Diplomarbeit

Quantitative Analyse von Geschäftsprozessen

Diplomarbeit für die Prüfung zum

Diplom-Informatiker (FH)

an der Privaten FernFachhochschule Darmstadt

Fachbereich Informatik

von

Name:	Christian Czech
Matrikel-Nummer:	872462
Anschrift:	Schloßstrasse 36
	45355 Essen
betreut von:	Prof. Dr. Thomas Freytag
Abgabetermin:	01. August 2007

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abs.	Abschnitt
Anh.	Anhang
ARIS	Architektur integrierter Informationssysteme
BPR	Business Process Reengineering
bzw.	beziehungsweise
ca.	circa
d.h.	das heißt
dto.	dito, ebenso
EPK	Ereignisgesteuerte Prozesskette
ggf.	gegebenenfalls
GP	Geschäftsprozess
GPM	Geschäftsprozessmodellierung
i.d.R.	in der Regel
IT	Informationstechnologie
Kap.	Kapitel
o.B.d.A.	ohne Beschränkung der Allgemeinheit
OMG	Object Management Group (`http://www.omg.org/`)
o.g.	oben genannt
resp.	respektive
u.a.	unter anderem
UML	Unified Modeling Language
v.a.	vor allem
vgl.	vergleiche
WfMC	Workflow Management Coalition (`http://www.wfmc.org/`)
z.B.	zum Beispiel

Tabellenverzeichnis

Abbildungsverzeichnis

Verzeichnis der verwendeten Definitionen

Symbolverzeichnis

$\wedge$	Konjunktion (logisches UND)
$\vee$	Disjunktion (logisches ODER)
$\Rightarrow$	Implikation (wenn ..., dann ...)
$\Leftrightarrow$	Äquivalenz (... genau dann, wenn ...)
$\forall x :$	Allquantor (für alle x gilt ...)
$\exists x :$	Existenzquantor (es existiert ein x, für das gilt ...)
$\dot{\exists} x :$	Existenzquantor (es existiert genau ein x, für das gilt ...)
$a := b$	a ist definiert durch b
$\varnothing$	leere Menge ({ })
$\mathbb{P}(M)$	Potenzmenge der Menge M
$\lvert M \rvert$	Kardinalität der Menge M
$\in$	ist Element von
$\notin$	ist kein Element von
$\cap$	Mengenschnitt
$\cup$	Mengenvereinigung
$\setminus$	Mengendifferenz
$\subseteq$	Teilmenge
$\times$	Kreuzprodukt
$\xrightarrow{t}$	Zustandsübergang durch Schalten der Transition t
$\xrightarrow{\tau}$	Zustandsübergang durch Schaltfolge τ
$\xrightarrow{*}$	Zustandsübergang durch eine beliebige Schaltfolge
$t_1 t_2 ... t_n$	Folge der Transitionen t_1, t_2, ..., t_n
$\langle j_1, j_2, ..., j_n \rangle$	Pfad von Knoten in einem Graphen
$\pi_i(x_1, x_2, ..., x_n)$	Projektorfunktion; liefert das i-te Element des Tupels
$(X_t)_{t \in \mathbb{N}_0}$	Folge von Zufallsvariablen X mit Zeitindex

$P(X_t = i_t)$	Wahrscheinlichkeit für $X = i_t$ zum Zeitpunkt t
$P(A \mid B)$	Bedingte Wahrscheinlichkeit
$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathbb{N}_0$	Menge der natürlichen Zahlen einschließlich der Null
$\mathbb{R}^+$	Menge der positiven rellen Zahlen
$\mathbb{R}_0^+$	Menge der nichtnegativen Zahlen
∞	unendlich
$(0..1]$	linksoffenes Intervall von 0 (exklusiv) bis 1 (inklusiv)
$[0..1]$	geschlossenes Intervall von 0 bis 1 (jeweils inklusiv)
(a, b)	offenes Intervall von a bis b (jeweils exklusiv)
$\mathbf{N}, \mathbf{R_M}, \mathbf{R_Z}$	Strukturen
S, T, F	Mengen
μ, σ, τ	Funktionen oder Schaltfolgen (dann ohne Argument)
s, t, n	Stellen und Transitionen oder Zahlen oder Elemente
const.	konstant
$\aleph[\mathcal{A}] = O(K)$	Komplexität des Algorithmus $\mathcal{A}$ entspricht der Komplexitätsklasse K

1 Einleitung

Die Gestaltung und Optimierung von Geschäftsprozessen ist ein aktuelles Thema. Besonders der komplexe Bereich des Workflow-Managements tritt immer weiter in den Fokus der strategischen Betrachtung des Unternehmens im Spannungsfeld zwischen Marktanforderungen und Konkurrenzdruck.

Ein Gesichtspunkt dabei ist die Modellierung der Geschäftsprozesse. Damit wird aber nur der nächste Schritt eingeleitet, nämlich das neue Modell auf semantische und formale Korrektheit hin zu überprüfen und vor allem Effizienzmessungen durchzuführen.

Gegenstand dieser Diplomarbeit soll die quantitative Analyse der Geschäftsprozesse sein. Sie beschäftigt sich mit den Fragen der Effizienz der Prozesse und ihrer Simulation. Der letzte Punkt erlangt besondere Bedeutung durch die Risikosenkung bei der Implementierung neuer Prozesse in eine bestehende Organisation.

Mit einem Top-Down-Ansatz wird zunächst die Modellierung von Geschäftsprozessen besprochen und hierbei die besondere Eignung von Petri-Netzen als Modellierungssprache zur Analyse herausgestellt. Anschließend wird auf die Analyse allgemein und dann speziell auf die quantitative Analyse eingegangen.

Schwerpunkt der Arbeit wird die Darstellung der Performance-Analyse zur Effizienzmessung und der Simulation als Mittel zur Erkenntnisgewinnung sein.

Die betrachteten Algorithmen sollen für das Modellierungstool WoPeD (**W**orkflow **P**etri **N**et **D**esigner)[1] in Java implementiert werden. Dabei handelt es sich um Open-Source-Software, die an der Berufsakademie Karlsruhe entwickelt wurde und zur Modellierung und Analyse von Geschäftsprozessen auf Basis von Workflow-Netzen insbesondere im akademischen Bereich genutzt wird.

Abschließend werden die Ergebnisse der Arbeit kritisch beurteilt und ein Ausblick auf weitere Entwicklungsmöglichkeiten gegeben.

[1] Vgl. `http://www.woped.org`.

2 Geschäftsprozessmodelle und ihre Analyse

In Zeiten höheren Wettbewerbsdrucks und kürzerer Produktlebenszyklen spielt die Frage der Effizienz der Produktion bzw. des Dienstleistungsprozesses eine immer wichtigere Rolle. Ein erster Schritt zur Verbesserung der Performance von Unternehmen ist die Herausstellung der für den Markterfolg relevanten Geschäftsprozesse und ihre Analyse. Daher soll zunächst eine Definition des Geschäftsprozesses angegeben werden ([HaSt04, S. 23]):

> "Ein **Geschäftsprozess** ist eine Abfolge von Aktivitäten, die der Erzeugung eines Produktes oder einer Dienstleistung dienen. Er wird durch ein oder mehrere Ereignisse gestartet und durch ein oder mehrere Ereignisse abgeschlossen. Es liegt eine Organisationsstruktur zu Grunde."

Ineffizienzen, die während der Analyse erkannt wurden, müssen im nächsten Schritt beseitigt werden. Oftmals ist dazu der Geschäftsprozess umzugestalten oder komplett neu zu entwerfen. Man spricht von Geschäftsprozess-Reengineering (im Englischen: Business Process Reengineering, BPR).

Dabei steht das Management vor der Alternative, neue Technologien, insbesondere aus dem EDV-Bereich, einzusetzen. Ihr Zweck besteht darin, die Produktivität zu steigern und immer wiederkehrende gleichförmige Teile des Geschäftsprozesses zu automatisieren. Man spricht dann von Workflows ([HaSt04, S. 28]):

> "Ein **Workflow-Prozess** ist ein zusammenhängender rechnergestützter Teil eines Geschäftsprozesses."

Die Verwendung von Workflows hat aber i.d.R. einen Einfluss auf die Gestaltung des Geschäftsprozesses und unmittelbar auf die damit zusammenhängenden Ressourcen und somit auch auf die Organisation des Unternehmens. Es handelt sich daher um einen vielschichtigen komplexen Vorgang, der gemanagt werden muss. In diesem Zusammenhang spricht man vom Workflow-Management.

Die Umgestaltung der Unternehmensorganisation ist aber ein langwieriger, riskanter und daher auch sehr teurer Prozess. Ein Manager möchte gern vorher sehen, wie das Ergebnis der Veränderung aussehen wird und insbesondere, welche Effizienzsteigerung resp. Kostensenkung damit erreicht werden kann.

Es geht also um zwei Aspekte, die das Workflow-Management hierbei zu leisten hat: Zum einen sollte eine Abbildung von der Idee des künftigen Geschäftsprozesses existieren, ein Modell. Zum anderen soll das Modell außer der Visualisierung auch die Möglichkeit bieten, mittels Verwendung von Kostenfunktionen konkrete Aussagen über die Effizienz des Prozesses zu machen.

Es wird daher in diesem Kapitel zunächst auf den Modellierungsaspekt eingegangen, da ein geeignetes Modell des Geschäftsprozesses die Voraussetzung für die Performance-Analyse darstellt. Anschließend wird die Analyse selbst näher betrachtet.

2.1 Modelle von Geschäftsprozessen

Prozesse beinhalten im Wesentlichen zwei Struktur-Einheiten: die Aktivitäten und den Kontrollfluss, also die zeitliche Abfolge der Aktivitäten.

Aktivitäten wiederum verbinden zwei Entitäten, die zusammen mit dem Prozess eine vollständige Geschäftsprozess-Spezifikation ergeben: Fälle (oder auch Kunden) und Ressourcen.

Ein Prozess beschreibt dabei die konkrete zeitliche Abfolge einer Teilmenge von Aktivitäten, die zur Bearbeitung eines konkreten Falles unter Nutzung konkreter Ressourcen-Objekte benötigt werden. Im Geschäftsprozess vereinigen sich alle Prozesse, die eine ähnliche Aufgabe erfüllen. Das Modell des Geschäftsprozesses ist ein Prozessmodell ([HaSt04, S. 30]):

> "Ein **Prozessmodell** beschreibt die Struktur eines realen Prozesses. Es bestimmt alle möglichen Pfade entlang des Prozesses und bestimmt die Regeln für die Wahl der Pfade. Weiterhin bestimmt das Prozessmodell alle Aktivitäten, die ausgeführt werden müssen."

Auf die Modellierung im Allgemeinen und im speziellen Fall von Geschäftsprozessen wird in Folgenden eingegangen.

2.1.1 Modellierung

Modellierung bezeichnet den Prozess der Erstellung eines Modells. Eine sehr allgemeine Definition lautet:

> "Ein **Modell** ist ein vereinfachendes Abbild der Wirklichkeit."

Ausgangspunkt dafür ist ein Anschauungsobjekt. Im Falle der Geschäftsprozessmodellierung handelt es sich dabei um einen bereits existierenden oder einen zukünftig existierenden (angestrebten) Geschäftsprozess.

Nach der allgemeinen Modelltheorie von Stachowiak ([Stac73]) besitzt jedes Modell drei Merkmale (vgl. [Broc03, S. 9]):

1. Abbildungsmerkmal:
   Es existiert eine Abbildung, die das Anschauungsobjekt (den Realitätsausschnitt) auf das Modell projiziert. Diese Abbildungsrelation soll dabei weitestgehend strukturerhaltend sein und im Verhalten dem Original ähnlich. Somit erhält das Modell eine *syntaktische und eine semantische Komponente*. Die Abgrenzung des Realitätsausschnittes erfolgt subjektiv je nach Zielsetzung durch die Entscheidung, welche der konstituierenden Objekte in das Modell übernommen werden.

2. Verkürzungsmerkmal:
   Das Modell besitzt i.d.R. nicht dieselbe Komplexität wie das Original, sondern stellt eine Vereinfachung dar, die durch *Abstraktion* erreicht wird.

3. Pragmatisches Merkmal:
   Die Auswahl der Eigenschaften des realen Anschauungsobjektes, die in das Modell aufgenommen werden sollen, erfolgt subjektiv durch den Modellierer je nach seiner *Zielsetzung*, die zum Zeitpunkt der Modellierung besteht. Das bedeutet insbesondere, dass eine solche Auswahl zu einem anderen Zeitpunkt anders ausfallen kann, z.B. wenn neues Wissen über das Anschauungsobjekt besteht.

Auf dem so entwickelten Modell können dann Operationen durchgeführt werden, die am realen Objekt entweder nicht möglich oder zu gefährlich, zu langwierig oder zu kostenaufwändig wären oder Seiteneffekte hervorrufen würden, die nicht mehr rückgängig zu machen sind. Eine Sensitivitätsanalyse zur schrittweisen Suche nach einer optimalen Lösung durch ”try and error” zum Beispiel ist am Anschauungsobjekt selten möglich.

Die Ergebnisse der Modell-Manipulation werden anschließend im Kontext des Realitätsausschnittes interpretiert und die so gewonnenen Erkenntnisse auf das reale Objekt übertragen.

Der Zweck der Modellierung besteht also darin, Operationen nicht am Original, sondern am Modell durchzuführen, weil das entsprechende Vorteile bringt (geringe Kosten, geringerer Zeit- und Materialaufwand, geringere Gefahren).

Das Metamodell zur Modellierung wird nochmals grafisch dargestellt (vgl. [KaKB05, S. 20]):

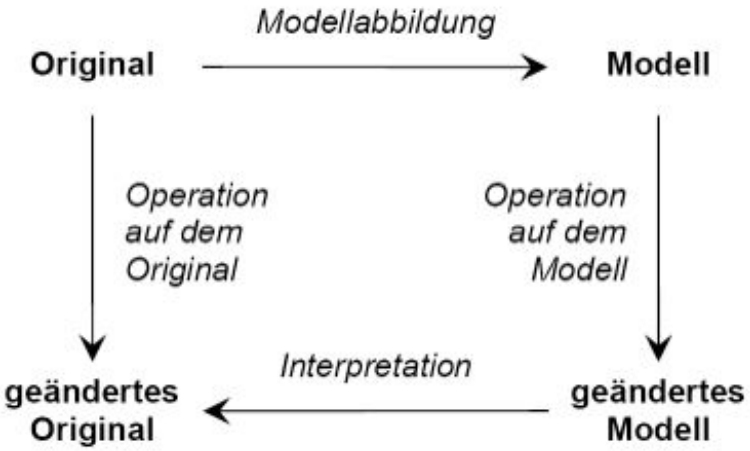


Abbildung 1: Metamodell der Modellierung.

In der nachfolgenden Abbildung werden systematisch verschiedene Möglichkeiten der Analyse eines Systems (bzw. Prozesses) aufgeführt (vgl. [Law07, S. 4], Übersetzung durch Autor):

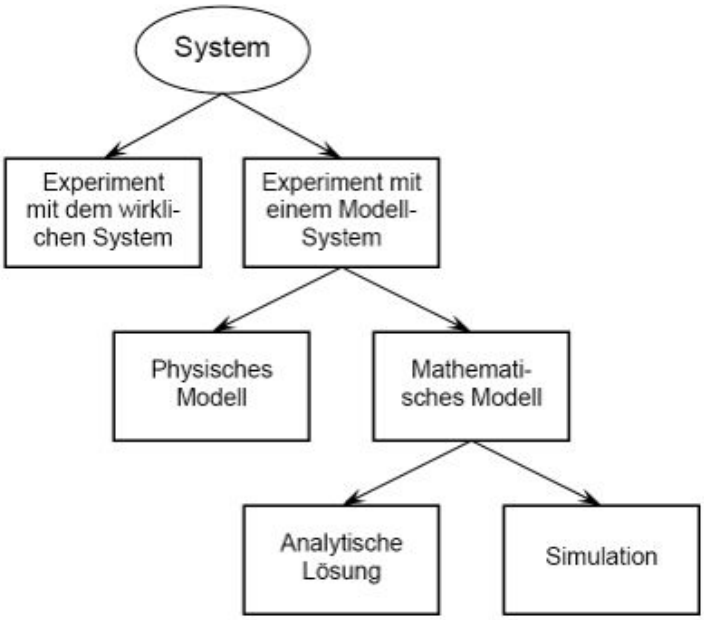


Abbildung 2: Analysevarianten eines Systems.

In Bezug auf die Modellierung von Geschäftsprozessen werden keine physischen Modelle konstruiert, da sich die Analyse auf Effizienzmaße bezieht, die keine physische Repräsentation besitzen.

Analytische Modelle sind teilweise möglich, scheitern aber oft an der Komplexität der Prozessstrukturen, wie in Unterabschnitt 2.2.3 noch ausgeführt werden wird.

Eine kostengünstige Variante ist die Simulation. Sie bietet sich immer dann an, wenn der Berechnungsaufwand enorm wird oder eine geschlossene analytische

Betrachtung nicht möglich ist. Sie bietet v.a. die Möglichkeit, sehr einfach die Auswirkung von Veränderungen der Eingabe-Parameter zu untersuchen.

Im Folgenden wird die Betrachtung auf die Geschäftsprozess-Modellierung beschränkt.

2.1.2 Modellelemente von Geschäftsprozessen

Geschäftsprozesse beschreiben die dynamischen Aspekte eines Unternehmens (die Ablauforganisation). Dynamik wird durch reaktive Elemente erreicht, die auf das Eintreffen bestimmter **Ereignisse** warten und dann vorgegebene Tätigkeiten ausführen. Man spricht auch von kybernetischen Systemen.

Ereignisse wiederum werden anhand der Veränderung der Werte von Zustandsvariablen des Systems erkannt. Die **Aktivitäten**, die sie auslösen, verändern diese aber auch und lösen somit ihrerseits wieder Ereignisse aus. Die Menge der aktuellen Werte aller Zustandsvariablen beschreiben den **Systemzustand** zu diesem Zeitpunkt. Damit ergibt sich als notwendige Eigenschaft für ein dynamisches System die strikte Wechselfolge von Ereignis und Aktivität.

Aktivitäten (activities) bezeichnen die Ausführung von **Aufgaben** (tasks) für einen bestimmten **Fall** (case), wobei **Ressourcen** in Anspruch genommen werden. Die Aufgaben sind dabei unabhängig von einem konkreten Fall definiert. Nach van der Aalst (vgl. [AaHe02, S. 33]) ergibt sich folgende Beziehung zwischen diesen Begriffen:

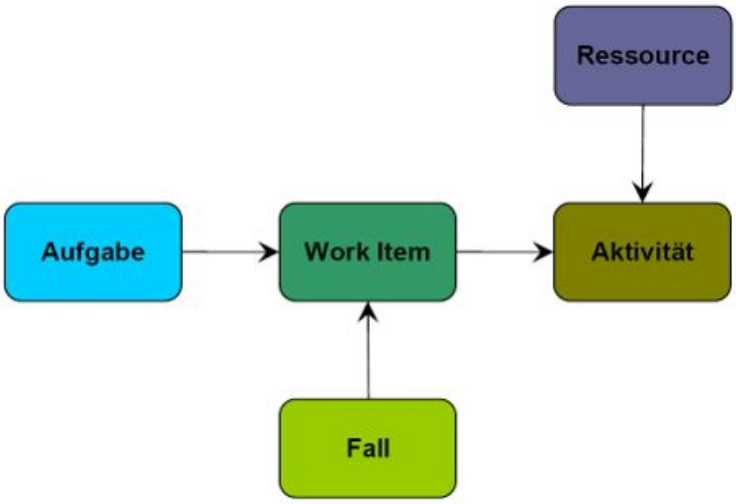


Abbildung 3: Beziehung zwischen Aufgabe, Fall und Aktivität.

Jeder Aktivität können somit drei Dimensionen zugeordnet werden ([HaSt04, S. 39]):

> "Eine **Aktivität** ist ein Tripel aus einer Aufgabe, einem Fall und einer Ressource. Ein Work Item, das von einer bestimmten Ressource (oder mehreren) bearbeitet wird, wird also als Aktivität bezeichnet. Aktivität = (Aufgabe, Fall, Ressource)."

Für den Fall, das ein flexibler Einsatz von Ressourcen möglich sein soll, muss ein adäquates **Ressourcenmodell** angegeben werden. Gemäß van der Aalst ([AaHe02, Abs. 3.1 und 3.2]) kann dies durch eine Ressourcenzuweisung mittels Klassifikation geschehen.

Hierbei werden zwei Dimensionen der Ressourcenklassifikation unterschieden: die **Rolle** und die **Gruppe** (auch Organisationseinheit). Die Rolle einer Ressource beschreibt ihre funktionale Eignung, bestimmte Aufgaben ausführen zu können. Die Gruppe bezeichnet ihre Zuordnung innerhalb der Unternehmenshierarchie.

Die funktionalen bzw. Organisationseinheiten können mehrere Ressourcenobjekte umfassen und werden daher als Ressourcenklassen bezeichnet. Auf diese Weise ist es möglich, auch komplexe Ressourcenzuweisungen abzubilden.

