Carlos A. ; Cerdá, Jaime ; Grossmann, Ignacio E. ; Harjunkoski, Iiro ; Fahl, Marco: State-of-the-art review of optimization methods for short-term scheduling of batch processes. In: *Computers & Chemical Engineering* 30 (2006), Nr. 6-7, S. 913–946. http://dx.doi.org/10.1016/j.compchemeng.2006.02.008. – DOI 10.1016/j.compchemeng.2006.02.008

[Meyr u. a. 2008] Meyr, Herbert ; Wagner, Michael ; Rohde, Jens: Structure of Advanced Planning Systems. In: Stadtler, Hartmut (Hrsg.) ; Kilger, Christoph (Hrsg.): *Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies*. 4. Berlin : Springer, 2008, S. 109–115

[Miller u. a. 2003] Miller, A.J. ; Nemhauser, G.L. ; Savelsbergh, M.W.P.: A multi-item production planning model with setup times: algorithms, reformulations, and polyhedral characterizations for a special case. In: *Mathematical Programming* 95 (2003), Nr. 1, S. 71–90. http://dx.doi.org/10.1007/s10107-002-0340-z. – DOI 10.1007/s10107–002–0340–z

[Miller u. Wolsey 2003] Miller, Andrew J. ; Wolsey, Laurence A.: Tight MIP Formulations for Multi-Item Discrete Lot-Sizing Problems. In: *Operations Research* 51 (2003), Nr. 4, S. 557–565

[Mula u. a. 2006] Mula, J. ; Poler, R. ; García-Sabater, J.P. ; Lario, F.C.: Models for production planning under uncertainty: A review. In: *International Journal of Production Economics* 103 (2006), Nr. 1, S. 271–285. http://dx.doi.org/10.1016/j.ijpe.2005.09.001. – DOI 10.1016/j.ijpe.2005.09.001

[Murray-Smith 1995] Murray-Smith, D.J.: *Continuous System Simulation*. 1. London : Chapman & Hall, 1995

[Muyldermans u. a. 2005] Muyldermans, Luc ; Beullens, Patrick ; Cattrysse, Dirk ; Van Oudheusden, Dirk: Exploring Variants of 2-Opt and 3-Opt for the General Routing Problem. In: *Operations Research* 53 (2005), Nr. 6, S. 982–995. http://dx.doi.org/10.1287/opre.1040.0205. – DOI 10.1287/opre.1040.0205

[Object Management Group 2009] Object Management Group: OMG Unified Modeling LanguageTM (OMG UML), Superstructure. 2009 (2.2). – Forschungsbericht

[Or 1976] Or, Ilhan: *Traveling Salesman Type Combinatorial Problems And Their Relation To The Logistics Of Regional Blood Banking*. Chicago, Northwestern University, Diss., 1976

[Or u. Pierskalla 1979] Or, Ilhan ; Pierskalla, William P.: A Transportation Location-Allocation Model for Regional Blood Banking. In: *IIE Transactions* 11 (1979), Nr. 2, S. 86–95

[Orlicky 1975] Orlicky, Joseph: *Material Requirements Planning; the New Way of Life in Production and Inventory Management*. New York : McGraw-Hill, 1975

[Page u. Kreutzer 2005] Page, Bernd (Hrsg.) ; Kreutzer, Wolfgang (Hrsg.): *Simulation Discrete Event Systems with UML and Java*. Bd. 1. Aachen : Shaker Verlag, 2005

[Pidd 2004] Pidd, Michael: *Computer Simulation in Management Science*. 5. Wiley, 2004

[Pinedo 2002] Pinedo, Michael: *Scheduling: Theory, Algorithms, and Systems*. 2. Upper Saddle River, NJ : Prentice Hall, 2002

[Pochet u. Wolsey 1988] Pochet, Yves ; Wolsey, Laurence A.: Lot-size models with backlogging: strong reformulations and cutting planes. In: *Mathematical Programming* 40 (1988), Nr. 1-3, S. 317–335

[Pochet u. Wolsey 2006] Pochet, Yves ; Wolsey, Laurence A. ; Mikosch, Thomas V. (Hrsg.) ; Resnick, Sidney I. (Hrsg.) ; Robinson, Stephen M. (Hrsg.): *Production Planning by Mixed Integer Programming*. 1. New York : Springer, 2006 (Springer Series in Operations Research and Financial Engineering)