[JoFr96] sprechen im Zusammenhang mit der quantitativen Modellierung von zwei Dimensionen: (1) dem dynamischen oder Verhaltensbereich und (2) dem Objektbereich. Der Verhaltensbereich beschreibt die Aktivitäten und deren Beziehungen, also praktisch das Prozessmodell. Der Objektbereich bezieht sich auf die Ressourcen, die die Aktivitäten ausführen. Der Zusammenhang zwischen beiden Bereichen wird durch Zuweisung von Aktivitäten zu Ressourcen (resp. Ressourcenklassen) hergestellt.

2.1.3 Sprachen zur Modellierung

Um Modelle von Geschäftsprozessen zu erstellen, werden Modellierungssprachen benötigt. Diese müssen v.a. folgende Eigenschaften haben:

- Universalität oder Allgemeingültigkeit:
  Mit der verwendeten Modellierungssprache soll es möglich sein, eine Vielzahl verschiedenster Geschäftsprozesse zu modellieren, ohne künstliche d.h. durch die Sprache selbst hervorgerufene Restriktionen hinnehmen zu müssen. Alle Aspekte der Modellierung von Geschäftsprozessen, wie sie eben aufgeführt wurden, sollen abbildbar sein.

- Einfachheit:
  Die Modellierung von Geschäftsprozessen erfolgt i.d.R. nicht durch IT-Fachpersonal, sondern durch die Anwender. Abstrakte Sprachen, die ein

Spezialwissen verlangen, eignen sich daher nicht. Meistens werden graphische Modelle verwendet, deren Metasprache weitestgehend intuitiv verständlich ist. Die verwendete Modellierungssprache muss also leicht zu erlernen sein.

- Semantische Eindeutigkeit:
  Trotz der Forderung nach Einfachheit der Ausdrucksweise ist eine präzise Beschreibung dennoch unumgänglich. Die Interpretation des Modells muss eindeutig sein. Dazu muss die Sprache gewisse formale Ansprüche erfüllen, auf deren Einhaltung bei der Modellierung streng zu achten ist.

Außerdem muss die Sprache in der Lage sein, die Ablauflogik des Prozesses korrekt wiederzugeben. Dazu werden Strukturen verwendet, die den Kontrollfluss steuern, wie sie prinzipiell auch von Programmiersprachen bekannt sind (vgl. [GuSo06, S. 131-136]): die Hintereinanderausführung (Sequenz), die Auswahl (Alternative) und die Wiederholung (Iteration). Bei Prozessen kommt außerdem noch die Parallelität von Abläufen (Nebenläufigkeit) hinzu.

Ferner ist es im Falle von Workflows wünschenswert, direkt vom Modell ausgehend den Prozess ausführen zu können. Dazu wird die Workflow Engine des Workflow Enactment Services mit der Prozessdefinition und der Ressourcenzuweisung geladen. Hierzu muss die verwendete Sprache das Interface der Workflow API implementieren. Das so skizzierte Referenzmodell stammt von der WfMC und ist hier abgebildet[2]:

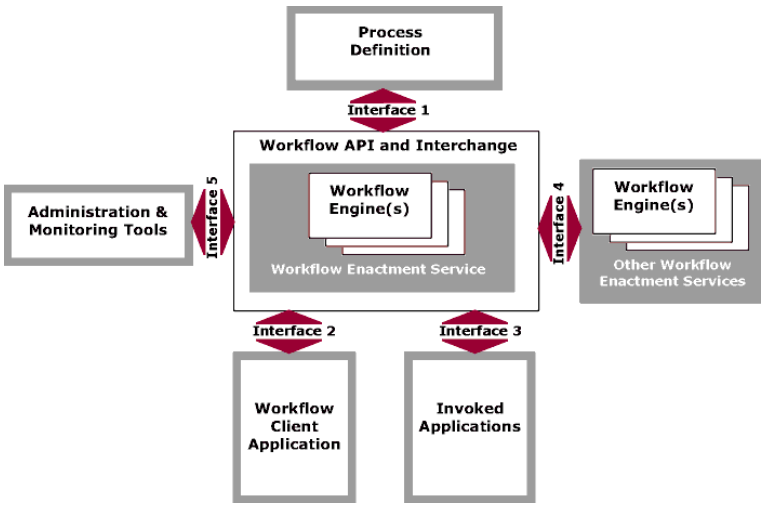


Abbildung 4: Referenzmodell der WfMC.

Im Folgenden werden das klassische Referenzmodell zur Geschäftsprozessmodellierung ARIS von Prof. Scheer sowie die universell verwendbare Modellierungssprache UML der OMG und das Konzept der Petri-Netze von Prof. Petri

[2] `http://www.wfmc.org/graphics/processmodel_large.gif`, 05.05.2007.

auf ihre Tauglichkeit hin bezüglich der drei genannten Kriterien und im Hinblick auf die quantitative Analyse bewertet.

ARIS / UML

Das inzwischen wohl bekannteste Werkzeug zur Geschäftsprozessmodellierung ist ARIS. Damit beschreibt Scheer vier Sichten auf einen Geschäftsprozess, die jeweils eine andere Dimension im Fokus haben (vgl. [Sche97, Teil A]). Eine grobe Definition lautet ([Sche96, S. 10]):

> "Mit **ARIS** (**Ar**chitektur integrierter **I**nformations**s**ysteme) ist ein Rahmenkonzept und eine Methodologie zur vollständigen Beschreibung von Geschäftsprozessen entwickelt worden [...]."

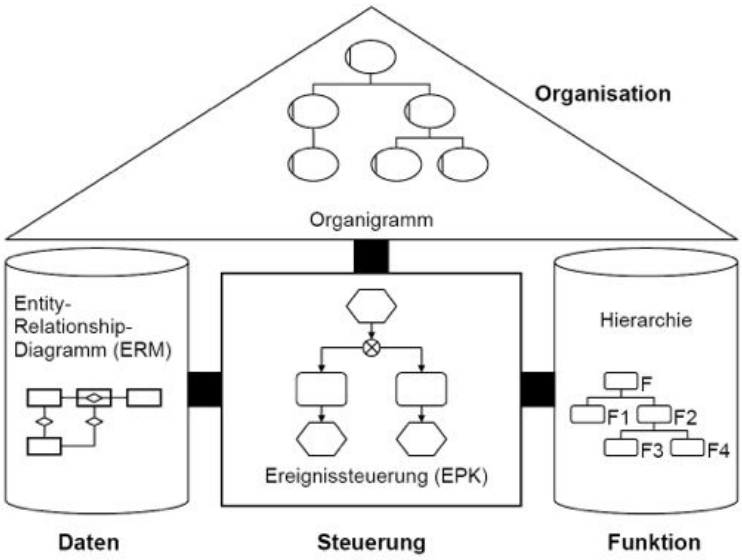


Abbildung 5: ARIS-Haus von Scheer (Auszug), eigene Darstellung.

Kernstück ist die erweiterte **ereignisgesteuerte Prozesskette** (eEPK) in der Steuerungssicht, die die Organisationssicht (Organigramm), die Funktionssicht (Funktionshierarchiebaum) und die Datensicht (Entity-Relationship-Modell) verbindet und die Ablauflogik beschreibt.

Dabei finden sich die Aufgaben in den Funktionen und die Ressourcen im Organigramm wieder. In der Datensicht sind Fälle, Ressourcen und Aufgaben gemeinsam vertreten. In Erweiterungen existiert eine eigene Ressourcensicht. Die EPK für das im Anhang A beschriebene Beispiel eines Geschäftsprozesses sieht dann so aus:

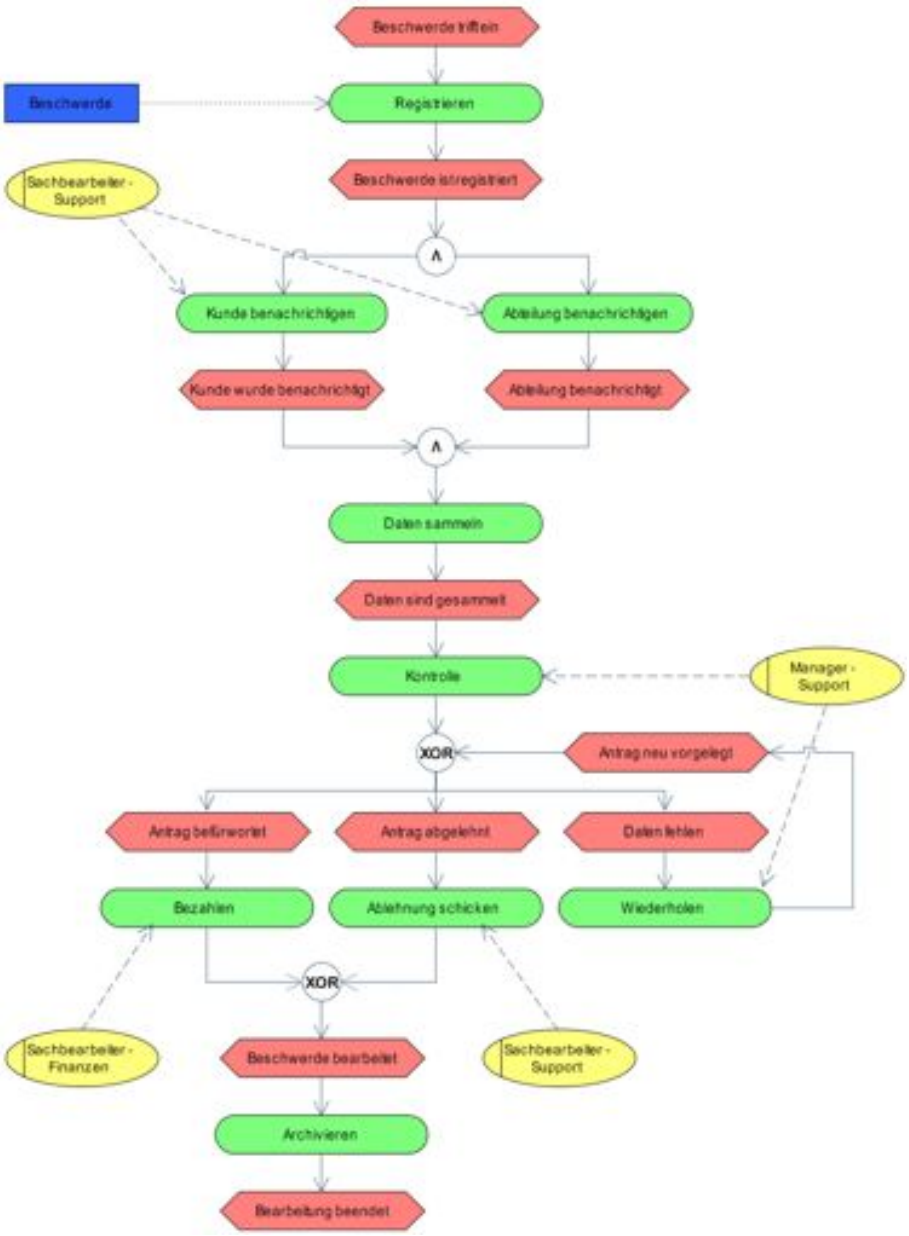


Abbildung 6: Ereignisgesteuerte Prozesskette zum Beispielprozess

Die Stärke von ARIS ist seine graphische Darstellung und intuitive Verständlichkeit. Es ist jedoch wenig formal und beinhaltet praktisch keine spezifischen Daten zur Analyse des Geschäftsprozesses. So finden sich z.B. keine Leistungsdaten der Ressourcenobjekte oder zeitliche Dimensionen.

Der Mangel an formaler Semantik in ereignisgesteuerten Prozessketten führt dazu, dass die Verifizierung eines Modells besonders schwierig ist oder sogar nicht entscheidbar. Besonders bei alternativen Prozesspfaden, denen kein gegenseitiger Ausschluss zugrunde liegt - sogenannte OR-Konstrukte[3] -, gibt es keine Möglichkeit, das Verhalten genau zu beschreiben[4].

[3] Damit ist das logische ODER (A oder B oder A und B) gemeint, im Gegensatz zum exklusiven XOR (entweder A oder B).

[4] Vgl. dazu die Diskussion zur Semantik von EPKs in [AaDe02] und der "Busfahrer-Semantik" in E. KINDLER. *On the semantics of EPCs: Resolving the vicious circle*. In: J. DE-

Mit dem ARIS-Toolset lassen sich Geschäftsprozesse definieren, darstellen und steuern. Eine direkte Umsetzung in Software ist damit möglich. Es besteht aber keine Unterstützung für Performanceanalysen.

Einen allgemeineren und zugleich formaleren Ansatz stellt die UML dar (vgl. [**?**], [GrBa04]). Sie ist im Wesentlichen eine Sammlung von Diagrammen, die unterschiedlichste Aspekte eines Modells veranschaulichen. Eine grobe Einteilung geschieht in statische und dynamische Sichten.

Die Definition der OMG lautet ([OMGI07, S. 9]):

> "The **Unified Modeling Language** is a visual language for specifying, constructing, and documenting the artifacts of systems. It is a general-purpose modeling language that can be used with all major object and component methods, and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, .NET)."

Die UML versteht unter einem Geschäftsprozess den dynamischen Anteil, die Aktionsfolge. Ressourcen, Fälle und sonstige Informationsobjekte werden in der statischen Sicht erfasst. Beide Sichten zusammen bilden das Geschäftssystem (vgl. [GrBa04, Kap. 3]).

Bei der Modellierung eines Geschäftssystems unterscheidet die UML zwischen einer **externen Sicht** - die die Schnittstellen des Geschäftsprozesses zu seiner Außenwelt spezifiziert und einer **internen Sicht** - die die Ablauforganisation des Geschäftsprozesses festlegt.

Hauptsächliches Mittel der Modellierung in UML ist das Aktivitätsdiagramm, das in der externen Sicht durch ein Sequenzdiagramm und in der internen Sicht durch Paket- und Klassendiagramme ergänzt wird. Zur Modellierung einzelner Szenarien können Anwendungsfalldiagramme hinzugefügt werden.

In [KoHu00] wird eine Erweiterung von UML-Zustandsdiagrammen mit einer zeitlichen Dimension vorgeschlagen, um quantitative Analysen zu ermöglichen. [RuAa06] zeigen, dass insbesondere die Modellierung einer Ressourcensicht mit Aktivitätsdiagrammen unbefriedigend ist. Quantitative Analysen sind daher auf ihrer Grundlage nicht durchführbar.

Der Schwerpunkt von UML-Modellen liegt auf der Spezifikation von Systemen. Sie wird bei der Analyse und im Entwurf verwendet. Eine anschließende quantitative Analyse der Modelle ist mit den angebotenen Mitteln nicht möglich und war auch nicht beabsichtigt.

SEL, B. PERNICI, M. WESKE (Eds.). Business Process Management. Second International Conference, BPM 2004. LNCS 3080, S. 82?97, Springer, 2004.

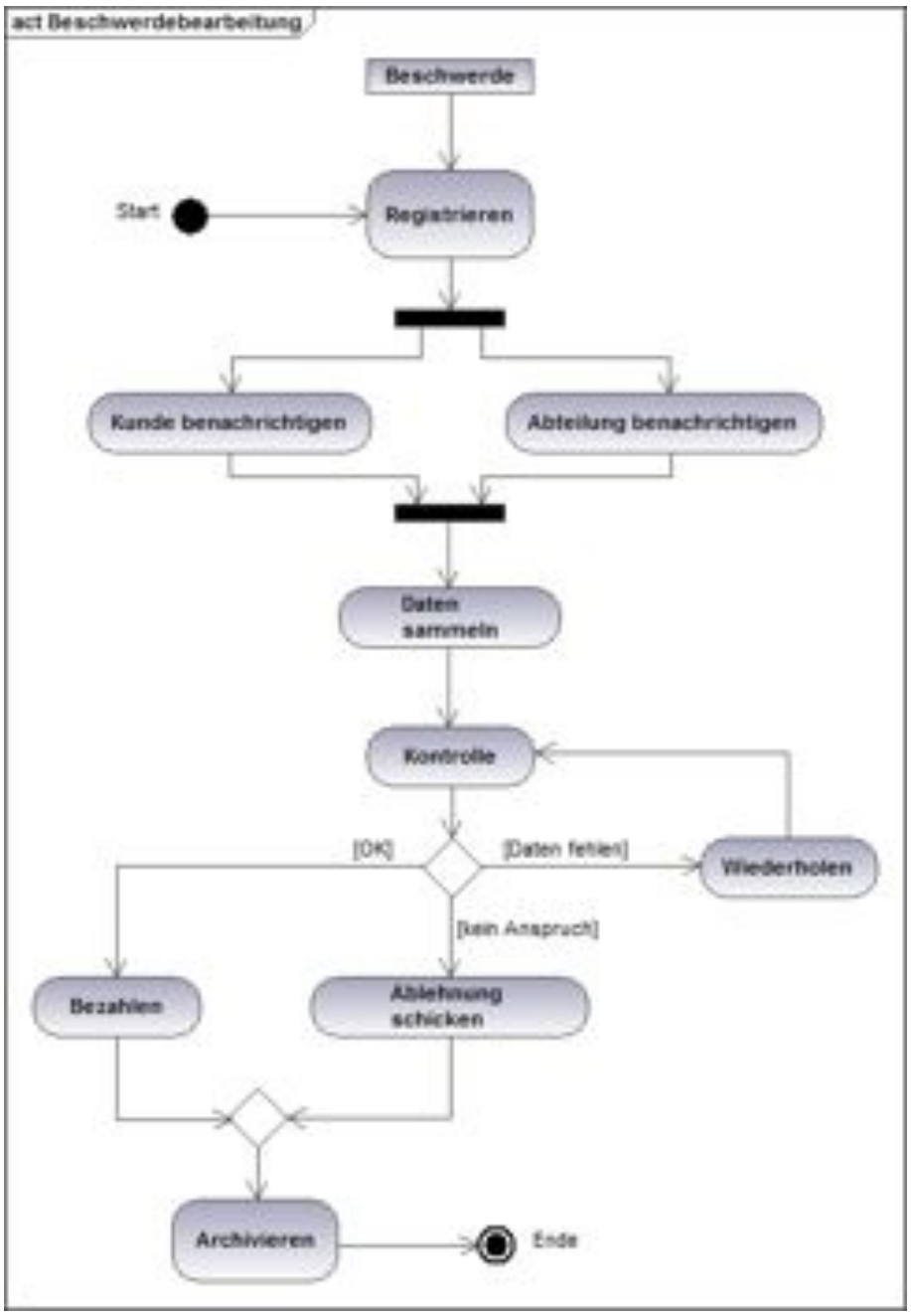


Abbildung 7: Aktivitätsdiagramm zum Beispiel aus Anhang A.

Eine weitere Modellierungssprache, die für die Geschäftsprozessmodellierung eingesetzt werden kann und dabei eine formale Begründung hat, die es mit einigen Ergänzungen des Grundmodells erlaubt, quantitative Analysen direkt auf dem Modell durchzuführen, sind die Petri-Netze.

Petri-Netze

Mit dem auf C. A. Petris Dissertation ”Kommunikation mit Automaten” von 1962 basierendem Modell zur formalen Beschreibung von dynamischen Systemen lassen sich mit einigen Erweiterungen auch Geschäftsprozesse modellieren.

Der formale Charakter erlaubt einen direkten Zugang zu quantitativen Analysemethoden, wie sich noch zeigen wird. Der Schwerpunkt dieser Diplomarbeit wird die Darstellung quantitativer Methoden unter Verwendung von Petri-Netzen sein.

Zunächst werden grundlegende Definitionen im Zusammenhang mit Petri-Netzen zusammengetragen[5]:

> Ein (endliches) **Petri-Netz** ist ein bipartiter Graph $\mathbf{N} = (S, T, F)$ mit (endlicher) Knotenmenge $S \cup T$, wobei S die Menge der **Stellen** und T die Menge der **Transitionen** bezeichnet, so dass gilt: $S \cap T = \varnothing$, $S \cup T \neq \varnothing$. Die Knoten werden über gerichtete Kanten verbunden. Die Menge der Kanten wird durch die binäre **Flussrelation** F beschrieben: $F \subseteq (S \times T) \cup (T \times S)$.

Stellen sind die passiven Bestandteile im Netz und können als Vor- bzw. Nachbedingungen betrachtet werden. Sie entsprechen den Ereignissenin der Definition des Geschäftsprozesses. Transitionen stellen die aktiven, dynamischen Strukturelemente dar. Sie stehen für die Ausführung von Tätigkeiten oder Zustandsübergänge.

Es werden in Abhängigkeit von einem betrachteten Knoten des Netzes folgende Knotenmengen definiert:

> Der **Vorbereich** $\bullet k$ des Knotens $k \in S \cup T$ ist die Menge aller Eingangsknoten von k: $\bullet k = \{e \in S \cup T \mid (e,k) \in F\}$. Der **Nachbereich** $k\bullet$ des Knotens $k \in S \cup T$ ist die Menge aller Ausgangsknoten von k: $k\bullet = \{a \in S \cup T \mid (k,a) \in F\}$.

Eine der herausragenden Stärken des Petri-Netz-Modells ist die Fähigkeit, dynamische Aspekte zu modellieren. Dies gelingt durch das Konzept der Markierung und dem Schalten von Transitionen:

> Sei $\mathbf{N} = (S, T, F)$ ein Petri-Netz. Eine Abbildung $\mu : S \to \mathbb{N}_0, s \mapsto \mu(s)$, die jeder Stelle $s \in S$ eine nicht-negative ganze Zahl zuordnet, heißt **Markierung** von $\mathbf{N}$. $\mu(s)$ wird als Anzahl von Marken oder Token verstanden. Eine Markierung von $\mathbf{N}$ beschreibt den Zustand des Netzes. Das Paar $(\mathbf{N}, \mu)$ bzw. das Tupel (S, T, F, μ) wird als **markiertes Netz** bezeichnet.

Der Anfangszustand eines Petri-Netzes wird durch die Anfangsmarkierung μ_0 beschrieben. Werden Stellen als Vor- oder Nachbedingungen verstanden, dann

[5] Vgl. [AaHe02, Anh. A], [Baum96, Kap. 3 und 4], [HaSt04, Abs. 3.3], [KaKB05, Abs. 7.2].

bedeutet eine Marke auf der Stelle s, dass die durch s symbolisierte Bedingung gilt.

Im Rahmen der Modellierung von Geschäftsprozessen stehen Marken für (anonyme) Fälle, die eine entsprechende Position im Prozess erreicht haben. Mehrere Marken in einer Stelle stehen für verschiedene Fälle.

Transitionen, die die Aktivitäten darstellen, konsumieren Marken von allen ihren Eingangsstellen und produzieren Marken auf allen ihren Ausgangsstellen.

Im Workflow-Kontext heißt das, dass Transitionen Fälle übernehmen, bearbeiten und weiterleiten. Es wird implizit angenommen, dass aus der Menge anstehender Fälle immer nur einer als nächstes entnommen wird und Fälle bei ihrer Bearbeitung nicht verschwinden oder sich vermehren können.

Im Folgenden werden daher nur noch Petri-Netze betrachtet, bei denen jede Transition von jeder Eingangsstelle genau eine Marke konsumiert und für jede Ausgangsstelle genau eine Marke produziert.

Das Schalten von Transitionen wird nun formalisiert:

Sei $\mathbf{N} = (S, T, F, \mu)$ ein markiertes Petri-Netz. Eine Transition $t \in T$ heißt **aktiviert**, wenn gilt: $\forall s \in \bullet t : \mu(s) > 0$. Aus der Menge aller unter μ aktivierten Transitionen kann eine beliebige Transition t schalten. Dadurch erhält $\mathbf{N}$ die Folgemarkierung μ', in Symbolen: $\mu \xrightarrow{t} \mu'$. Das Schalten erfolgt nach der **Schaltregel**:

$$\begin{aligned} \mu'(s) = \mu(s) \quad &\text{wenn} \quad (s \notin \bullet t \wedge s \notin t\bullet) \vee (s \in \bullet t \wedge s \in t\bullet) \\ \mu'(s) = \mu(s) - 1 \quad &\text{wenn} \quad s \in \bullet t \wedge s \notin t \bullet \\ \mu'(s) = \mu(s) + 1 \quad &\text{wenn} \quad s \notin \bullet t \wedge s \in t \bullet . \end{aligned}$$

Eine Folge von Transitionen $\tau = t_1 t_2 ... t_n$, die μ nacheinander über $\mu', \mu'', ...$ in $\mu^{(n)}$ überführt, heißt **Schaltfolge**, in Symbolen: $\mu \xrightarrow{\tau} \mu^{(n)}$. μ' ist von μ **erreichbar**, wenn gilt: $\exists \tau : \mu \xrightarrow{\tau} \mu'$. Symbolisch wird die Erreichbarkeit ausgedrückt durch: $\mu \xrightarrow{*} \mu'$. Mit $[\mu\rangle$ wird die Menge aller von μ erreichbaren Markierungen bezeichnet. Es gilt: $\mu \in [\mu\rangle$. Eine Transition $t \in T$ heißt (unter μ) **aktivierbar**, wenn gilt: $\mu \xrightarrow{*} \mu' \Rightarrow \mu'(s) > 0 \; \forall s \in \bullet t$.

Für die Geschäftsprozessmodellierung müssen an die Struktur der Petri-Netze weitere Anforderungen gestellt werden, da sonst semantische Fehler auftreten können. Workflow-Netze sind eine spezielle Klasse von Petri-Netzen zur Modellierung von Geschäftsprozessen:

Ein Petri-Netz $\mathrm{N} = (S, T, F)$ ist ein **Workflow-Netz** genau dann, wenn

(i) $\dot{\exists} i \in S : \bullet i = \varnothing$ (**Quelle**).

(ii) $\dot{\exists} o \in S : o\bullet = \varnothing$ (**Senke**).

(iii) $\forall k \in S \cup T : \exists \langle i, j_1, j_2, ..., j_m, k, l_1, l_2, ..., l_n, o \rangle$ mit $j_u, l_v \in S \cup T$

$\forall u \in \{1, 2, ..., m\}, v \in \{1, 2, ..., n\}$ und $m, n \in \mathbb{N}_0$.

Semantische Fehler treten auf in Form von (vgl. [AaHe02, S. 106]):

- Transitionen mit leerem Vor- oder Nachbereich,
- Toten Transitionen,
- Deadlocks (oder Verklemmungen),
- Livelocks (oder Endlosschleifen),
- Existenz aktivierter Transitionen, obwohl o erreicht wurde,
- Vorkommen weiterer Marken im Netz, obwohl der zugehörige Fall abgeschlossen wurde.

Eine Transition heißt **tot**, wenn sie unter keiner von μ_0 erreichbaren Markierung aktiviert ist. Sei $\mathrm{N} = (S, T, F, \mu_0)$ ein markiertes Petri-Netz. Eine Markierung $\mu \in [\mu_0\rangle$ heißt **Deadlock** oder Verklemmung, wenn gilt: $[\mu\rangle = \varnothing$. N **terminiert**, wenn es endlich viele Schaltfolgen gibt. Eine Markierung μ von N heißt **Livelock**, wenn N nach Erreichen von μ nicht terminiert und keine der von μ erreichbaren Markierungen ein Deadlock ist.

Workflow-Netze als Modelle von Geschäftsprozessen müssen zudem ein sicheres dynamisches Verhalten aufweisen. Darin spiegelt sich die Vermeidung o.g. Fehler wieder. Man beschreibt diese Eigenschaft mit ”sound und safe”.

Die Soundness-Eigenschaft ist formal begründet und lässt sich aus der Netzstruktur ableiten. Informell bedeutet sie, dass ein Workflow-Netz korrekt ist, in dem Sinne, dass jeder Fall, der den Prozess durchläuft, die Senke erreicht, womit seine Bearbeitung endet und dass dann keine weiteren Referenzen (Marken) auf diesen Fall im Netz und keine toten Transitionen existieren.

Die Soundness-Eigenschaft ist eine wichtige Voraussetzung der quantitativen Analyse. Für eine formale Definition sind zunächst einige weitere Begriffe notwendig:

Mit $\iota_{\mathbf{N}}$ wird die **Anfangsmarkierung** des Netzes $\mathbf{N} = (S, T, F)$ bezeichnet. Es gilt: $\iota_{\mathbf{N}}(i) = 1 \wedge \iota_{\mathbf{N}}(s) = 0 \; \forall s \in S \setminus \{i\}$.
Mit $o_{\mathbf{N}}$ wird die **Endemarkierung** des Netzes $\mathbf{N} = (S, T, F)$ bezeichnet. Es gilt: $o_{\mathbf{N}}(o) = 1 \wedge o_{\mathbf{N}}(s) = 0 \; \forall s \in S \setminus \{o\}$.
Wenn es nicht zu Mehrdeutigkeiten führt, kann jeweils der Index auch weggelassen werden.
Auf der Menge der Markierungen eines Netzes $\mathbf{N} = (S, T, F)$ wird eine **Ordnung** $\leq$ definiert. Für zwei beliebige Markierungen μ_1 und μ_2 gilt: $\mu_1 \leq \mu_2 :\Leftrightarrow \mu_1(s) \leq \mu_2(s) \; \forall s \in S$.
Ein markiertes Netz $\mathbf{N} = (S, T, F, \iota)$ heißt (stark) **lebendig**, wenn gilt: $\forall \mu \in [\iota\rangle : \forall t \in T : \exists \mu' : \mu \xrightarrow{t} \mu'$.
Ein markiertes Netz $\mathbf{N} = (S, T, F, \iota)$ heißt **beschränkt**, wenn gilt: $\exists n \in \mathbb{N} : \iota \xrightarrow{*} \mu \Rightarrow \mu(s) \leq n \; \forall s \in S$. Es heißt **sicher**, wenn $n = 1$.
Ein markiertes Netz $\mathbf{N} = (S, T, F, \iota)$ heißt **sound** genau dann, wenn gilt:

(i) $\iota \xrightarrow{*} \mu \Rightarrow \mu \xrightarrow{*} o$ und

(ii) $\iota \xrightarrow{*} \mu \wedge \mu \geq o \Rightarrow \mu = o$ und

(iii) $\forall t \in T : \exists \mu, \mu' : \iota \xrightarrow{*} \mu \xrightarrow{t} \mu'$.

Der Nachweis der Soundness eines Workflow-Netzes lässt sich indirekt mit Hilfe folgenden Satzes führen[6]:

Sei $\mathbf{N} = (S, T, F)$ ein Workflow-Netz. $\varphi\mathbf{N} = (\varphi S, \varphi T, \varphi F)$ bezeichne das kurzgeschlossene Netz von $\mathbf{N}$, definiert durch: $\varphi S = S \wedge \varphi T = T \cup \{t^*\} \wedge \varphi F = F \cup \{(o, t^*), (t^*, \iota)\}$ mit $t^* \notin S \cup T$. Dann gilt: $\mathbf{N}$ ist sound genau dann, wenn $(\varphi\mathbf{N}, \iota)$ lebendig und beschränkt ist.

Lebendigkeit und Beschränktheit lassen sich algorithmisch mit vertretbarem Rechenaufwand bestimmen (vgl. [Eckl06]).

Die graphische Repräsentation von Petri-Netzen bildet Stellen als Kreise ◯, Transitionen als Rechtecke □ und die Kanten als Pfeile ⇆ ab. Marken werden als Punkte bzw. als Zahl in den Stellen ◉ dargestellt.

Zur Veranschaulichung wird der Beispiel-Prozess als Petri-Netz wiedergegeben:

[6] Ein Beweis ist in [AaHo98, S. 22-23] aufgeführt.

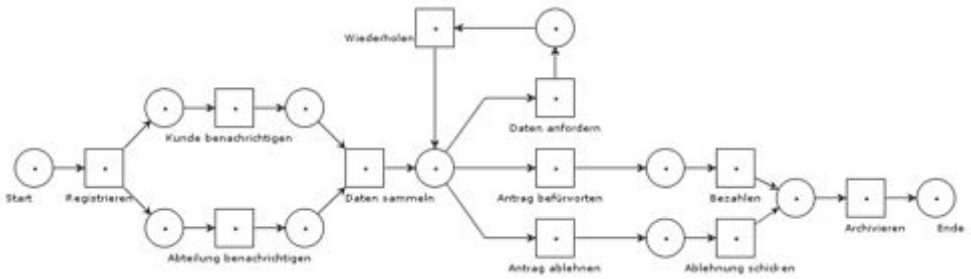
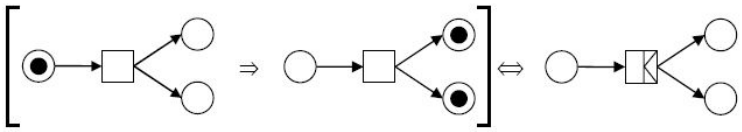


Abbildung 8: Petri-Netz des Beispiel-Prozesses

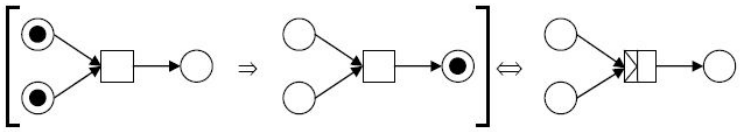
Um die Modellierung von Geschäftsprozessen mittels Petri-Netzen zu vereinfachen, werden im Folgenden symbolische Erweiterungen vorgestellt. Diese bestehen darin, dass Substrukturen eines Netzes durch einfachere Symbole ersetzt werden, die die graphische Darstellung einer einfachen Transition erweitern. Dabei ist zu beachten, dass die Schaltregel sinngemäß unter Berücksichtigung des originären ”Teilnetzes” anzuwenden ist.

Vergleicht man die Semantik bestimmter Petri-Netz-Strukturen mit den Kontrollstrukturen, die auf Seite 8 aufgezählt wurden, ergeben sich folgende Äquivalenzen, für die eigene Symbole eingeführt werden, wie sie u.a. in [AaHe02] zu finden sind:

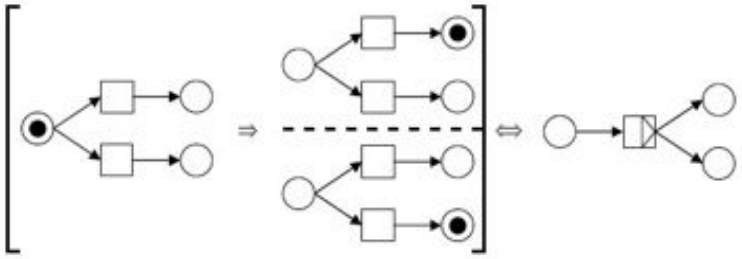
AND-Split: Aufspaltung in parallele Prozesspfade



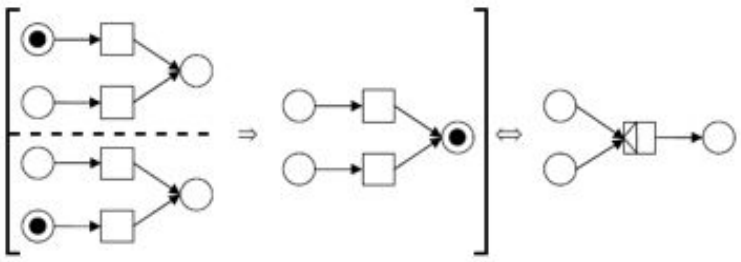
AND-Join: Zusammenführung paralleler Pfade



XOR-Split: Aufspaltung in alternative Pfade



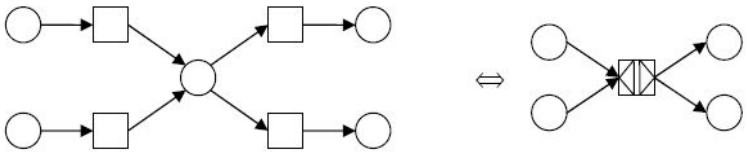
XOR-Join: Zusammenführung alternativer Pfade



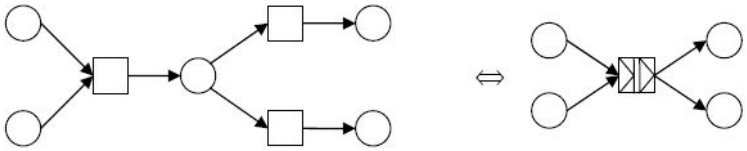
Weitere besondere Symbole:

AND-Split-Join:

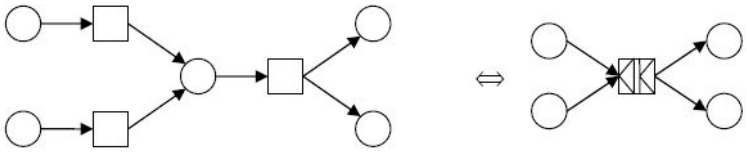
XOR-Split-Join:



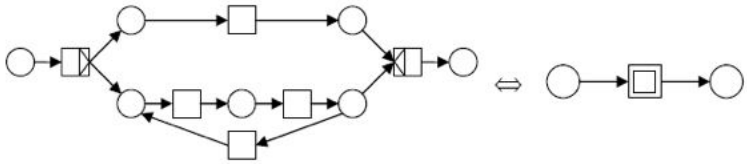
AND-Join-XOR-Split:



XOR-Join-AND-Split:



Subprozess: Hierarchische Gliederung



Die Ersetzung eines Teilnetzes durch einen Subprozess stellt eine Netztransformation, genauer eine Vergröberung, dar. Der umgekehrte Vorgang, bei dem ein Subprozess durch ein Teilnetz ersetzt wird, heißt Verfeinerung. Die zur theoretischen Untermauerung notwendigen Definitionen und Sätze sind im Anhang B ab Seite XIII zu finden.

Für die spätere Bezugnahme auf die verwendbaren Knotensymbole in Workflow-Netzen werden jetzt Kurzbezeichnungen eingeführt sowie spezielle Mengen definiert:

 s	 t	 sub	
 $t_{\_A}$	 $t_{A\_}$	 $t_{\_X}$	 $t_{X\_}$
 t_{AA}	 t_{XX}	 t_{XA}	 t_{AX}

Tabelle 1: Knotensymbole für Workflow-Netze

Menge der Knotentypen $\mathbb{K} := \{\mathtt{s}, \mathtt{t}, \mathtt{sub}, \mathtt{t\_A}, \mathtt{t_A\_}, \mathtt{t\_X}, \mathtt{t_X\_}, \mathtt{t_{AA}}, \mathtt{t_{XX}}, \mathtt{t_{XA}}, \mathtt{t_{AX}}\}$.

Menge der AND-Splits $\mathbb{A}_S := \{\mathtt{t\_A}, \mathtt{t_{AA}}, \mathtt{t_{XA}}\} \cup \{\mathtt{t} \mid |t\bullet| > 1\}$.

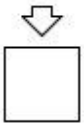
Menge der AND-Joins $\mathbb{A}_J := \{\mathtt{t_A\_}, \mathtt{t_{AA}}, \mathtt{t_{AX}}\} \cup \{\mathtt{t} \mid |\bullet t| > 1\}$.

Mit $\Upsilon : (S \cup T) \to \mathbb{K}, k \mapsto \Upsilon(k)$ wird einem Knoten sein **Typ** zugeordnet.

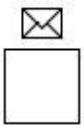
Es ist zu beachten, dass die XOR-Strukturen semantisch ein exklusives Oder (entweder A oder B, aber nicht beide) bezeichnen. Das logische ODER, das z.B. die Alternativen entweder A oder B oder A und B zuläßt, macht bei der formalen Begründung der Semantik Probleme[7].

Eine andere Erweiterung der reinen Petri-Netze sind Beschriftungen der Knoten, insbesondere der Transitionen. Dabei handelt es sich zum Beispiel um Funktionsbezeichnungen. Formal kann dafür eine Funktion definiert werden, die einem Knoten seine Bezeichnung zuordnet[8].

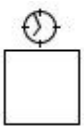
Weiterhin wird unterschieden, wie sich aktivierte Transitionen verhalten. Ohne weitere Angaben schaltet eine aktivierte Transition sofort. Um dieses Verhalten zu ändern und realitätsnaher zu machen, werden Trigger eingeführt. Davon gibt es drei Arten:



Ressourcen-Trigger
Die Aktivität startet, sobald eine Ressource verfügbar ist und ein unbearbeiteter Fall ansteht.



Externer Trigger
Die Aktivität wird durch ein externes Ereignis ausgelöst, z.B. die Ankunft einer Email oder eines Anrufs.



Zeit-Trigger
Die Aktivität startet erst nach Ablauf einer bestimmten Zeit bzw. mit Eintreten eines bestimmten Zeitpunkts.

Die Symbole werden oberhalb der Transitionen angeordnet.

Zusammenfassend soll noch mal das Geschäftsprozess-Beispiel als Workflow-Netz mit den erwähnten Erweiterungen dargestellt werden.

[7] Vgl. dazu die bereits erwähnte Diskussion zur "Busfahrer-Semantik".
[8] Vgl. H. M. W. VERBEEK. *Verication of WF-nets*. PhD thesis, TU Eindhoven, 2004.

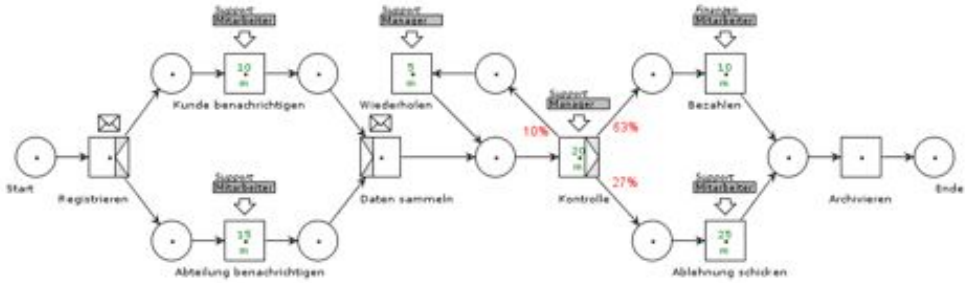


Abbildung 9: Beispiel-Prozess als Workflow-Netz mit Subprozess

Abschließend kann festgestellt werden, dass sich Petri-Netze sehr gut eignen, um Geschäftsprozesse zu modellieren. Sie sind in ihrer graphischen Notation einfach und vielseitig und in ihrer Semantik relativ leicht zu verstehen. [Vojn97] vergleicht zudem verschiedene Typen von Petri-Netzen bezüglich ihrer Eignung für quantitative Methoden.