[Potvin u. Bengio 1996] Potvin, Jean-Yves ; Bengio, Samy: The Vehicle Routing Problem with Time Windows Part II: Genetic Search. In: *INFORMS Journal on Computing* 8 (1996), Nr. 2, S. 165–172. http://dx.doi.org/10.1287/ijoc.8.2.165. – DOI 10.1287/ijoc.8.2.165

[Potvin u. a. 1996] Potvin, Jean-Yves ; Kervahut, Tanguy ; Garcia, Bruno-Laurent ; Rousseau, Jean-Marc: The Vehicle Routing Problem with Time Windows Part I: Tabu Search. In: *INFORMS Journal on Computing* 8 (1996), Nr. 2, S. 158–164. http://dx.doi.org/10.1287/ijoc.8.2.158. – DOI 10.1287/ijoc.8.2.158

[Potvin u. Rousseau 1993] Potvin, Jean-Yves ; Rousseau, Jean-Marc: A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. In: *European Journal of Operational Research* 66 (1993), Nr. 3, S. 331–340

[Potvin u. Rousseau 1995] Potvin, Jean-Yves ; Rousseau, Jean-Marc: An Exchange Heuristic for Routeing Problems with Time Windows. In: *Journal of the Operational Research Society* 46 (1995), Nr. 12, S. 1433–1446

[Psaraftis 1983] Psaraftis, Harilaos N.: k-Interchange procedures for local search in a precedence-constrained routing problem. In: *European Journal of Operational Research* 13 (1983), Nr. 4, S. 391–402. http://dx.doi.org/10.1016/0377-2217(83)90099-1. – DOI 10.1016/0377–2217(83)90099–1

[Quadt u. Kuhn 2008] Quadt, Daniel ; Kuhn, Heinrich: Capacitated lot-sizing with extensions: a review. In: *4OR: A Quarterly Journal of Operations Research* 6 (2008), Nr. 1, S. 61–83. http://dx.doi.org/10.1007/s10288-007-0057-1. – DOI 10.1007/s10288–007–0057–1

[R Development Core Team 2008] R Development Core Team: *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2008. http://www.R-project.org

[Reilly 2009] Reilly, Charles H.: Synthetic Optimization Problem Generation: Show Us the Correlations! In: *INFORMS Journal On Computing* 21 (2009), Nr. 3, S. 458–467. http://dx.doi.org/10.1287/ijoc.1090.0330. – DOI 10.1287/ijoc.1090.0330

[Reith-Ahlemeier 2002] Reith-Ahlemeier, Gabriele: *Ressourcenorientierte Bestellmengenplanung und Lieferantenauswahl*. Norderstedt : Books on Demand, 2002

[Romeijn u. Morales 2001] Romeijn, H.E. ; Morales, D.R.: Generating experimental data for the Generalized Assignment Problem. In: *Operations Research* 49 (2001), Nr. 6, S. 866–878

[Savelsbergh 1990] Savelsbergh, M.W.P.: An efficient implementation of local search algorithms for constrained routing problems. In: *European Journal of Operational Research* 47 (1990), Nr. 1, S. 75–85

[Schmitt 1984] Schmitt, Thomas G.: Resolving uncertainty in manufacturing systems. In: *Journal of Operations Management* 4 (1984), Nr. 4, S. 331–345. http://dx.doi.org/10.1016/0272-6963(84)90020-2. – DOI 10.1016/0272–6963(84)90020–2

[Schneeweiß 2002] Schneeweiß, Christoph: *Einführung in die Produktionswirtschaft*. Berlin, Heidelberg : Springer, 2002

[Scholl 2001] Scholl, Armin: *Robuste Planung und Optimierung. Grundlagen - Konzepte und Methoden - Experimentelle Untersuchungen*. 1. Heidelberg : Physica-Verlag, 2001

[Schrage 1979] Schrage, Linus: A More Portable Fortran Random Number Generator. In: *ACM Transactions on Mathematical Software* 5 (1979), Nr. 2, S. 132–138. http://dx.doi.org/10.1145/355826.355828. – DOI 10.1145/355826.355828. – ISSN 0098–3500

[Schwerin u. Wäscher 1997] Schwerin, P. ; Wäscher, G.: The Bin-Packing Problem: A Problem Generator and Some Numerical Experiments with FFD Packing and MTP. In: *International Transactions in Operational Research* 4 (1997), Nr. 5-6, S. 377–389. http://dx.doi.org/10.1111/j.1475-3995.1997.tb00093.x. – DOI 10.1111/j.1475–3995.1997.tb00093.x

[Sherali u. Smith 2001] Sherali, Hanif D. ; Smith, J. C.: Improving Discrete Model Representations via Symmetry Considerations. In: *Management Science* 47 (2001), Nr. 10, S. 1396–1407