Ihr formaler Charakter bietet anderen Modellierungssprachen gegenüber den Vorteil, dass die Semantik der Petri-Netze auf eindeutigen Begriffen fußt, was bereits zur Entwurfszeit die Verifikation des Modells ermöglicht. Dabei wird die Korrektheit des Netzes mittels formaler Methoden der Strukturanalyse überprüft. Hierzu existieren inzwischen zahlreiche, theoretisch untermauerte Tests.

Weiterhin existieren viele Software-Produkte, die die Modellierung von Geschäftsprozessen auf Basis von Petri-Netzen und deren Verifikation unterstützen.

Nachdem die Geschäftsprozesse modelliert wurden, müssen sie analysiert werden. Die Analyse verfolgt verschiedene Ziele. Dies ist Thema des nächsten Abschnittes.

2.2 Analyse von Geschäftsprozessmodellen

In der Überschrift zu diesem Abschnitt ist ausdrücklich die Rede von der Analyse von Modellen, nicht von den realen Geschäftsprozessen. Gründe dafür, warum es sinnvoll ist, Modelle zu verwenden, wurden bereits im Abschnitt 2.1.1 auf Seite 3 aufgezählt.

Geschäftsprozesse beschreiben die (bzw. eine) Wertschöpfungskette des Unternehmens. Sie sind also der Kern des Unternehmensmodells. Der ganze Erfolg einer Unternehmung hängt von ihnen ab. Das bedeutet zum einen, dass die Abläufe in der richtigen Weise organisiert werden müssen und die Unternehmensressourcen dort eingesetzt werden, wo sie nützen. Zum anderen muss diese Organisation aber auch effizient ablaufen, um wettbewerbsfähig zu bleiben.

Damit werden dann auch Durchlauf- und Wartezeiten, Ressourcenauslastung

und Kundenzufriedenheit ebenso wichtig wie die richtige Lenkung der Arbeitsschritte.

Aufgrund dieser hohen Bedeutung der Geschäftsprozesse für ein Unternehmen ist es nicht ratsam, sie zu implementieren ohne genau zu wissen, ob sie dem Unternehmenszweck dienen oder diesem eher abträglich sind infolge von Fehlplanung oder Ineffizienz. Da einmal laufende Prozesse nur sehr schwer und unter hohen Kosten geändert werden können, ist die genaue Analyse eines Modells im Vorfeld die einzige Möglichkeit, um diese Nachteile zu vermeiden. Eine methodische Unterstützung der Analyse im Zuge der Geschäftsprozessmodellierung ist daher von großem Nutzen.

Die Literatur ist reichhaltig an Theorien und es existieren zahlreiche praktische Methoden, um korrekte Geschäftsprozesse zu modellieren. Es gibt auch visuelle Unterstützung, die es gestattet, sich gewissermaßen im Zeitraffer den Prozess am Computerbildschirm vorführen zu lassen[9].

Bezüglich praktischer Verfahren zur Messung der Performanz von Geschäftsprozessen aber hält sich die Fachliteratur etwas zurück. Es gibt vor allem im Bereich des Operations Research viele Methoden, wie die Effizienz von Prozessmodellen gemessen werden kann. Hier tauchen vor allem Spreadsheet-Lösungen auf, die den Nachteil haben, nur für einen Einzelfall ausgelegt zu sein. Bei einer kleinen Änderung im Modell ist relativ viel Aufwand notwendig, um die Datenbasis des Tabellenkalkulationsprogramms anzupassen.

Eine andere Strategie ist die Verwendung von Simulationssprachen. Aufgrund ihrer Allgemeingültigkeit ist aber viel Interpretation nötig, um die richtigen Simulationsbausteine in der richtigen Weise anzuordnen, um den Geschäftsprozess semantisch korrekt abzubilden. Damit ist auch hier eine Korrektheitsprüfung und somit zusätzlicher Aufwand nötig. Aufgrund der Mehrdeutigkeit entstehen bei mehreren Modellierern u.U. verschiedene Modelle, die unterschiedliche Ergebnisse liefern oder Modellteile, die dann nicht zusammenpassen[10].

Wünscheswert wäre es, wenn Modellierung und Analyse in einer gemeinsamen Umgebung, aufeinander aufbauend und interaktiv stattfinden könnten. Es gibt zahlreiche Werkzeuge zur Modellierung von Geschäftsprozessen, die auch strukturelle Analysen unterstützen. Das bereits in der Einleitung erwähnte Tool "WoPeD" ist so ein Programm, das Geschäftsprozesse mit Workflow-Netzen modelliert und eine semantische Analyse durchführen kann[11]. Performance-Analysen aber sind zurzeit nicht möglich. Hier besteht ein Defizit, das durch diese Diplomarbeit behoben werden soll.

[9] Siehe dazu z.B. die Beschreibung der Werkzeuge der ARIS-Plattform: `http://www.ids-scheer.de/`

[10] Ein Beispiel für solch eine Simulationssprache ist GPSS (= **G**eneral **P**urpose **S**imulation **S**ystem). Siehe auch `http://www.webgpss.com/ENG/`.

[11] Vgl. [Land03], [FrLa03], [Frey05] und [Eckl06].

In diesem Abschnitt wird nach einer allgemeinen Diskussion des Analysebegriffes der Schwerpunkt auf die quantitative Analyse von Geschäftsprozessen gelegt. Er dient der Vorbereitung der beiden folgenden Kapitel über zwei spezielle Verfahren der quantitativen Analyse: der Kapazitätsplanung und der Simulation.

2.2.1 Zum Begriff der Analyse

Die Analyse beschreibt das systematische Untersuchen eines Objektes vor dem Hintergrund einer spezifischen Fragestellung. Dabei wird ein komplexer Untersuchungsgegenstand in seine einfacheren Bestandteile zerlegt, auf diese geeignete Analysemethoden angewendet und unter Berücksichtigung der gegenseitigen Beziehungen der Elemente Rückschlüsse auf das Verhalten des Gesamtsystems gezogen.

Im Hinblick auf die Anzahl der Elemente des Gesamtobjektes spricht man von hoher oder niedriger Komplexität. Bestehen zwischen den einzelnen Elementen zahlreiche, ggf. auch verschiedenartige Beziehungen, spricht man von einem komplizierten System.

Geschäftsprozesse zeichnen sich i.d.R. durch eine hohe Komplexität aufgrund der Größe, d.h. der Anzahl der Einzelaktivitäten, und eine relativ geringe Kompliziertheit aus. Aktivitäten mit zahlreichen Eingangs- oder Ausgangskanten sind selten.

Es gibt grundsätzlich zwei verschiedene Ansätze bei der Analyse von Geschäftsprozessen (vgl. [HaSt04, S. 97-125]). Die **qualitative Analyse** beschäftigt sich mit der Güte des Modells, also der Aussagekraft der Erkenntnisse über das Modell bezüglich des realen Prozesses. Hier finden sich Methoden der Validierung und der Verifikation wieder.

Validierung überprüft die semantische Übereinstimmung des Modells mit der Realität. Bei der Verifikation wird die strukturelle Korrektheit des Modells nachgewiesen.

Im Gegensatz dazu geht die **quantitative Analyse** von einem validierten und verifizierten Geschäftsprozessmodell aus, betrachtet Leistungsgrößen und bewertet die Effizienz des Geschäftsprozesses. Hier finden vorrangig statistische Analysemethoden Anwendung.

2.2.2 Quantitative Analysemethoden

Die quantitative Analyse hat zum Ziel, die Effizienz eines Systems, wie z.B. ein Geschäftsprozess, zu messen. Es existiert eine Vielzahl von Messgrößen, die

Auskunft über den Wirkungsgrad geben können, je nach Zielsetzung und Perspektive.

[JoFr96] nennen besipeilhaft zeitbezogene, kostenbezogene und Zuverlässigkeitsmaße und weisen darauf hin, dass solche "Grundgrößen" auch kombinert werden können zu abgeleiteten Maßen.

Bei den zeitbezogenen Maßen werden folgende Perspektiven und Messgrößen unterschieden:

- Kundenperspektive ➔ *Antwortzeit*
  Zeit zwischen Auftragsvergabe und Ende der Auftragsausführung. Sie ist die Summe aus Bearbeitungszeit und Wartezeiten für diesen Auftrag.
  Im Beispiel ist sie die Zeit, die vergeht vom Absenden der Email bis zum Zahlungseingang oder dem Erhalt des Ablehnungsbescheides.

- Prozessperspektive ➔ *Durchlaufzeit*
  Zeit, die benötigt wird, um eine konkrete Instanz eines Prozesses zu bearbeiten. Sie beschreibt eine interne Sichtweise.
  Die Zeit vom Lesen der Email bis zur Tätigung der Überweisung bzw. dem Abschicken der Ablehnung ist die Durchlaufzeit für eine Beschwerdebearbeitung.

- Produktperspektive ➔ *Bearbeitungszeit*
  Zeit, in der tatsächlich an der Ausführung eines Auftrages gearbeitet und damit ein Bearbeitungsfortschritt vollzogen wird. Sie ist die Antwortzeit abzüglich Wartezeiten.

- Systemperspektive ➔ *Durchsatz*
  Anzahl der abgearbeiteten Aufträge pro Zeiteinheit. Es handelt sich um ein Maß für die Kapazität des Prozesses. Der Durchsatz ist abhängig von den verwendeten Ressourcen und unabhängig von der Rate, mit der neue Fälle in den Prozess eintreten.

- Ressourcenperspektive ➔ *Auslastung*
  Prozentsatz der Zeit, die die Ressource beschäftigt war, an der Gesamtzeit, die die Ressource zur Verfügung stand. Hohe Auslastungsgrade sind erwünscht, um Leerkosten zu vermeiden. Sie können aber auch ein Zeichen für den Bedarf weiterer Ressourcen sein.

Die verschiedenen Messgrößen stehen in einem engen Zusammenhang miteinander und korrelieren teils positiv, teils negativ. Als Beispiel für ein Maß der Zuverlässigkeit soll die Qualität gelten. Wie diese zu messen ist, bestimmt sich

objektspezifisch und kann allgemein nicht angegeben werden. Als kostenbezogene Größen können beispielhaft variable und fixe Kosten betrachtet werden. Damit ergibt sich auszugsweise die nachfolgende synoptische Darstellung der Zusammenhänge[12].

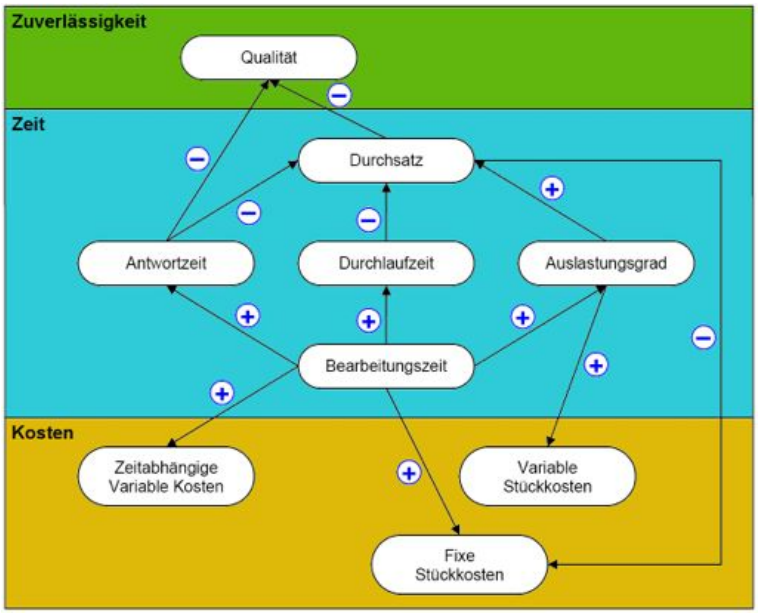


Abbildung 10: Zusammenhang zwischen Effizienzmaßen

Die Zusammenhänge zwischen verschiedenen Effizienzmaßen können perspektivenübergreifend verallgemeinert werden. Ein wichtiges und nützliches Ergebnis solcher Verallgemeinerungen ist Little's Gesetz[13], welches besagt, dass die Populationsgröße (z.B. Anzahl von Fällen im System) gleich dem Produkt aus Durchsatz[14] und Verweilzeit ist. Bemerkenswert daran ist die Unabhängigkeit von anderen Einflussgrößen wie der Art der Häufigkeitsverteilung oder der Bedienreihenfolge von Fällen. Eine kurze Diskussion zu Little's Gesetz ist im Anhang C zu finden.

Die quantitativen Methoden werden bei [JoFr96] in drei Klassen unterteilt:

1. Messungen:
   Messungen können nur an realen Prozessen vorgenommen werden. Ihre Bedeutung bei der Analyse von Geschäftsprozessmodellen liegt eher darin, dass bestimmte empirisch ermittelte Daten wie Bedienzeiten und ihre Häufigkeitsverteilungen oder mittlere Ressourcenauslastungen als Inputgrößen für die anderen Methoden beim anschließenden Prozess-Redesign verwendet werden können.

[12] Ein Pfeil von A nach B bedeutet, dass A einen Einfluss auf B hat. Die Richtung der Korrelation ist als Vorzeichen (+ oder -) angegeben.

[13] Vgl. J. D. C. LITTLE. *A proof of the queuing formula* $L = \lambda W$. Operations Research, vol. 9, no. 3, S. 383-387, 1961.

[14] In einem stabilen System ist der Durchsatz gleich der Ankunftsrate.

2. (Quantitative) Simulation:
   Simulation ist die Imitation eines realen Prozesses und bedeutet i.d.R. die ”Ausführung” eines Modells auf einem Computer (vgl. [HiLi05, S. 930]). Dieses Verfahren wird auf Modelle von stochastischen Systemen angewandt - wie eben auch Geschäftsprozessmodelle -, also solchen Systemen, deren Voranschreiten mittels Wahrscheinlichkeiten bestimmt wird.
   Die quantitative Simulation ist ein Verfahren zur Beobachtung numerischer Größen, die aus der Sammlung von Daten während des Simulationslaufes berechnet werden. Im Unterschied dazu dient eine qualitative Simulation der Veranschaulichung der Abläufe und findet eher bei der Validierung des Modells Anwendung.
   Die Vorteile der Simulation liegen in der Kostenersparnis und relativ einfachen Ausführbarkeit sowie ihrer Flexibilität. Größter Nachteil ist der teilweise enorme Zeitaufwand, der aufgrund der komplexen Berechnungen entsteht.
   Die detaillierte Behandlung der Simulation findet im Kapitel 4 ab Seite 48 statt. Die Umsetzung in Java wird im Abschnitt 5.2 beschrieben.

3. Analytische Lösungsverfahren:
   Exakte Ergebnisse können mithilfe von mathematischen Verfahren erzielt werden. ”Exakt” ist dabei relativ: Denn die Präzision der Ergebnisse hängt von der Genauigkeit der Eingabeparameter des Modells ab, die praktisch immer Mittelwerte darstellen. Mitunter liefern diese Verfahren aber auch nur Abschätzungen.
   Analytische Methoden sind anderen Methoden vorzuziehen, weil sie auch überprüfbare Ergebnisse liefern. Die Rechnungen führen bei gleichen Daten immer zum gleichen Ergebnis. Bei der Simulation dagegen ist jeder Simulationslauf von einer Folge von Zufallsgrößen bestimmt und liefert daher unterschiedliche Ergebnisse.
   Ein großes Problem bei der praktischen Anwendung dieser Klasse von Verfahren ist aber, dass sie nur unter sehr restriktiven Voraussetzungen anwendbar sind. Die so entstehenden mathematischen Modelle nicht dann nicht mehr sehr realitätsnah und liefern kaum brauchbare Ergebnisse. Außerdem besteht auch hier das Problem der Komplexität, die teilweise eine analytische Behandlung (z.B. wegen eines unendlich großen Zustandsraumes) unmöglich machen.
   Ein Beispiel für ein analytisches Modell wird im Kapitel 3 vorgestellt und seine Umsetzung im Abschnitt 5.1 besprochen.

Bei der Auswahl der anzuwendenden Verfahren müssen ihre Vor- und Nachteile gegeneinander abgewogen werden. Es kommt dabei auf die Zielsetzung der

Analyse an. [JoFr96] nennen drei formale Kriterien für die Auswahl der richtigen Methode.

- *Genauigkeit*: Diese kann durch fehlerbehaftete Verfahren und Ungenauigkeiten oder Fehler bei der Modellierung beeinträchtigt werden.
- *Robustheit*: Damit wird die Eigenschaft beschrieben, für alle Kombinationen von Parameterwerten relativ genaue Ergebnisse zu liefern. Ein Verfahren ist wenig zweckmäßig, wenn es zwar einige sehr genaue Ergebnisse liefert, aber für bestimmte (aber realistische) Inputwerte Fehler erzeugt.
- *Effizienz*: Die Qualität der Ergebnisse, die eine Methode liefert, muss in einem zweckmäßigen Verhältnis zum Aufwand stehen, der zu ihrer Gewinnung notwendig ist.

2.2.3 Quantitative Analyse von Geschäftsprozessen

Es gibt drei Analyseverfahren, die für die quantitative Analyse von Geschäftsprozessen besonders relevant sind: Markov-Analyse, Warteschlangentheorie und Simulation[15].

Markov-Ketten

Geschäftsprozesse können mathematisch als stochastische Prozesse aufgefaßt werden. Das heißt, sie befinden sich zu einem Zeitpunkt t in genau einem von endlich vielen Zuständen $\{1, 2, ..., s\}$. Mit Voranschreiten der Zeit wechseln sie ihren Zustand zu bestimmten Zeitpunkten $\{0, 1, 2, ...\}$. Eine Folge von Zufallsvariablen X, die eine Größe von Interesse bezüglich des Geschäftsprozesses mißt (z.B. Anzahl der Kunden im Prozess) und mit den Zeitpunkten des Zustandswechsels indiziert ist, heißt (zeitdiskreter) **stochastischer Prozess** $(X_t)_{t\in\mathbb{N}_0}$.

Die Betrachtung stochastischer Prozesse dient der Vorhersage von zukünftigen Ereignissen aufgrund der zuletzt angenommenen Zustände und ihrer Wahrscheinlichkeitsverteilungen. Dazu bezeichne $P(X_t = i_t)$ die Wahrscheinlichkeit, mit der die Zufallsvariable X zum Zeitpunkt t den Wert (Zustand) i_t annimmt. Ein stochastischer Prozess $(X_t)_{t\in\mathbb{N}_0}$ mit der Eigenschaft

$$P(X_{t+1} = i_{t+1} \mid X_t = i_t \wedge X_{t-1} = i_{t-1} \wedge ... \wedge X_1 = i_1 \wedge X_0 = i_0) \\ = P(X_{t+1} = i_{t+1} \mid X_t = i_t)$$

[15] Für die folgenden Ausführungen vgl. [AaHe02, S. 118-133], [HiLi05, Kap. 16, 17 und 20], [Wins04, Kap. 17, 20-22].

für die Zeitpunkte $t = 0, 1, 2, ...$ und alle Zustände $\{1, 2, ..., s\}$ heißt **Markov-Kette** (erster Ordnung). Die Eigenschaft wird als *Gedächtnislosigkeit* oder *Markov-Eigenschaft* bezeichnet und besagt, dass die Wahrscheinlichkeit, im Zustand i_{t+1} zu landen, nur vom zuletzt eingenommenen Zustand abhängt.

Die Wahrscheinlichkeiten $P(X_{t+1} = j \mid X_t = i) =: p_{ij}(t) \wedge i, j \in \{1, 2, ..., s\}$ heißen **Übergangswahrscheinlichkeiten**. Gilt zudem, dass $p_{ij}(t) = p_{ij}\ \forall i, j \in \{1, 2, ..., s\}$, also dass die Übergangswahrscheinlichkeiten vom Zeitpunkt t unabhängig sind, spricht man von einer **homogenen Markov-Kette**. Die Unabhängigkeit von der Zeit kann praktisch nur in stabilen Systemen angenommen werden und stellt meist eine Näherung dar.

Die Übergangswahrscheinlichkeiten p_{ij} können als $s \times s$-Matrix angeordnet werden, wobei die Ausgangszustände in den Zeilen und die Endzustände in den Spalten stehen. $\mathbf{M} := (p_{ij})$ ist die **Übergangsmatrix** und es gilt: $p_{ij} \geq 0 \wedge \sum_{j=1}^{s} p_{ij} = 1\ \forall i \in \{1, 2, ..., s\}$. Der letzte Teil der Aussage bedeutet, dass die Wahrscheinlichkeit, das das System vom Zustand i in irgendeinen Zustand übergeht - das entspricht der Zeilensumme -, gleich Eins ist.

Als **Wahrscheinlichkeitsverteilung** auf dem Zustandsraum $\{1, 2, ..., s\}$ wird ein Vektor $\mathbf{q} \in [0..1]^s$ mit $\mathbf{q} = [q_1, q_2, ..., q_s]$ bezeichnet, wenn $q_i = P(X_t = i)$ und $\sum_{i=1}^{s} q_i = 1$ gilt. Der spezielle Vektor $\mathbf{a} = [P(X_0 = 1), P(X_0 = 2), ..., P(X_0 = s)]$ heißt **Anfangsverteilung**.