[Silver u. a. 1998] Silver, Edward A. ; Pyke, David F. ; Peterson, Rein: *Inventory Management and Production Planning and Scheduling*. 3. New York : John Wiley & Sons, 1998

[Solomon 1987] Solomon, Marius M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. In: *Operations Research* 35 (1987), Nr. 2, S. 254–265

[Solomon u. Desrosiers 1988] Solomon, Marius M. ; Desrosiers, Jacques: Time Window Constrained Routing and Scheduling Problems. In: *Transportation Science* 22 (1988), Nr. 1, S. 1–13

[Spitter 2005] Spitter, Judith M.: *Rolling Schedule Approaches for Supply Chain Operations Planning*, Beta, Research School for Operations Management and Logistics, TU Eindhoven, Diss., 2005

[Stadtler 2000] Stadtler, Hartmut: Improved Rolling Schedules for the Dynamic Single-Level Lot-Sizing Problem. In: *Management Science* 46 (2000), Nr. 2, S. 318–326

[Suerie 2006] Suerie, Christopher: Modeling of period overlapping setup times. In: *European Journal of Operational Research* 174 (2006), Nr. 2, S. 874–886. http://dx.doi.org/10.1016/j.ejor.2005.03.033. – DOI 10.1016/j.ejor.2005.03.033

[Suerie u. Stadtler 2003] Suerie, Christopher ; Stadtler, Hartmut: The Capacitated Lot-Sizing Problem with Linked Lot Sizes. In: *Management Science* 49 (2003), Nr. 8, S. 1039–1054. http://dx.doi.org/10.1287/mnsc.49.8.1039.16406. – DOI 10.1287/mnsc.49.8.1039.16406

[Sun 2005] Sun: *The Java Language Specification, 3rd Edition*. http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html. Version: 2005. – abgerufen am 27.10.2009

[Sun 2006] Sun: *Java Platform, Standard Edition 6 API Specification*. http://java.sun.com/j2se/1.5.0/docs/api. Version: 2006. – abgerufen am 27.10.2009

[Sung u. Maravelias 2008] Sung, Charles ; Maravelias, Christos T.: A mixed-integer programming formulation for the general capacitated lot-sizing problem. In: *Computers & Chemical Engineering* 32 (2008), Nr. 1-2, S. 244–259. http://dx.doi.org/10.1016/j.compchemeng.2007.05.001. – DOI 10.1016/j.compchemeng.2007.05.001

[Swain 2007] Swain, James J.: New Frontiers in Simulation - Biennial survey of discrete-event simulation software tools. In: *OR/MS Today* 34 (2007), Nr. 5

[Tadikamalla 1978] Tadikamalla, Pandu R.: Computer Generation of Gamma Random Variables – II. In: *Communications of the ACM* 21 (1978), Nr. 11, S. 925–928

[Tempelmeier 2002] Tempelmeier, Horst: A Simple Heuristic for Dynamic Order Sizing and Supplier Selection with Time-Varying Data. In: *Production and Operations Management* 11 (2002), Nr. 4, S. 499–515

[Tempelmeier 2006] Tempelmeier, Horst: *Bestandsmanagement in Supply Chains*. 1. Norderstedt : Books on Demand, 2006

[Tempelmeier u. Buschkühl 2008] Tempelmeier, Horst ; Buschkühl, Lisbeth: Dynamic multi-machine lotsizing and sequencing with simultaneous scheduling of a common setup resource. In: *International Journal of Production Economics* 113 (2008), Nr. 1, S. 401–412

[Tempelmeier u. Buschkühl 2009] Tempelmeier, Horst ; Buschkühl, Lisbeth: A heuristic for the dynamic multi-level capacitated lotsizing problem with linked lotsizes for general product structures. In: *OR Spectrum* 31 (2009), Nr. 2, S. 385–404. http://dx.doi.org/10.1007/s00291-008-0130-y. – DOI 10.1007/s00291–008–0130–y

[Tempelmeier u. Derstroff 1996] Tempelmeier, Horst ; Derstroff, Matthias: A Lagrangean-based Heuristic for Dynamic Multilevel Multiitem Constrained Lotsizing with Setup Times. In: *Management Science* 42 (1996), Nr. 5, S. 738–757

[Tempelmeier u. Helber 1994] Tempelmeier, Horst ; Helber, Stefan: A heuristic for dynamic multi-item multi-level capacitated lotsizing for general product structures. In: *European Journal of Operational Research* 75 (1994), Nr. 2, S. 296–311