Die Frage nach der Wahrscheinlichkeit, vom Zustand i zum Zustand j mit einem Übergang zu gelangen, läßt sich nun sehr leicht beantworten, indem man das Produkt $\mathbf{q} \cdot \mathbf{M}$ bildet. Es ergibt sich:

$$\mathbf{q} \cdot \mathbf{M} = [q_1, q_2, ..., q_s] \cdot \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix}$$

$$= \begin{bmatrix} q_1 p_{11} + q_2 p_{21} + \ldots + q_s p_{s1} \\ q_1 p_{12} + q_2 p_{22} + \ldots + q_s p_{s2} \\ \vdots \\ q_1 p_{1s} + q_2 p_{2s} + \ldots + q_s p_{ss} \end{bmatrix}^T$$

Noch interessanter ist die Frage, wie hoch die Wahrscheinlichkeit ist, vom Zustand i ausgehend nach n Zustandsübergängen im Zustand j zu landen. Zunächst ergibt sich aus der Eigenschaft der Gedächtnislosigkeit, dass $P(X_{t+n} = j \mid X_t = i) = P(X_n = j \mid X_0 = i) =: p_{ij}^n$ gilt. Es läßt sich zeigen, dass sich die p_{ij}^n durch Bildung der n-ten Potenz der Übergangsmatrix berechnen lassen.

Um nun die Wahrscheinlichkeit zu bestimmen, mit der ausgehend vom Anfangs-

zustand der Zustand *j* in *n* Schritten erreicht wird, bildet man das Produkt $\mathbf{a} \cdot \mathbf{M}^n$ und liest das Ergebnis in der *j*-ten Spalte ab.

Eine Markov-Kette mit nicht zu vielen Zuständen kann graphisch als Zustandsübergangsdiagramm dargestellt werden. Dazu werden die Zustände als Kreise und die Übergänge als Pfeile gezeichnet. Ein Pfad im Diagramm entspricht dann einer Folge von Zustandsübergängen.

Die Anwendung der Markov-Analyse auf Geschäftsprozesse, die als Workflow-Netz modelliert sind, ergibt sich dann folgendermaßen:

Sei $\mathbf{N} = (S, T, F, \iota)$ ein Workflow-Netz mit Anfangsmarkierung ι, das sound ist. Es wird dann die Menge $[\iota\rangle$ aller Folgemarkierungen der Anfangsmarkierung bestimmt. Sie heißt Erreichbarkeitsmenge und ist endlich, da $\mathbf{N}$ beschränkt ist (vgl. [Baum96, Satz 6.2, S. 132]). Der Graph $\mathbf{E} = (E, K)$ mit $E = [\iota\rangle \wedge K = \{(\mu, \mu', t) \in E \times E \times T \mid \mu \xrightarrow{t} \mu'\}$ heißt Erreichbarkeitsgraph (vgl. [HaSt04, S. 117]). Wird der Erreichbarkeitsgraph erweitert um die Funktion $\nu : K \rightarrow [0..1], (\mu, \mu', t) \mapsto p_{\mu\mu'}$, so dass die Werte von ν die Übergangswahrscheinlichkeiten von Markierung μ zu Markierung μ' durch Schalten der Transition t angibt, dann stellt der Graph (E, K, ν) das Zustandsübergangsdiagramm einer homogenen Markov-Kette dar.

Die Übergangswahrscheinlichkeiten müssen empirisch ermittelt und zusätzlich zum Geschäftsprozessmodell angegeben werden. Dann lassen sich verschiedenste Wahrscheinlichkeiten berechnen und mit Kosten- oder Ertragsfunktionen oder Zeitangaben kombinieren, um Aussagen über die Effizienz des Geschäftsprozesses zu machen. Wird ein stabiles - d.h. in einem Gleichgewicht befindliches - System angenommen, kann unter bestimmten Voraussetzungen ein Grenzübergang in der Zeit nach ∞ durchgeführt werden, wobei die p_{ij} festen (oder stabilen) Wahrscheinlichkeiten π_j zustreben. Bei Geschäftsprozessmodellen sind die Voraussetzungen aber i.d.R. nicht erfüllt.

Fragestellungen, die unabhängig von stochastischen Untersuchungen sind, lassen sich nicht analysieren. Darüber hinaus hängt die Effizienz dieses Verfahrens stark von der Ermittlung des Erreichbarkeitsgraphen ab. Dies stellt sich aufgrund der kombinatorischen Vielfalt der Berechnungen als sehr komplexes Unterfangen heraus. Bei relativ komplexen Geschäftsprozessen ist dieses Verfahren daher eher ungeeignet.

Warteschlangentheorie

Beim Durchlaufen eines Geschäftsprozesses wird ein Fall oftmals keine freie Ressource vorfinden zum Zeitpunkt der Weiterleitung zu einer neuen Aufgabe. Das heißt, dieses Work Item kann im Augenblick nicht bearbeitet werden und

muss warten. Beispiele aus dem Alltag sind die Auftragsablage auf dem Schreibtisch oder die Warteschlange am Geldautomat.

Die begrenzte Verfügbarkeit von Ressourcen stellt den hauptsächlichen Flaschenhals bei Geschäftsprozessen dar[16]. Sind zu wenig Ressourcen vorhanden, kommt es mitunter zu sehr langen Wartezeiten, die mitunter ein Vielfaches der eigentlichen Bearbeitungszeit ausmachen kann.

Zunächst folgt daraus, die Anzahl der Ressourcen einfach zu erhöhen, so dass für jeden Fall, der sich gerade einfindet, eine freie Ressource zur Verfügung steht. Das verringert die Wartezeit des Falles und erhöht den Service-Level, also das Verhältnis der gesamten Bearbeitungszeit für diesen Fall zur Gesamtdurchlaufzeit durch den Prozess und erhöht die Kundenzufriedenheit. Für die Ressourcen gilt aber Gegenteiliges: Sie warten die meiste Zeit auf einen Fall, womit sich der Auslastungsgrad - das ist das Verhältnis von Bedienzeit zur Gesamtzeit, die die Ressource zur Verfügung steht - verringert.

Anschaulich bedeutet das beispielsweise, dass die Warteschlange an der einzigen Kasse im Supermarkt, die gerade geöffnet ist, sehr lang werden kann, wenn sich zurzeit sehr viele Kunden im Geschäft befinden. Würden während der gesamten Öffnungszeit alle Kassen besetzt sein, wären die Warteschlangen sehr viel kürzer. Ein anderes Beispiel ist die Bahn, die zu bestimmten Tageszeiten nicht jedem Passagier einen Sitzplatz anbieten kann. Würden weitere Wagons angehängt, würde sich die Kapazität erhöhen. Allerdings hat die Bahn zu anderen Zeiten, insbesondere in den frühen Morgenstunden, so wenige Kunden, dass fast alle Sitzplätze leer bleiben.

Mit relativ einfachen empirischen Untersuchungen kann man das Aufkommen von Fällen pro Zeiteinheit messen. Mit dieser Information wäre ein Manager in der Lage, eine so große Kapazität zur Verfügung zu stellen, dass alle Fälle sofort bedient werden könnten.

ABER: Das kostet Geld. Die Kassiererinnen im Supermarkt werden pro Arbeitsstunde bezahlt, haben aber die meiste Zeit nichts zu tun und tragen somit wenig zum Geschäftserfolg bei. Um die Produktivität einer Kassiererin zu erhöhen, muss die Anzahl ihrer Kollegen reduziert werden. Die Bereitstellung weiterer Wagons bei der Bahn ist ebenfalls kostspielig. Denn die Wagons müssen fahrtüchtig gehalten und angekoppelt werden; außerdem ist eine höhere Anzahl dieser teuren Vermögensgegenstände notwendig. Eine hohe Anzahl von Passagieren pro Wagon senkt die relativen Fixkosten.

Es besteht also ein Zielkonflikt zwischen der Bereitstellung ausreichender Kapazitäten, um die Wartezeiten der Kunden zu reduzieren und damit die Qualität der

[16]Eine weitere Ursache für Wartezeiten sind Synchronisationspunkte im Prozess, wo auf die Beendigung aller parallel ausgeführten Instanzen eines Falles gewartet werden muss, bevor es weiter gehen kann.

Dienstleistung und daraus folgend die Kundenzufriedenheit zu erhöhen, und den damit verursachten Kosten.

Die Warteschlangentheorie liefert nützliche analytische Modelle, um diesen Zielkonflikt möglichst effizient zu lösen. Dazu wird jeder Ort, an dem eine Dienstleistung erbracht oder ein Produkt erzeugt wird, als Server betrachtet, dem eine Warteschlange für eingehende Aufträge vorangeht.

Ein Warteschlangenmodell ist eine Abstraktion einer Bedieneinrichtung mit folgendem grundsätzlichen Aufbau:

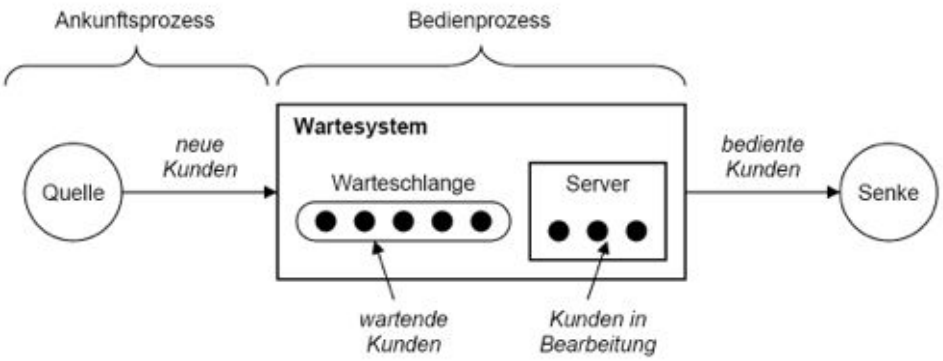


Abbildung 11: Elemente eines Wartesystems

Die Charakteristika der Bestandteile sind:

- *Quelle*:
  - **Größe**: die (potentielle) Anzahl der Kunden, die den angebotenen Service in Anspruch nehmen könnten oder wollten. Sie kann begrenzt oder unbegrenzt sein. Wenn es unmöglich ist, eine obere Schranke anzugeben, auch wenn die Anzahl nur endlich sein kann, wird die Größe als unbegrenzt angenommen, da es bei jeder Festlegung einer Schranke noch einen weiteren Kunden geben könnte, der den Service doch beansprucht[17]. Außerdem ist die Rechnung mit unbegrnezten Quellen deutlich einfacher.
- *Ankunftsprozess*:
  - **Mittlere Ankunftsrate**: das ist ein empirisch ermittelter Mittelwert für die Anzahl ankommender Kunden pro Zeiteinheit.

[17]Zum Beispiel wird für die Warteschlange an der Supermarktkasse von einer (potentiell) unbegrenzten Quelle ausgegangen, da niemand mit Sicherheit sagen kann, wieviele Kunden am Tag maximal dort anstehen werden.

  - **Verteilung der Zwischenankunftszeiten**: die Wahrscheinlichkeitsverteilung für die Ankünfte. Die Anzahl ankommender Kunden in einem vorgegebenen Zeitintervall resp. die Zeitdifferenz zwischen zwei Ankünften (Zwischenankunftszeit) sind stochastisch um den Mittelwert verteilt.
  - **Kundenverhalten**: Wenn der Kunde die Wahl hat zwischen verschiedenen (physischen) Warteschlangen, wählt er die kürzere oder wählt er willkürlich? Wenn die Warteschlange zu lang ist oder der Kunde bereits sehr lange wartet, verläßt der die Warteschlange oder bleibt er? Diese Größe ist nicht quantifizierbar und es wird daher i.d.R. mit einer Warteschlange gerechnet, in der der Kunde so lange wartet, bis er bedient wird.

- *Warteschlange*:

  - **Größe**: die Aufnahmekapazität, also die maximale Anzahl von Kunden, die in der Warteschlange Platz haben. Auch hier wird zwischen begrenzt und unbegrenzt unterscheiden; es gilt das bereits oben Gesagte.
  - **Bedienreihenfolge**: auch Warteschlangendisziplin genannt. Damit wird beschrieben, in welcher Reihenfolge die wartenden Kunden aus der Warteschlange geholt werden:
    - ➤ First in - First out (FIFO): Wer zuerst kommt, wird zuerst bedient.
    - ➤ Last in - First out (LIFO): Wer zuletzt kommt, wird zuerst bedient.
    - ➤ Serice in random order (SIRO): Aus der Menge wartender Kunden wird einer zufällig ausgewählt.
    - ➤ Shortest Processing Time (SPT): Der Kunde mit der kleinsten Bedienzeit ist der nächste.
    - ➤ Service in priority driven order: Die Menge der Kunden wird in Prioritätsklassen eingeteilt. Kunden mit höherer Priorität werden bevorzugt. Innerhalb einer Prioritätsklasse kommt eine der anderen Methoden zur Anwendung; üblich ist FIFO.

- *Server*:

  - **Anzahl der verfügbaren Ressourcen**: sie kann fest oder flexibel sein, Eins oder größer als Eins. Je nach Ressourcenmodell ist einem

Server eine Ressource fest zugeordnet oder es sind mehrere Ressourcen zugeordnet, so dass eine parallele Bearbeitung von Kunden möglich ist. Bei einer flexiblen Zuordnung schwankt die Anzahl mit der Zeit zwischen Null und dem Maximum.

- **Mittlere Bedienzeit**: Die Zeit vom Beginn bis zum Ende der Bearbeitung eines Kunden ist die Bedienzeit. Aus Erfahrungswerten oder Messungen ergibt sich ein Mittelwert.
- **Verteilung der Bedienzeit**: eine Wahrscheinlichkeitsverteilung ähnlich der Verteilung für die Ankünfte.

Weiterhin wird von den Wahrscheinlichkeitsverteilungen der Zwischenankunfts- und Bedienzeiten verlangt, dass aufeinanderfolgende Werte voneinander unabhängig sind und dieselbe Verteilung besitzen.

Wartesysteme lassen sich je nach Festlegung der o.g. Spezifika in Klassen einteilen. Zu ihrer Benennung verwendet man die sogenannte **Kendall-Lee-Notation** (oder kurz Kendall-Notation). Danach ist ein Wartesystem $\mathbf{Q} = (\mathcal{A}, \mathcal{B}, \mathcal{R}, \mathcal{K}, \mathcal{P}, \mathcal{D})$ ein 6-Tupel mit

$\mathcal{A}$ Verteilung der Zwischenankunftszeiten
$\mathcal{A} \in V = \{\mathsf{M}, \mathsf{D}, \mathsf{H}, \mathsf{E_k}, \mathsf{PH}, \mathsf{GI}\}$, womit folgende Verteilungen bezeichnet werden:
M = Markovian (Exponential) distribution = Exponentialverteilung
D = Deterministic distribution = Konstante Verteilung
H = Hyperexponentialverteilung
$\mathsf{E_k}$= Erlang distribution = Erlang-Verteilung
PH = Phasenverteilung
GI = General independent distribution = beliebige Verteilung

$\mathcal{B}$ Verteilung der Bedienzeiten
$\mathcal{B} \in (V \setminus \{\mathsf{GI}\}) \cup \{\mathsf{G}\}$ mit G = (wie GI) beliebige Verteilung

$\mathcal{R}$ Anzahl der Ressourcen, $\mathcal{R} \geq 1$

$\mathcal{K}$ Kapazität des Wartesystems = maximale Anzahl von wartenden und bedienten Kunden zugleich
ohne Angabe gilt: $\mathcal{K} = \infty$

$\mathcal{P}$ Populationsgröße = Größe der Quelle
ohne Angabe gilt: $\mathcal{P} = \infty$

$\mathcal{D}$ Bedienreihenfolge
$\mathcal{D} \in \{\mathsf{FIFO}, \mathsf{LIFO}, \mathsf{SIRO}, \mathsf{SPT}, \mathsf{GD}\}$

GD = General queue discipline
ohne Angabe gilt: $\mathcal{D} = \mathsf{FIFO}$

Üblicherweise wird eine Kurzform verwendet, die nur die Angaben von $\mathcal{A}$, $\mathcal{B}$ und $\mathcal{R}$ enthält. Die anderen Eigenschaften erhalten dann ihre Standardwerte.

Die Arbeitsgrößen der Warteschlangentheorie sind:

λ_n mittlere Ankunftsrate unter der Bedingung, dass sich bereits n Kunden im System befinden.
Ist $\lambda_n = const.\ \forall n$, wird dafür λ geschrieben. $\frac{1}{\lambda}$ ist die Zwischenankunftszeit.

μ_n mittlere Bedienrate (Anzahl abgearbeiteter Kunden pro Zeiteinheit), wenn n Kunden im System sind.
Ist $\mu_n = const.\ \forall n \geq 1$, wird hierfür μ geschrieben. Dann gilt $\mu_n = s\mu$, wenn s die Anzahl beschäftigter Ressourcen ist. $\frac{1}{\mu}$ ist die mittlere Bedienzeit.
Es existiert ein μ für jeden Server.

$L(t)$ Anzahl von Kunden im System zum Zeitpunkt t. Im stabilen System ist L unabhängig von t.

$P_n(t)$ Wahrscheinlichkeit, dass sich zum Zeitpunkt t genau n Kunden im System befinden ($P(L(t) = n)$). Im stabilen System ist P_n unabhängig von t.

Nach einer gewissen Einpendelphase geht das System in einen stabilen Zustand über, in dem die Beziehungen der Größen untereinander zeitunabhängig sind. Dann gilt:

$$L = \sum_{n=0}^{\infty} nP_n \ \wedge \ L_q = \sum_{n=s}^{\infty} (n-s)P_n$$

wobei L_q die mittlere Warteschlangenlänge und s die Anzahl der beschäftigten Ressourcen ist. Mit L_s gleich der Anzahl der bedienten Kunden gilt: $L = L_q + L_s$.

Mit W wird die mittlere Verweilzeit im System, also die mittlere Gesamtdurchlaufzeit, und mit W_q die mittlere Wartezeit bezeichnet. Dann gilt $W = W_q + \frac{1}{\mu}$.

Der Auslastungsgrad ρ eines Servers läßt sich nach $\rho = \frac{\lambda}{s\mu}$ berechnen.

Nach dem Gesetz von Little folgt der für die Warteschlangentheorie wichtige Zusammenhang:

$$\begin{aligned} L &= \lambda W \\ L_q &= \lambda W_q \\ L_s &= \frac{\lambda}{\mu} \end{aligned}$$

Damit ist es möglich, bei Kenntnis von nur einer Größe, alle anderen zu bestimmen. Es folgt ein auszugsweiser Überblick über die Leistungsparameter verschiedener Wartesysteme[18]:

Typ	ρ	L	L_q	L_s	W	W_q
$(\mathsf{M},\mathsf{M},1)$	$\frac{\lambda}{\mu}$	$\frac{\rho}{1-\rho}$	$\frac{\rho^2}{1-\rho}$	ρ	$\frac{1}{\mu-\lambda}$	$\frac{\rho}{\mu-\lambda}$
$(\mathsf{M},\mathsf{M},\mathsf{s})$	$\frac{\lambda}{s\mu}$	$\frac{p_j^s\rho}{1-\rho}+\frac{\lambda}{\mu}$	$\frac{p_j^s\rho}{1-\rho}$	$\frac{\lambda}{\mu}$	$\frac{p_j^s}{s\mu-\lambda}+\frac{1}{\mu}$	$\frac{p_j^s}{s\mu-\lambda}$
$(\mathsf{M},\mathsf{G},\infty)$ $(\mathsf{GI},\mathsf{G},\infty)$	$\frac{\lambda}{\mu}$	$\frac{\lambda}{\mu}$	0	$\frac{\lambda}{\mu}$	$\frac{1}{\mu}$	0
$(\mathsf{M},\mathsf{G},1)$	$\frac{\lambda}{\mu}$	$\frac{\lambda^2\sigma^2+\rho^2}{2(1-\rho)}+\rho$	$\frac{\lambda^2\sigma^2+\rho^2}{2(1-\rho)}$	ρ	$\frac{\lambda^2\sigma^2+\rho^2}{2\lambda(1-\rho)}+\frac{1}{\mu}$	$\frac{\lambda^2\sigma^2+\rho^2}{2\lambda(1-\rho)}$

Tabelle 2: Leistungsparameter von Wartesystemen

Es ist zu erkennen, wie sehr die Änderung einzelner Parameter das Verhalten des Systems ändert. Allerdings ist die Berechnung in konkreten Fällen sehr aufwendig.

Bei Geschäftsprozessen handelt es sich meistens um mehrere Server, die sequentiell oder parallel angeordnet sind. Zu ihrer Beschreibung verwendet man daher **Warteschlangen-Netzwerke**. Sie sind eine Erweiterung eines einfachen Wartesystems.