[Thomas 2005] Thomas, Douglas J.: Measuring Item Fill-Rate Performance in a Finite Horizon. In: *Manufacturing & Service Operations Management* 7 (2005), Nr. 1, S. 74–80. http://dx.doi.org/10.1287/msom.1040.0064. – DOI 10.1287/msom.1040.0064

[Trick 1992] Trick, Michael A.: A linear relaxation heuristic for the generalized assignment problem. In: *Naval Research Logistics* 39 (1992), Nr. 2, S. 137–151. http://dx.doi.org/10.1002/1520-6750(199203)39:2<137::AID-NAV3220390202>3.0.CO;2-D. – DOI 10.1002/1520–6750(199203)39:2<137::AID–NAV3220390202>3.0.CO;2–D

[Trigeiro u. a. 1989] Trigeiro, William W. ; Thomas, L. J. ; McClain, John O.: Capacitated Lot Sizing with Setup Times. In: *Management Science* 35 (1989), Nr. 3, S. 353–366

[Vanhoucke u. a. 2008] Vanhoucke, Mario ; Coelho, José ; Debels, Dieter ; Maenhout, Broos ; Tavares, Luís V.: An evaluation of the adequacy of project network generators with systematically sampled networks. In: *European Journal of Operational Research* 187 (2008), Nr. 2, S. 511–524. http://dx.doi.org/10.1016/j.ejor.2007.03.032. – DOI 10.1016/j.ejor.2007.03.032

[Vanhoucke u. Maenhout 2009] Vanhoucke, Mario ; Maenhout, Broos: On the characterization and generation of nurse scheduling problem instances. In: *European Journal of Operational Research* 196 (2009), Nr. 2, S. 457–467. http://dx.doi.org/10.1016/j.ejor.2008.03.044. – DOI 10.1016/j.ejor.2008.03.044. – ISSN 0377–2217

[Varga 2001] Varga, András: The OMNeT++ Discrete Event Simulation System. In: *Proceedings of the European Simulation Multiconference (ESM'2001)* (2001), June

[Vollmann u. a. 2005] Vollmann, Thomas E. ; Berry, William L. ; Whybark, D. C. ; Jacobs, F. R.: *Manufacturing Planning and Control Systems for Supply Chain Management*. 5. New York : McGraw-Hill, 2005

[Wacker 1985] Wacker, John G.: A theory of material requirements planning (MRP): an empirical methodology to reduce uncertainty in MRP systems. In: *International Journal of Production Research* 23 (1985), Nr. 4, S. 807–824

[Wallace 1974] Wallace, N.D.: Computer Generation of Gamma Random Variates with Non-integral Shape Parameters. In: *Communications of the ACM* 17 (1974), Nr. 12, S. 691–695

[Wemmerlöv u. Whybark 1984] Wemmerlöv, Urban ; Whybark, D. C.: Lot-sizing under uncertainty in a rolling schedule environment. In: *International Journal of Production Research* 22 (1984), Nr. 3, S. 467–484

[Whybark u. Williams 1976] Whybark, D. C. ; Williams, J. G.: Material Requirements Planning under Uncertainty. In: *Decision Sciences* 7 (1976), Nr. 4, S. 595–606. http://dx.doi.org/10.1111/j.1540-5915.1976.tb00704.x. – DOI 10.1111/j.1540–5915.1976.tb00704.x

[Wäscher u. Gau 1996] Wäscher, Gerhard ; Gau, Thomas: Heuristics for the integer one-dimensional cutting stock problem: A computational study. In: *OR Spectrum* 18 (1996), Nr. 3, S. 131–144. http://dx.doi.org/10.1007/BF01539705. – DOI 10.1007/BF01539705

[Zangwill 1966] Zangwill, Willard I.: A Deterministic Multi-Period Production Scheduling Model with Backlogging. In: *Management Science* 13 (1966), Nr. 1, S. 105–119

[Zangwill 1969] Zangwill, Willard I.: A Backlogging Model and a Multi-Echelon Model of a Dynamic Economic Lot Size Production System–A Network Approach. In: *Management Science* 15 (1969), Nr. 9, S. 506–527. http://dx.doi.org/10.1287/mnsc.15.9.506. – DOI 10.1287/mnsc.15.9.506

[Zhu u. Wilhelm 2006] Zhu, Xiaoyan ; Wilhelm, Wilbert E.: Scheduling and lot sizing with sequence-dependent setup: A literature review. In: *IIE Transactions* 38 (2006), Nr. 11, S. 987–1007