Die wichtigste Frage im Zusammenhang mit Warteschlangen-Netzwerken ist zunächst, welchen Wert die Ankunftsraten bei den verschiedenen Servern haben. Es kann intuitiv angenommen werden, dass die Ankunftsraten nachfolgender Server abhängig sind von den Ankunftsraten der Vorgänger.

Eines der wichtigsten Ergebnisse der Warteschlangentheorie ist die "**Äquivalenz-Eigenschaft**". Sie stellt eine Beziehung zwischen Input-Prozess und Output-Prozess eines Servers her und besagt[19]:

> Gegeben sei ein Wartesystem mit s Ressourcen und unbegrenzter Warteschlange. Die Ankünfte seien Poisson-verteilt mit Parameter λ und der Server habe eine exponential-verteilte Bedienzeit mit Parameter $\frac{1}{\mu}$ und es gelte $s\mu > \lambda$. Im Gleichgewichtszustand gilt dann für den Output-Prozess, dass auch er Poisson-verteilt ist mit Parameter λ.

[18] Dabei bedeutet: $p_j^s := P(j \geq s) = \frac{(s\rho)^s \pi_0}{s!(1-\rho)}$ mit

$$\pi_0 = \frac{1}{\sum_{i=0}^{s-1} \frac{(s\rho)^i}{i!} + \frac{(s\rho)^s}{s!(1-\rho)}}$$

[19] Ein Beweis ist angegeben in: P. J. BURKE. *The Output of a Queueing System*. Operations Research, 4 (6), S. 699-704, 1956.

Bei in Reihe angeordneten Wartesystemen mit exponential-verteilten Ankunft- und Bedienraten heißt das, das jedes Wartesystem dieselbe Ankunftsrate mit derselben Verteilung hat. Damit ist es möglich, alle Wartesysteme unabhängig voneinander als $(\mathsf{M}, \mathsf{M}, \mathsf{s})$- Modelle zu analysieren. Zudem ergibt sich für die Berechnung der Wahrscheinlichkeit, dass sich am Server 1 n_1, am Server 2 n_2, ... und am Server k n_k Kunden befinden, eine sehr einfache Produktformel:

$$P(N_1 = n_1 \wedge N_2 = n_2 \wedge ... \wedge N_k = n_k) = \prod_{i=1}^{k} P_{n_i}$$

Eine komplexere Art von Netzwerken, die ebenfalls eine Produktformel verwenden, sind Jackson-Netzwerke. Sie sind folgendermaßen definiert:

Ein **Jackson-Netzwerk** ist ein System von k Wartesystemen, wobei für alle $i \in \{1, 2, ..., k\}$ gilt:

1. Wartesystem i hat eine unbegrenzte Warteschlange.
2. Der Input-Prozess für *externe* Ankünfte am Wartesystem i ist Poisson-verteilt mit Parameter a_i.
3. Server i hat s_i Ressourcen und eine exponential-verteilte Bedienzeit-Verteilung mit Parameter $\frac{1}{\mu_i}$.
4. Eine Kunde, der Server i verläßt, wird zum Server j mit Wahrscheinlichkeit p_{ij} weitergeleitet und verläßt das System mit Wahrscheinlichkeit $q_i = 1 - \sum_{j=1}^{k} p_{ij}$.

Hier wird eine gewisse Verwandschaft zu Markov-Ketten sehr deutlich. Für Jackson-Netzwerke gelten folgende Eigenschaften[20]:

[20] Zum Beweis vgl. J. R. JACKSON. *Jobshop-Like Queueing Systems*. Management Science, 10 (1), S. 131-142, 1963.

- Im Gleichgewichtszustand verhält sich jedes Wartesystem j mit $j \in \{1, 2, ..., k\}$ eines Jackson-Netzwerkes als unabhängiges $(\mathsf{M}, \mathsf{M}, \mathsf{s_j})$-System mit der Ankunftsrate

$$\lambda_j = a_j + \sum_{i=1}^{k} \lambda_i p_{ij} \; \forall j$$

  wobei gilt: $s_j \mu_j > \lambda_j$.

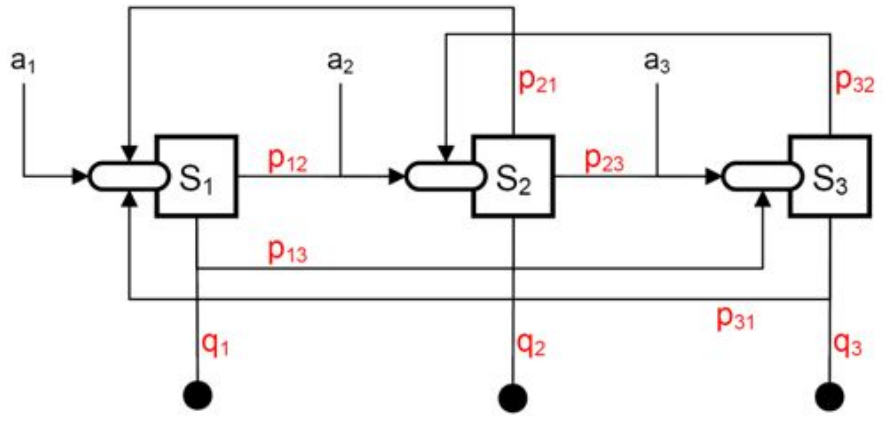
- **Jackson-Theorem**:

$$P(N_1 = n_1 \wedge N_2 = n_2 \wedge ... \wedge N_k = n_k) = \prod_{i=1}^{k} (1 - \rho_i) \rho_i^{n_i}$$

  wobei $\rho_i = \frac{\lambda_i}{\mu_i} < 1 \; \forall i$ der Auslastungsgrad des Servers i ist.

Die erstgenannte Eigenschaft ermöglicht es, aus den Parametern eines Jackson-Netzes die Ankunftsraten λ_i durch Lösen eines Linearen Gleichungssystems mit k Gleichungen und k Unbekannten zu ermitteln. Anschließend können die Formeln für $(\mathsf{M}, \mathsf{M}, \mathsf{s})$-Modelle auf jedes Wartesystem einzeln angewendet werden, um seine Leistungsgrößen zu bestimmen. Das Jackson-Theorem besagt, dass die mittlere Anzahl der Kunden im Wartesystem i unabhängig ist von der Anzahl irgendeines anderen Wartesystems im Netzwerk.

Ein Beispiel für ein Jackson-Netzwerk zeigt die nächste Abbildung.



$$\begin{aligned}
\lambda_1 &= a_1 + & & & p_{21}\lambda_2 &+ p_{31}\lambda_3 \\
\lambda_2 &= a_2 + p_{12}\lambda_1 + & & & p_{32}\lambda_3 \\
\lambda_3 &= a_3 + p_{13}\lambda_1 + p_{23}\lambda_2 & & &
\end{aligned}$$

Abbildung 12: Beispiel für ein Jackson-Netzwerk

Für die quantitative Analyse von Geschäftsprozessen kann die Warteschlangentheorie nur sehr begrenzt eingesetzt werden. Im Allgemeinen sind die Vorausset-

zungen zur Anwendung der Warteschlangenmodelle zu restriktiv oder unrealistisch. Beispielsweise ist die Länge einer Warteschlange in der Praxis nicht unbegrenzt. Auch gelten o.g. Ergebnisse wie z.B. das Jackson-Theorem nicht, wenn andere Verteilungen als die Exponentialverteilung verwendet werden. Jackson-Netzwerke können auch dann nicht verwendet werden, wenn Parallelität im Geschäftsprozess vorkommt.

Simulation

Simulation ist praktisch die einzige Methode zur quantitativen Analyse von Geschäftsprozessen, die für einfache wie komplexe Modelle relativ problemlos funktioniert. Sie ist immer dann eine Alternative, wenn Markov-Analyse oder Warteschlangentheorie nicht anwendbar oder wegen des Aufwandes nicht effizient einsetzbar sind.

Die Besonderheiten der Simulation werden im Kapitel 4 vorgestellt, so dass die weitere Beschreibung an dieser Stelle erfolgt.

In den nächsten beiden Kapiteln werden zwei Verfahren zur quantitativen Analyse von Geschäftsprozessen im Detail vorgestellt: die Kapazitätsplanung und die quantitative Simulation. Anschließend wird die Implementierung beider Verfahren in Java für das Modellierungswerkzeug ”WoPeD” beschrieben.

3 Kapazitätsplanung

Im Abschnitt 2.1.2 wurde bereits erwähnt, dass bei der Modellierung von Geschäftsprozessen zwei Aspekte zu berücksichtigen sind. Zum einen werden die Prozesselemente und ihre Anordnung spezifiziert und zum anderen werden die Ressourcen organisiert. Die Trennung macht deshalb Sinn, weil es möglich ist, verschiedene Ressourcenbelegungen auf eine Prozesspezifikation anzuwenden.

Es wird also bei der Geschäftsprozessmodellierung ein eigenes Ressourcenmodell entwickelt. Dabei werden zunächst die einzelnen Ressourcen (-objekte) nicht berücksichtigt, da es bei ihnen ständig Veränderungen gibt in ihrer Anzahl und Qualifikation. Kurzfristige und langfristige Schwankungen sind der Normalfall. Die kurzfristigen Schwankungen wegen z.B. Krankheitsausfällen, kurzfristigen vorübergehenden Kapazitätssteigerungen, Fluktuation, etc., werden im Geschäftsprozessmodell nicht berücksichtigt, da es auf ein längerfristiges Bestehen ausgerichtet ist.

Langfristige Schwankungen z.B. aufgrund saisonaler Unterschiede im Kapazitätsbedarf oder als Auswirkung einer neuen Personalentwicklungsstrategie werden durch Bildung eines Mittelwertes berücksichtigt. Bei der Kapazitätsplanung, wie sie hier vorgestellt wird[21], werden diese Mittelwerte auf Basis der quantitativen Spezifikation des Prozesses und des zugehörigen Ressourcenmodells berechnet. Damit wird sie für den Manager zum Werkzeug, einen weitestgehend optimalen Bestand an Ressourcen zu bestimmen.

Das von van der Aalst vorgestellte Ressourcenmodell verwendet eine Klassifikation der Anforderungen an die Qualifikation der Ressourcen. Diese Klassifikation erfolgt in zwei Dimensionen, nach organisatorischen Gesichtspunkten (Gruppen) und nach funktionalen Gesichtspunkten (Rollen). Damit wird für jede Ressource angegeben, zu welchen organisatorischen Einheiten (Stellen) sie gehört und welche Funktionen sie im Prozess aufgrund ihrer individuellen Fähigkeiten wahrnehmen kann.

Mit diesem Verfahren ist es möglich, auch sehr komplexe Ressourcenzuteilungen abzubilden. Auch die feste Bindung von Ressourcen an einzelne Aufgaben oder der komplett flexible Einsatz sind damit modellierbar. Nachfolgend ist die Ressourcenklassifikation für den Beispiel-Prozess abgebildet.

[21] Die hier vorgestellte Methodik bezieht sich auf die Ausführungen von [AaHe02, Abs. 3.1, 3.2, 4.5].

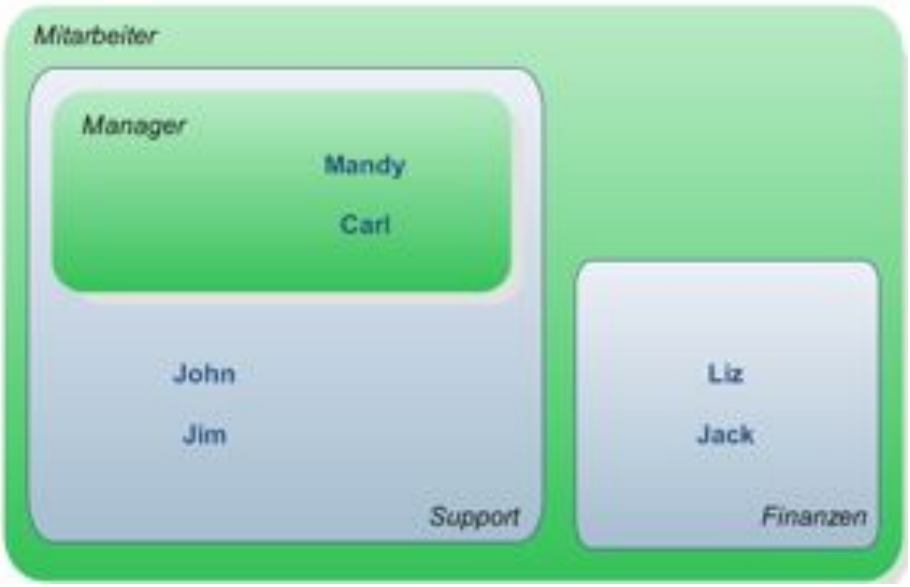


Abbildung 13: Beispiel für eine Ressourcenklassifikation

Die Zuordnung der Ressourcen zu den Aufgaben erfolgt durch Spezifikation eines Anforderungsprofils. Das heißt, jeder Aufgabe wird ein Paar bestehend aus einer Gruppenbezeichnung und einer Rollenbezeichnung zugeordnet. Soll zu einem Zeitpunkt für ein Work Item eine Ressource bestimmt werden, dann wird mithilfe des Anforderungsprofils aus der Menge der zu diesem Zeitpunkt ungebundenen Ressourcen eine passende ausgewählt.

Diese Form der Ressourcenmodellierung kann auch formalisiert werden, was im Folgenden geschieht:

Ein **Ressourcenmodell** $\mathbf{R_M}$ ist ein Tupel (G, R, O, κ) mit endlichen Mengen G, R und O.
Dabei bedeuten G eine nichtleere Menge von Organisationseinheiten (Gruppen) und R eine nichtleere Menge von Funktionsbeschreibungen (Rollen). Die Elemente von $G \cup R$ werden **Ressourcenklassen** genannt. O ist eine nichtleere Menge von Ressourcenobjekten. ϵ bezeichnet die **leere Ressourcenklasse**, die sowohl Gruppe als auch Rolle sein kann.
$\kappa : O \to \{C \in \mathbb{P}(G \cup R) \mid \exists g, r \in C \wedge g \neq r \Rightarrow g \in G \wedge r \in R\}$, ist die **Qualifizierungsfunktion**, die jedem Ressourcenobjekt eine nichtleere Menge von Ressourcenklassen zuordnet, und zwar so, dass mindestens eine Gruppe und mindestens eine Rolle enthalten sind.
Sei $b \in O$ ein Ressourcenobjekt. Dann werden weiterhin folgende zwei Abbildungen definiert:
Die Funktion $\gamma_\kappa : O \to \mathbb{P}(G), b \mapsto (\kappa(b) \cap G)$ liefert alle Gruppen, die b zugeordnet sind.
Die Funktion $\rho_\kappa : O \to \mathbb{P}(R), b \mapsto (\kappa(b) \cap R)$ liefert alle Rollen, die b zugeordnet sind.

Ein Ressourcenmodell kann auf mehrere verschiedene Prozessmodelle angewendet werden. Es beschreibt lediglich die Aufteilung der Ressourceninstanzen auf Gruppen und Rollen.

Die Spezifizierung der Ressourcenverteilung für einen konkreten Geschäftsprozess erfolgt durch die Zuweisung der Ressourcen:

Eine **Ressourcenzuweisung** $\mathbf{R_Z}$ ist ein Tupel $(\mathbf{N}, \mathbf{R_M}, \zeta)$.
Dabei sind $\mathbf{N} = (S, T, F)$ ein Workflow-Netz und $\mathbf{R_M} = (G, R, O, \kappa)$ ein Ressourcenmodell.
$\zeta : T \rightarrow ((G \times R) \cup \{(\epsilon, \epsilon)\})$ ist die **Zuweisungsfunktion**, die jeder Transition des Prozessmodells eine Gruppe und eine Rolle zuordnet. Transitionen, die nur den Kontrollfluss steuern und daher keinen Beitrag zur Performanz des Geschäftsprozesses leisten, erhalten das Paar (ϵ, ϵ).

Durch die Zuweisungsfunktion erhält jede Transition eine Art (Ressourcen-) Signatur, in dem Sinne, dass die Transition ein Anforderungsprofil an die zur Verfügung stehenden Ressourcen erhält.

Damit können nur Ressourcenobjekte die durch eine Transition symbolisierte Aufgabe bearbeiten, wenn sie (1) frei sind und (2) dem Anforderungsprofil der Transition entsprechen. Dieser Fall liegt dann vor, wenn für $t \in T$ und $b \in O$ gilt:

$(\pi_1(\zeta(t)), \pi_2(\zeta(t))) \in (\gamma_\kappa(b) \times \rho_\kappa(b))$	oder
$\pi_1(\zeta(t)) \in \kappa(b) \wedge \pi_2(\zeta(t)) \in \kappa(b)$	was zum gleichen Ergebnis führt.

Damit liegt eine komplette Beschreibung des Geschäftsprozesses vor. Zur quantitativen Analyse, insbesondere für die hier vorgestellte Kapazitätsplanung, sind aber weitere Angaben notwendig. Grundlage für die Bestimmung der benötigten Anzahl von Ressourcen[22] ist der aus diesen Angaben zu bestimmende Kapazitätsbedarf. Er ist offensichtlich abhängig von der Ankunftsrate neuer Fälle, den mittleren Bedienzeiten der einzelnen Aufgaben und dem Kontrollfluss. Besonders Schleifen und Synchronisationspunkte bei paralleler Durchführung von Aufgaben haben großen Einfluss auf die Effizienz des Prozesses.

Alternative Prozesspfade (XOR-Entscheidungen) werden aufgrund von Erfahrungswerten oder Messungen mit empirischen Häufigkeitsverteilungen ergänzt. Das heißt, bei einer Entweder-Oder-Entscheidung erhält jede Verzweigung eine relative Häufigkeit als Angabe, wie viele Fälle von 100% im Mittel diesem Pfad folgen.

[22] Genauer: des Mittelwertes der benötigten Anzahl. Vgl. die Ausführungen weiter oben.

Zur Berechnung der Anzahl benötigter Ressourcen wird weiterhin die zeitliche Länge der Betrachtungsperiode sowie eine mittlere Auslastung verwendet. Ohne die Angabe von Auslastungsgraden würde die Rechnung implizit von 100% Auslastung ausgehen, was unrealistisch ist. Statt jeder einzelnen Ressource einen Wert zuzuordnen, wird davon ausgegangen, dass alle Ressourcen einer Klasse im Mittel dieselbe Auslastung haben. Noch weiter läßt sich die Berechnung vereinfachen, wenn es zwischen den Klassen keine erheblichen Unterschiede gibt. Dann wird für alle Klassen ein- und derselbe Auslastungsgrad verwendet.

Ausgehend von der Verwendung von Workflow-Netzen zur Modellierung von Geschäftsprozessen muss das Petri-Netz-Modell um die genannten Größen erweitert werden. In Anlehnung an van der Aalst gilt folgende Notation:

- mittlere Bedienzeit einer Aufgabe: Anzahl Zeiteinheiten in der Mitte der Transition in grüner Farbe.
- Verzweigungswahrscheinlichkeiten: Angabe in Prozent entlang der Pfeile in roter Farbe.
- Anforderungsprofil an Ressourcen: Wenn eine Aufgabe ressourcengetriggert ist, wird die Angabe ergänzt um ein zweizeiliges Label mit der Angabe der Gruppe in kursiver Schrift in der ersten Zeile und der Angabe der Rolle in Normalschrift, grau unterlegt in der zweiten Zeile (

).

Das so erweiterte Petri-Netz für den Geschäftsprozess ”Beschwerdebearbeitung” wurde bereits in Abbildung 9 auf Seite 20 gezeigt.

Insgesamt ergibt sich damit:

Das Tupel $(\mathbf{R_Z}, V, \pi, \lambda, \vartheta, \eta)$ ist eine vollständige **Spezifikation für eine Kapazitätsplanung** mit

- der Ressourcenzuweisung $\mathbf{R_Z}$ für ein Workflow-Netz (S, T, F) und ein Ressourcenmodell (G, R, O, κ),
- der Menge der Verteilungsfunktionen V für jeden Knoten k des Netzes, außer der Senke, so dass gilt:

$$V = \{\nu_k \forall k \in (S \cup T) \setminus \{o\} \mid \nu_k : k\bullet \to (0..1], a \mapsto p_a\}$$
$$\Upsilon(k) \in \mathbb{A}_S \Rightarrow p_a = 1 \; \forall a \in k\bullet$$
$$\Upsilon(k) \in \mathbb{K} \setminus \mathbb{A}_S \Rightarrow \sum_{\forall a \in k\bullet} p_a = 1$$

- der Periode $\pi \in \mathbb{R}^+$,
- der mittleren Ankunftsrate von Fällen pro Periode $\lambda \in \mathbb{R}^+$,
- der Bedienzeitfunktion $\vartheta : T \to \mathbb{R}_0^+, t \mapsto \vartheta(t) = \mu_t$, die jeder Transition t eine mittlere Bedienzeit μ_t zuordnet, und
- der Auslastungsfunktion $\eta : (G \cup R) \to (0..1], c \mapsto \eta(c) = \upsilon_c$, welche für jede Ressourcenklasse c einen Auslastungsgrad υ_c angibt.

Als Kapazitätsplanung wird dann das Tupel $(\mathbf{R_Z}, V, \pi, \lambda, \vartheta, \eta, \chi)$ mit der Spezifikation $(\mathbf{R_Z}, V, \pi, \lambda, \vartheta, \eta)$ und dem Ergebnis χ bezeichnet. $\chi : (G \cup R) \to \mathbb{R}^+$ ist eine Funktion, die jeder Ressourcenklasse die Anzahl der benötigten Ressourcen zuordnet.

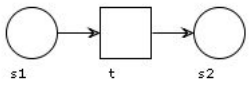
Die Werte der Funktion χ lassen sich aus den Angaben der Spezifikation berechnen. Gemäß der Methodik von [AaHe02] muss zunächst für jede Transition aus dem Prozessmodell die mittlere Anzahl ihrer Ausführung abgeleitet werden. Dieser Wert multipliziert mit der Bedienzeit ergibt den Zeitbedarf der Transition pro Periode für die durch λ angegebene Anzahl von Fällen.

Anschließend wird aus der Ressourcenzuweisung abgeleitet, wie hoch der Zeitbedarf pro Ressourcenklasse ist. Dazu werden die Werte für den Zeitbedarf der Transitionen addiert, die zur selben Ressourcenklasse gehören.

Zum Schluss ist der Zeitbedarf pro Ressourcenklasse durch den Auslastungsgrad und die Periodenlänge zu teilen. Diese Schritte werden nochmals formelmäßig dargestellt:

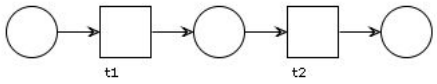
1. **Bestimmung der Ausführungshäufigkeit der Transitionen:**
   Die **Ausführungshäufigkeit** sei definiert als eine Funktion $\phi : (S \cup T) \to \mathbb{R}^+$. Für die Quelle i des Netzes gilt: $\phi(i) = \lambda$. Gemäß [AaHe02] gilt nun für die jeweiligen Kontrollstrukturen:

(a) *Basiselement*:



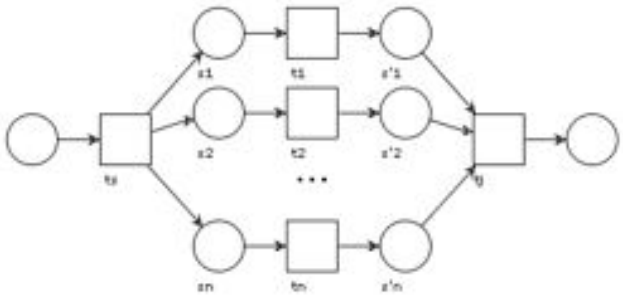
Es gilt: $s_1\bullet = \{t\} \wedge t\bullet = \{s_2\} \Rightarrow \phi(s_2) = \phi(t) = \phi(s_1)$.

(b) *Sequenz* :



Es gilt: $(t_1\bullet)\bullet = \{t_2\} \Rightarrow \phi(t_2) = \phi(t_1)$.

(c) *Parallelität* :

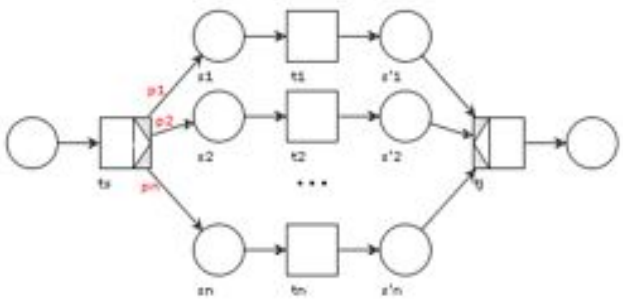


i. Split:
Es gilt: $t_s\bullet = \{s_1, s_2, ..., s_n\} \wedge s_i\bullet = \{t_i\}\ \forall i \in \{1, 2, ..., n\}$ mit $n > 1$ und $\Upsilon(t_s) \in \mathbb{A}_S \Rightarrow \phi(t_i) = \phi(s_i) = \phi(t_s)\ \forall i$.

ii. Join:
Es gilt: $\exists t_1, t_2, ..., t_n$ mit $n > 1$ und $\Upsilon(t_j) \in \mathbb{A}_J$, so dass gilt: $t_i\bullet = \{s_i^{'}\} \wedge s_i^{'}\bullet = \{t_j\} \Rightarrow \phi(t_j) = \phi(s_1^{'}) = \phi(s_2^{'}) = ... = \phi(s_n^{'}) = \phi(t_1) = \phi(t_2) = ... = \phi(t_n)$.

ANMERKUNG: Daraus folgt insbesondere für das gesamte AND-Konstrukt (bei einem syntaktisch korrekten Netz), dass $\phi(t_j) = \phi(t_s)$ gilt. Das bedeutet, dass für alle Fälle, die vom Prozess verarbeitet werden, genauso viele Synchronisierungen paralleler Abläufe stattfinden, wie Aufteilungen vorgenommen wurden. Damit bleibt also die semantische Korrektheit erhalten.

(d) *Alternative* :



i. Expliziter Split:

Es gilt: $\Upsilon(t_s) \in \{\texttt{t\_x}, \texttt{t}_{\texttt{XX}}, \texttt{t}_{\texttt{AX}}\} \wedge t_s\bullet = \{s_1, s_2, ..., s_n\}$ mit $n > 1$ und $s_i\bullet = \{t_i\}\ \forall i \in \{1, 2, ..., n\}$. Ferner gilt: $0 < (\nu_{t_s}(s_i) = p_i) < 1\ \forall i \wedge \sum_{\forall i} p_i = 1 \Rightarrow \phi(t_i) = \phi(s_i) = \phi(t_s) \cdot p_i\ \forall i$.

ii. Expliziter Join:

Es gilt: $\exists t_1, t_2, ..., t_n \wedge n > 1 : t_i\bullet = \{s_i'\} \wedge s_i'\bullet = \{t_j\}\ \forall i \in \{1, 2, ..., n\} \wedge \Upsilon(t_j) \in \{\texttt{t}_{\texttt{X\_}}, \texttt{t}_{\texttt{XX}}, \texttt{t}_{\texttt{XA}}\} \Rightarrow \phi(s_i') = \phi(t_i)\ \forall i \wedge \phi(t_j) = \sum_{\forall i} \phi(s_i')$.

ANMERKUNG: Aus den Berechnungsvorschriften folgt somit:

$$\phi(t_i) = \phi(t_s) \cdot p_i\ \forall i \text{ (Split)} \tag{1}$$

$$\phi(s_i') = \phi(t_i)\ \forall i \text{ (Basiselement)} \tag{2}$$

$$\phi(t_j) = \sum_{\forall i} \phi(s_i') \text{ (Join)} \tag{3}$$

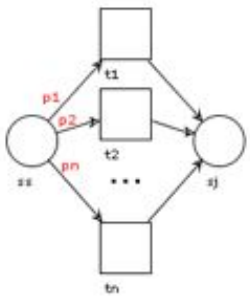
$$= \sum_{\forall i} \phi(t_i) \text{ (nach (2))} \tag{4}$$

$$= \sum_{\forall i} \phi(t_s) \cdot p_i \text{ (nach (1))} \tag{5}$$

$$= \phi(t_s) \cdot \sum_{\forall i} p_i \tag{6}$$

$$= \phi(t_s) \text{ (wegen } \sum_{\forall i} p_i = 1) \tag{7}$$

Insgesamt folgt also: $\phi(t_j) = \phi(t_s)$. Damit gilt sinngemäß das in der Anmerkung zur Parallelität Gesagte.



i. Impliziter Split:

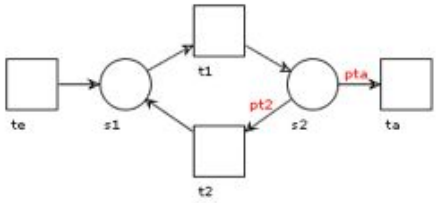
Es gilt: $s_s\bullet = \{t_1, t_2, ..., t_n\}$ mit $0 < (\nu_{s_s}(t_i) = p_i) < 1 \wedge \sum_{\forall i} p_i = 1 \Rightarrow \phi(t_i) = \phi(s_s) \cdot p_i\ \forall i$.

ii. Impliziter Join:

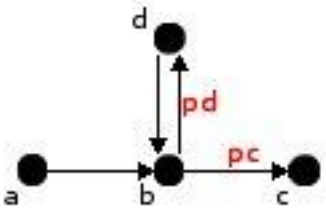
Es gilt: $\exists t_1, t_2, ..., t_n \wedge n > 1 : t_i\bullet = \{s_j\}\ \forall i \in \{1, 2, ..., n\} \Rightarrow \phi(s_j) = \sum_{\forall i} \phi(t_i)$.

ANMERKUNG: Mit $\phi(s_j) = \sum_{\forall i} \phi(t_i) = \sum_{\forall i} \phi(s_s) \cdot p_i = \phi(s_s) \cdot \sum_{\forall i} p_i = \phi(s_s) \cdot 1 = \phi(s_s)$ gilt damit dasselbe wie für die Anmerkung zum expliziten XOR.

(e) *Iteration*:



Wegen der Vielzahl der Möglichkeiten der Modellierung einer Iteration, nicht zuletzt auch wegen der Wahl zwischen expliziten und impliziten XOR-Splits, wird zunächst eine Abstraktion des Workflow-Netzes zu einem (allgemeinen) gerichteten Graphen mit Kantengewichten vorgenommen. Die Funktion der Ausführungshäufigkeiten ϕ unterscheidet ohnehin nicht zwischen Transitionen und Stellen.
Es wird also der Graph $\mathbf{G} = (E, K, \omega)$ mit der Gewichtsfunktion ω betrachtet, der sich aus dem Netz $\mathbf{N} = (S, T, F)$ wie folgt ergibt:
Die Menge der Knoten ist $E = S \cup T$,die Menge der Kanten $K = F$. Für die Kantengewichte gilt: $\omega : K \rightarrow (0..1], k \mapsto \omega(k) = \nu_{\pi_1(k)}(\pi_2(k))$.
Wenn man zusätzlich bedenkt, dass sich Teilnetze[23] gemäß der Betrachtungen zu Netztransformationen (siehe Anhang B) durch Vergröberung auf einen Knoten reduzieren lassen, ergibt sich als allgemeine Form einer Schleife der folgende Graph:



[23]Für diese Teilnetze kann eine gesonderte Betrachtung gemäß der hier vorgestellten Berechnungsvorschriften durchgeführt werden.

Es gilt:

$$\begin{aligned}\phi(b) &= \phi(a) + \phi(d)\\ \phi(d) &= \omega(b,d) \cdot \phi(b)\\ \phi(c) &= \omega(b,c) \cdot \phi(b)\end{aligned}$$

Daraus folgt:

$$\begin{aligned}\phi(b) = \phi(a) + \omega(b,d) \cdot \phi(b) &\Leftrightarrow (1 - \omega(b,d)) \cdot \phi(b) = \phi(a)\\ &\Leftrightarrow \phi(b) = \frac{1}{1-\omega(b,d)} \cdot \phi(a)\\ &\Rightarrow \phi(d) = \frac{\omega(b,d)}{1-\omega(b,d)} \cdot \phi(a)\\ &\Rightarrow \phi(c) = \frac{\omega(b,c)}{1-\omega(b,d)} \cdot \phi(a)\end{aligned}$$

ANMERKUNG: Wenn speziell gilt $\omega(b,c)+\omega(b,d) = 1 \Leftrightarrow \omega(b,c) = 1 - \omega(b,d)$, dann folgt $\phi(c) = \phi(a)$. Analog zu den Betrachtungen der Parallelität und Alterative bleibt somit auch bei der Iteration die semantische Korrektheit erhalten.

2. **Bestimmung des Zeitbedarfs der Transitionen:**
   Der Zeitbedarf θ einer Transition *t* ergibt sich durch:

$$\theta(t) = \phi(t) \cdot \vartheta(t) = \phi(t) \cdot \mu_t$$

3. **Bestimmung des Zeitbedarfs pro Ressourcenklasse:**
   Der Zeitbedarf Θ pro Ressourcenklasse $c \in G \cup R$ bestimmt sich wie folgt:

$$\Theta(c) = \sum \theta(t) \; \forall t : (\pi_1(\zeta(t)) = c \vee \pi_2(\zeta(t)) = c)$$

4. **Bestimmung von χ:**
   Für die Werte von χ gilt:

$$\chi = \frac{\Theta(c)}{\pi} \cdot (\eta(c))^{-1} = \frac{\Theta(c)}{\pi} \cdot \upsilon_c^{-1}$$

Der größte Teil der Kapazitätsplanung ist recht problemlos. Die besondere Schwierigkeit liegt in der Bestimmung der Ausführungshäufigkeiten der Aufgaben. Vor allem mehrfach verschachtelte Schleifen und "Überkreuzungen" von Pfaden machen es mitunter notwendig, von einem linearen Durchlauf durch das Netz abzuweichen. Detailliertere Angaben dazu werden im Abschnitt 5.1 gemacht.

Voraussetzung für die erfolgreiche Durchführung einer Kapazitätsanalyse ist ein

semantisch und strukturell korrektes Netz. Das bedeutet, dass vor Beginn der Berechnung die Soundness des Netzes festgestellt werden muss. In einem Netz, das nicht sound ist, können tote Transitionen auftauchen, was bedeutet, dass an der Senke nicht mehr 100% der Fälle ankommen oder die Senke wird erst gar nicht erreicht. Ein weiterer Fehler ergäbe sich dadurch, das bei Erreichen der Senke doch noch Referenzen auf einen Fall im Netz existieren, was bei der quantitativen Analyse zu Häufigkeitswerten von mehr als 100% führen würde.

Zum Abschluss dieses Kapitels sollen die Ergebnisse der Kapazitätsplanung für das Beispiel-Netz vorgestellt werden:

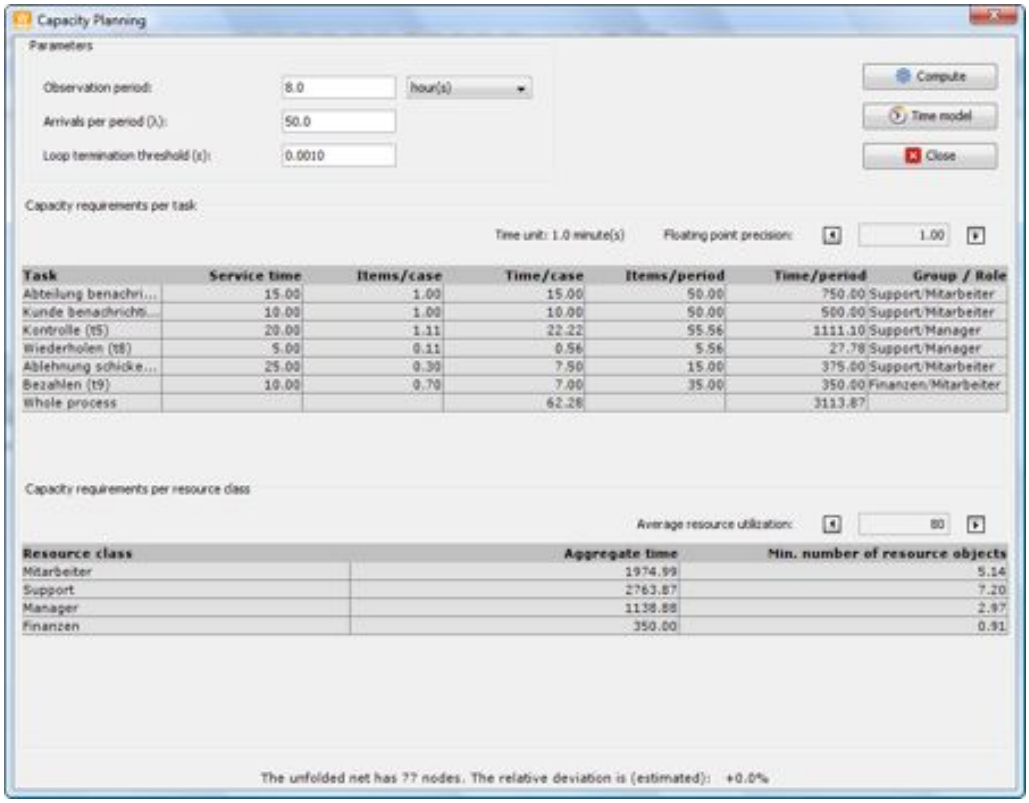


Abbildung 14: Kapazitätsplanung für das Beispiel

4 Ereignisdiskrete Simulation

Mit Simulation wird das Analyseverfahren bezeichnet, bei dem ein reales System, z.B. ein Geschäftsprozess, durch ein Modell ersetzt und direkt "ausgeführt" wird. A. M. LAW schreibt dazu ([Law07, S. 1]):

> "In a *simulation* we use a computer to evaluate a model *numerically*, and data are gathered in order to *estimate* the desired true characteristics of the model."

Die beiden hervorgehobenen Aspekte sind dabei charakteristisch: (1) Simulation ist kein analytisches Verfahren und liefert daher keine exakten Resultate. Vielmehr handelt es sich dabei um (2) Schätzungen, deren Aussagekraft durch statistische Verfahren gesichert werden muss. Dazu zählt insbesondere, dass im Rahmen einer Simulationsstudie mehrere Simulationsläufe mit denselben Parametern durchgeführt werden müssen. Aus den Ergebnissen werden dann Mittelwerte und Streuungen berechnet und *Konfidenzintervalle* angegeben.

Außer zur Kostensenkung bzw. Risikominimierung dient die Simulation als eine Art Sensitivitätsanalyse. Das heißt, es kann gezielt die Auswirkung der Änderung verschiedener Parameter auf das Systemverhalten untersucht werden. Das Optimum läßt sich auf diese Weise nicht finden, man kann sich ihm aber in ausreichendem Maße annähern. Simulation ist daher ein *passives Verfahren*, da es nicht selbst nach einer optimalen Lösung sucht - im Gegensatz zu analytischen Verfahren.

Der entscheidende Vorteil der Simulation gegenüber anderen Verfahren wie Markov-Analyse oder Warteschlangentheorie ist ihre *Flexibilität*. Sie läßt sich praktisch immer anwenden. Zum Beispiel unterliegt sie nicht der Beschränkung, dass nur exponentialverteilte Prozesse betrachtet werden können.

Um die Suche nach geeigneten Parameterwerten für das erste Experiment einzuschränken, können im Vorfeld der Simulation durchaus analytische Methoden eingesetzt werden, ggf. mit vereinfachenden Annahmen, um gute Näherungslösungen als Simulationsinput zu erhalten.

Im Simulationskontext spielen einige Begriffe eine zentrale Rolle. Sie sollen hier kurz definiert werden (vgl. [Wins04, S. 1146-1147]):

Ein **System** ist eine Sammlung von Objekten (oder Entitäten), die agieren und interagieren, um ein gemeinsames Ziel zu erreichen. Entitäten besitzen **Attribute**, d.h. Eigenschaften. Diese Eigenschaften können verschiedene Werte annehmen.
Zustandsvariablen sind Variablen, deren Werte sich mit der Zeit ändern und die charakterisierend für das System sind. Sie entsprechen den Attributen des Systems.
In einem (zeit-) **diskreten System** ändern sich die Zustandsvariablen zu diskreten Zeitpunkten und damit in abzählbarer Häufigkeit. Bei einem (zeit-) **kontinuierlichen Sytem** finden die Zustandsänderungen kontinuierlich in der Zeit statt.
Ein **statisches Simulationsmodell** ist eine Abbildung eines Systems, genauer seines Zustands, zu einem bestimmten Zeitpunkt. Eine **dynamische Simulation** ist die Abbildung der Änderung des Systemzustands über einen Zeitraum.
Ein **stochastisches Simulationsmodell** beinhaltet mindestens eine Zufallsvariable. Das Gegenteil ist ein **deterministisches Simulationsmodell**.

Simulationen können also nach drei Dimensionen klassifiziert werden: nach der zeitlichen Granularität, nach der Betrachtung eines Zeitpunktes oder Zeitintervalls und nach der Verwendung von Zufallsvariablen. Im Kontext der Geschäftsprozessmodellierung finden diskrete dynamische stochstische Simulationen Anwendung. Da die Zustandsänderungen durch Ereignisse, z.B. Ankunft eines Kunden am Server X, ausgelöst werden bzw. Änderungen des Systemzustands wiederum Ereignisse auslösen, nennt man diese Simulationsart **Ereignisdiskrete Simulation**.

Wie aus den bisherigen Ausführungen zu erkennen ist, ist *Zeit* eine wichtige Eigenschaft im Zusammenhang mit Simulation. Zu unterscheiden sind insgesamt drei Zeitbegriffe:

1. **Echtzeit**: Das ist die Zeit, in der der reale Prozess stattfindet. Allgemein kann es sich um Tage, Wochen oder Monate genauso wie um Sekunden oder Millisekunden handeln. Das ist of auch ein Grund, eine Simulationsstudie durchzuführen.

2. **Simulationszeit**: Das ist die Zeit, die der simulierte Prozess benötigt. Sie entspricht der Ausführungszeit einer Simulationsstudie auf dem Computer. Bei einer ereignisgesteuerten diskreten Simulation existiert i.d.R. kein Zusammenhang zwischen Simulationszeit und Echtzeit. Sie ist abhängig von der Komplexität der Berechnungen und der Leistungsfähigkeit des Rechners.

3. **Modellzeit**: Die Modellzeit ist die während des Simulationslaufes veränderte Zeit, die *simulierte Zeit*. Sie wird in den Einheiten der Echtzeit angegeben, bewegt sich aber im Rahmen der Simulationszeit. Sie wird bei einer ereignisdiskreten Simulation während eines Simulationslaufes explizit auf den Zeitpunkt des nächsten Ereignisses *gesetzt*.

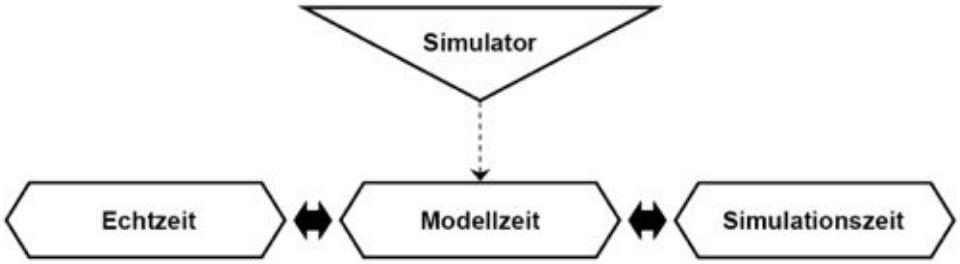
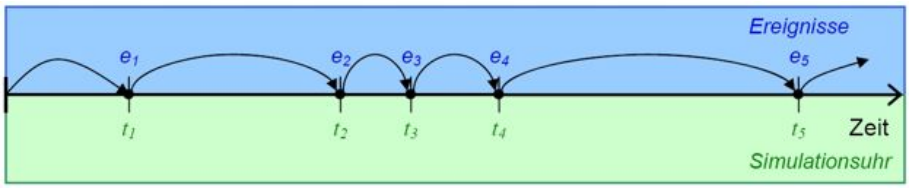


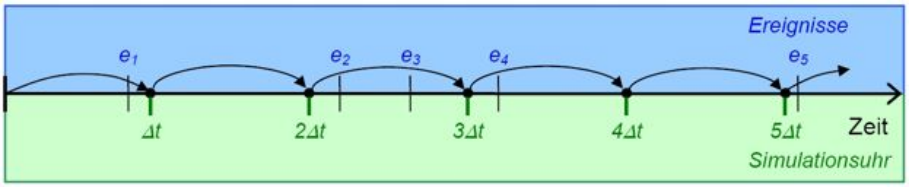
Abbildung 15: Zeitbegriffe in der Simulation

Die jeweils aktuelle Modellzeit wird von der *Simulationsuhr* angezeigt. Das Fortschreiten der Zeit kann grundsätzlich nach zwei Strategien erfolgen (vgl. [Law07, Abs. 1.3.1, Anh. 1A]):

1. **Next-event time advance**:
   Nach vollständiger Abarbeitung des Reaktionsalgorithmus' für das eben eingetretene Ereignis wird die Simulationsuhr auf den Zeitpunkt des als nächstes eintretenden Ereignisses gesetzt. Die zeitabhängigen Attribute aller betroffenen Entitäten des Systems sind dann zu aktualisieren.



2. **Fixed-increment time advance**:
   Die Simulationsuhr wird nach Beenden der Ereignisroutine für das letzte festgestellte Ereignis zum Zeitpunkt t um einen konstanten Wert Δt erhöht. Danach wird festgestellt, ob in diesem Zeitintervall Ereignisse stattgefunden haben. Diese werden dann zum Zeitpunkt $t + \Delta t$ berücksichtigt.



Bei der zweiten Methode werden die Ereignisse nicht zum Zeitpunkt ihres Eintretens verarbeitet, was zu Fehlern bei der Berechnung statistischer Werte führen

kann und es besteht die Notwendigkeit, eine Strategie zu entwickeln, wie bei Feststellung mehrerer Ereignisse im Zeitintervall Δt vorzugehen ist. Wegen dieser Nachteile eignet sich der "Fixed-increment"-Ansatz nur bei Systemen, bei denen Ereignisse sehr regelmäßig in festen Zeitabständen vorkommen. Für die Analyse von Geschäftsprozessen wird daher der "Next-event"-Ansatz gewählt.

Ein allgemeingültiges Ablaufmodell für einen (top-level) Simulationsalgorithmus sieht folgendermaßen aus (vgl. [Law07, S. 9-27], [Fish01, Kap. 2]):

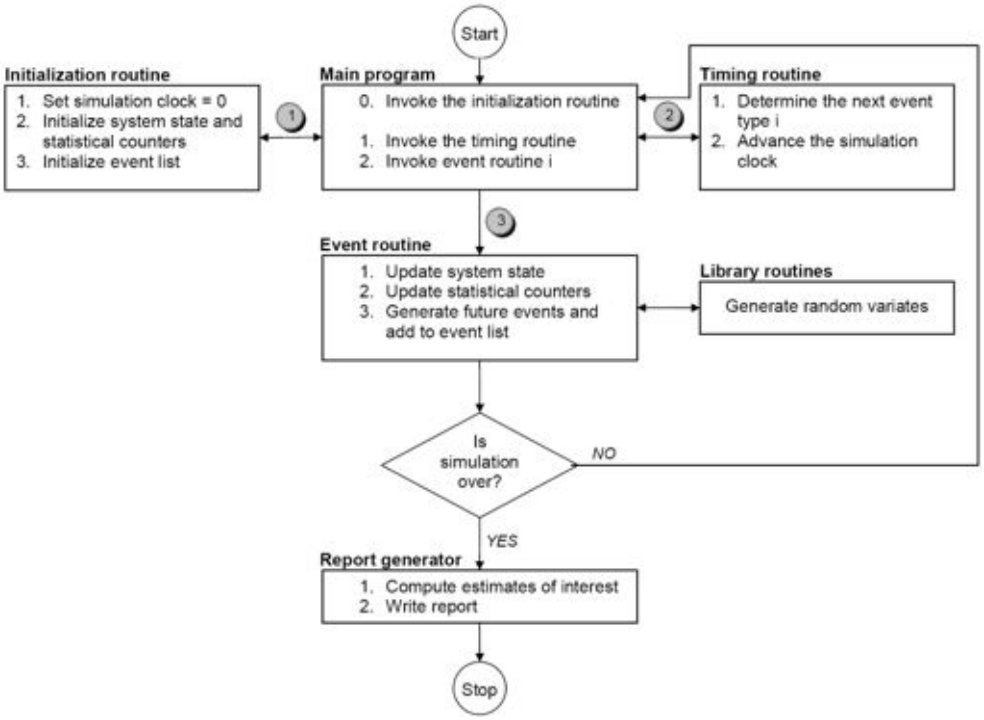


Abbildung 16: Allgemeiner Simulationsalgorithmus

Als erstes wird der Simulator initialisiert. Dazu wird die Simulationsuhr auf Null gestellt. Die Systemvariablen werden mit ihren Startwerten belegt. In welchem Zustand das System gestartet wird, ist allerdings nicht so wichtig. Bei jeder Simulation gibt es eine Einpendelphase, nach der sich das System unabhängig vom Startzustand verhält. Außerdem gibt es statistische Zählervariablen, um die Performance des Modells zu messen. Sie sind auf Null zu setzen.

Der Simulator verwaltet weiterhin die Ereignisliste. Hier werden alle Ereignisse gespeichert, die noch nicht eingetreten sind. Im Anfangszustand enthält sie nur das Ankunftsereignis für den ersten Fall.

Nach der Initialisierung startet die eigentliche Simulation. Dazu wird in einer Endlosschleife geprüft, ob eine vorher definierte Abbruchbedingung bereits erfüllt ist. Wenn nicht, wird aus der Ereignisliste das zeitlich nächste Ereignis entnommen, die Simulationsuhr auf die Zeit dieses Ereignisses gestellt und die Ereignisbehandlungsroutine gestartet.

Das Eintreten eines Ereignisses bedeutet, dass sich der Zustand des Systems ändert. Daher müssen durch die Ereignisbehandlungsroutine die Zustandsvariablen entsprechend dem eingetretenen Ereignis geändert werden. Anschließend werden die Statistikzähler aktualisiert und ein neues zukünftiges Ereignis generiert und in der Ereignisliste gespeichert. Dazu müssen Zeitwerte, wie Ankunftszeiten oder Bedienzeiten, aus den Parametern der Wahrscheinlichkeitsverteilungen berechnet werden. Hierbei werden Zufallszahlen mit der entsprechenden Verteilung verwendet.

Nach Abarbeiten eines Ereignisses wird wiederum geprüft, ob die Simulation zu Ende ist. Die Abbruchbedingung kann sehr individuell formuliert sein. Übliche Kriterien sind: die Anzahl bedienter Kunden erreicht eine Obergrenze oder die Zeit für einen Simulationslauf ist abgelaufen.

Nach dem Beenden der Simulation sind die Statistikzähler auszuwerten und aufzubereiten und die Ergebnisse entweder visuell darzustellen oder in einer Datei zu speichern. Diese Aufgaben übernimmt der Berichtgenerator. Mit ihm endet die Simulation.

Es ist aus Gründen der Nachvollziehbarkeit ratsam, während der Simulation ein Protokoll zu führen, das in einer Datei gespeichert werden kann. Das ist vor allem deswegen wichtig, weil stochastische Simulationen nicht reproduziert werden können, wie man es von Experimenten kennt. Bei jedem neuen Simulationslauf entstehen neue Ergebnisse, die mitunter sehr verschieden sein können. Es wird deshalb nochmal darauf hingewiesen, dass die Resultate einer Vielzahl von Simulationsläufen (bei gleichen Parametern) zu mitteln sind, um verlässliche Informationen über die Effizienz des Systems zu erhalten.

Handelt es sich bei dem zu simulierenden System um ein Wartesystem, dann gibt es grundsätzlich drei Ereignisse, die eintreten können:

1. *Ankunftsereignis*: Ein Fall kommt am Server an. Je nachdem, ob der Server bereits beschäftigt bzw. Ressourcen zur Bearbeitung des Falles frei sind oder nicht, kann der Fall direkt bearbeitet werden oder wird zunächst in die Warteschlange des Servers aufgenommen.

2. *Startereignis*: Wenn der Server frei ist bzw. freie Ressourcen zur Verfügung stehen, kann der nächste ankommende Fall oder gemäß der Warteschlangendisziplin der nächste Fall aus der Warteschlange übernommen werden und der Service beginnt.

3. *Endeereignis*: Nach Ablauf der (simulierten) Bedienzeit stoppt der Service und der Fall gilt als vollständig abgearbeitet.

Zwischen diesen Ereignissen bestehen folgende Beziehungen:

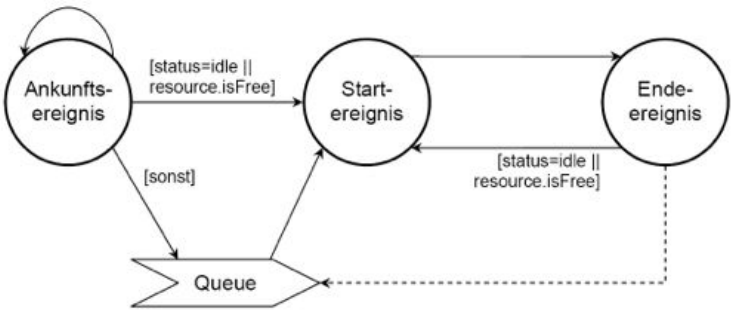


Abbildung 17: Beziehungen zwischen Ereignissen

Die Ereignisbehandlungsroutine eines Ankunftsereignisses erzeugt zunächst ein neues Ankunftsereignis. Dazu wird seine Zwischenankunftszeit als Pseudozufallszahl gemäß der Verteilung des stochastischen Prozesses mit der mittleren Ankunftsrate als Parameter erzeugt und zur Ankunftszeit des zuletzt erzeugten Ankunftsereignisses addiert.

Kann der neu angekommene Fall sofort bearbeitet werden, wird zudem ein neues Startereignis erzeugt, das denselben Zeitstempel trägt wie das Ankunftsereignis. Anderenfalls wird der Fall in der Warteschlange abgelegt.

Das Startereignis bestimmt den Zeitpunkt des Bedienungsendes für den aktuellen Fall. Dazu wird ebenfalls eine Pseudozufallszahl generiert mit der mittleren Bedienzeit als Parameter. Damit erzeugt es das Endeereignis für diesen Fall.

Die Ereignisbehandlungsroutine des Endeereignisses setzt eine gebundene Ressource frei und prüft, ob Fälle in der Warteschlange anstehen. Ist dies der Fall, wird das nächste Startereignis erzeugt.

Es werden implizit zwei Voraussetzungen gemacht: (1) Jede Ressource kann zu einer Zeit höchstens einen Fall bearbeiten und (2) die Warteschlange jedes Servers ist unbeschränkt.

Bei der Ausführung der Ereignisbehandlungsroutinen werden außerdem statistische Zählervariablen aktualisiert. In ihnen werden die aktuellen Werte zur Berechnung der Effizienzmaße gespeichert. Übliche zeitbezogene Messgrößen, wie sie bereits im Abschnitt 2.2.2 auf Seite 22 aufgeführt wurden, lassen sich wie folgt bestimmen[24]:

- mittlere Ankunftsrate:
$$\bar{A}(s,t) := \frac{A(s,t)}{t-s}$$
- durchschnittlicher Durchsatz:
$$\bar{N}(s,t) := \frac{N(s,t)}{t-s}$$

[24] Vgl. [Fish01, S. 10-13, 72-78]. Anmerkung: Diese Größen beziehen sich auf die Performance eines einzelnen Servers.

- durchschnittliche Warteschlangenlänge:

$$\bar{Q}(s,t) := \frac{1}{t-s}\int_s^t Q(u)\,du$$

- mittlere Serverauslastung: (entspricht durchschnittlicher Anzahl zu einem Zeitpunkt beschäftigter Ressourcen)

$$\bar{B}(s,t) := \frac{1}{t-s}\int_s^t B(u)\,du$$

- mittlere Wartezeit:

$$\bar{W}(s,t) := \frac{1}{N(s,t)}\sum_{i=N(s)+1}^{N(t)} W_i$$

Die Zählervariablen, die zur Berechnung verwendet werden, sind:

s	:=	Zeitpunkt des Endes der Einschwingphase
t	:=	beliebiger Zeitpunkt während der Simulation; es gilt: $t > s \geq 0$
$A(s,t)$	:=	Anzahl der Ankünfte im Zeitintervall $(s,t]$
$N(s,t)$	:=	Anzahl abgeschlossener Fälle im Intervall $(s,t],\ N(t) := N(0,t)$
$Q(t)$	:=	Länge der Warteschlange zum Zeitpunkt t
$B(t)$	:=	Anzahl beschäftigter Ressourcen zum Zeitpunkt t
W_i	:=	Wartezeit des (abgeschlossenen) Falles i

Zu den in der Warteschlangentheorie verwendeten Symbolen besteht ein Zusammenhang: L_q entspricht der mittleren Warteschlangenlänge $\bar{Q}$, W_q der mittleren Wartezeit $\bar{W}$ und L_s der mittleren Serverauslastung $\bar{B}$. λ bezeichnet hier wie da die mittlere Ankunftsrate und entspricht $\bar{A}$. Nach dem *Gesetz von Little* ließe sich $\bar{A}$ auch durch den Quotienten $\bar{Q}/\bar{W}$ berechnen. Damit kann bei Bestimmung der Durchlaufzeit W mittels $W = W_q + W_s = \bar{W} + \frac{1}{\mu}$ über die Formel $L = \lambda W$ die Gesamtzahl von Fällen im Server (wartend oder bedient) L berechnet werden. Außerdem ergibt sich L nach $L = L_q + L_s$.

Wird das Prädikat $I_A(x)$ wie folgt definiert, lassen sich außerdem die nachfolgend angegebenen abgeleiteten Effizienzmaße berechnen:

$$I_A(x) := \begin{cases} 1 & \text{wenn } x \in A \\ 0 & \text{sonst} \end{cases}$$

- Zeitanteil, dass die Warteschlangenlänge größer als q ist[25]:

$$\bar{P}_Q(q,s,t) := \frac{1}{t-s}\int_s^t I_{\{q'|q'>q\}}(Q(u))\,du \;\forall q \in \{0,1,...,q_{max}\}$$

- Zeitanteil, dass mehr als b Ressourcen parallel beschäftigt sind[26]:

$$\bar{P}_B(b,s,t) := \frac{1}{t-s}\int_s^t I_{\{b|b\in\{0,1,...,m\}\}}(B(u))\,du \;\forall b \in \{0,1,...,m\}$$

- Anteil der Fälle, die mehr als w Zeiteinheiten warten:

$$\bar{P}_W(w,s,t) := \frac{1}{N(s,t)}\sum_{i=N(s)+1}^{N(t)} I_{(w,\infty)}(W_i)$$

Diese Anteile können als Wahrscheinlichkeiten interpretiert werden. Hier wird der Bezug zu Markov-Ketten bzw. der Warteschlangentheorie deutlich. Dort werden Wahrscheinlichkeiten für den stabilen Zustand des Systems durch Grenzwertbetrachtungen berechnet. Bei der Simulation werden Approximationen solcher Werte bestimmt.

Wie in den angegebenen Formeln zu erkennen ist, gibt es zwei unterschiedliche Arten von Messgrößen: diskrete und kontinuierliche; je nachdem, ob sie sich auf einen Zeitraum oder eine Anzahl von Fällen beziehen. Dementsprechend unterscheidet sich ihre Berechnungsweise. So werden die Wartezeiten während der Simulation aufaddiert und am Ende durch die Anzahl bearbeiteter Fälle geteilt.

Für die Berechnung der zeitkontinuierlichen Größen wird zunächst festgestellt, dass ihre Veränderung sprunghaft zu bestimmten Zeitpunkten stattfindet, zwischen denen die Zählerwerte konstant sind. Damit kann mittels rekursiver Formeln zu jedem Zeitpunkt, an dem eine Änderung stattfindet, der aktuelle Wert berechnet werden[27].

Die Ereignisbehandlungsroutinen spielen dabei die folgenden Rollen:

- Die aktuelle Länge der Warteschlange wird durch ein Ankunftsereignis inkrementiert, wenn gerade keine Kapazitäten zur Bearbeitung eines weiteren Falles existieren. Sie wird dekrementiert durch ein Startereignis und ggf. durch ein Endeereignis, wenn die Warteschlange nicht leer ist und Kapazitäten existieren.

[25] Mit q_{max} wird die am Ende der Simulation festgestellte maximale Länge der Warteschlange bezeichnet.

[26] Mit m wird die für diesen Server maximal zur Verfügung stehende Anzahl von Ressourcen bezeichnet.

[27] Vgl. [Fish01, S. 75-77]. Zu Einzelheiten der Umsetzung siehe Abschnitt (5.2).

- Die aktuelle Serverauslastung (= Anzahl der parallel an diesem Server beschäftigten Ressourcen) wird durch ein Startereignis inkrementiert und durch ein Endeereignis dekrementiert.
- Die kumulierte Wartezeit wird durch ein Startereignis erhöht, indem aus dem Ankuftszeitpunkt und dem Startzeitpunkt des betroffenen Falles seine Wartezeit berechnet und zur letzten kumlierten Wartezeit des Servers addiert wird.

Bei der Simulation von Geschäftsprozessen müssen einige ergänzende Betrachtungen gemacht werden. Ein Prozess wird durch ein Netz von Servern dargestellt, die einen bearbeiteten Fall an ihren Nachfolger weitergeben. Dabei sind zwei Dinge zu beachten:

1. Bei einer Entweder-Oder-Entscheidung muss der aktuelle Server aus der Menge seiner Nachfolger entsprechend der im Prozessmodell unterlegten Wahrscheinlichkeitsverteilung einen auswählen, an den der aktuelle Fall weitergegeben wird.
2. Die Ankunfts- und Endeereignisse sind relativ, d.h. auf einen konkreten Server bezogen, zu verstehen. Es kommen als besondere Ereignisse das Ankunftsereignis und das Endeereignis für den Gesamtprozess hinzu[28].

Außerdem werden Ressourcen nicht anonym behandelt und jedem Server fest zugeordnet. Es existiert ein Pool von Ressourcen, aus denen zu gegebener Zeit eine passende ausgewählt wird, sofern vorhanden. Damit existiert auch die Möglichkeit, Performance-Messungen für die einzelnen Ressourcenobjekte durchzuführen. Das bezieht sich dann im Wesentlichen auf die Bestimmung des Auslastungsgrades.

Damit wird die theoretische Behandlung des Themas ”Quantitative Analyse von Geschäftsprozessen” geschlossen. Im nächsten Kapitel wird die Umsetzung der Theorie in ablauffähige Software unter Verwendung des Modellierungswerkzeugs ”WoPeD” beschrieben. Ein Screenshot einer Simulationsstudie für den Beispielprozess ist in der Abbildung auf der nächsten Seite zu sehen.

[28] Zum Beziehungsgeflecht dieser und weiterer Ereignisse siehe die Betrachtungen im Abschnitt 5.2.

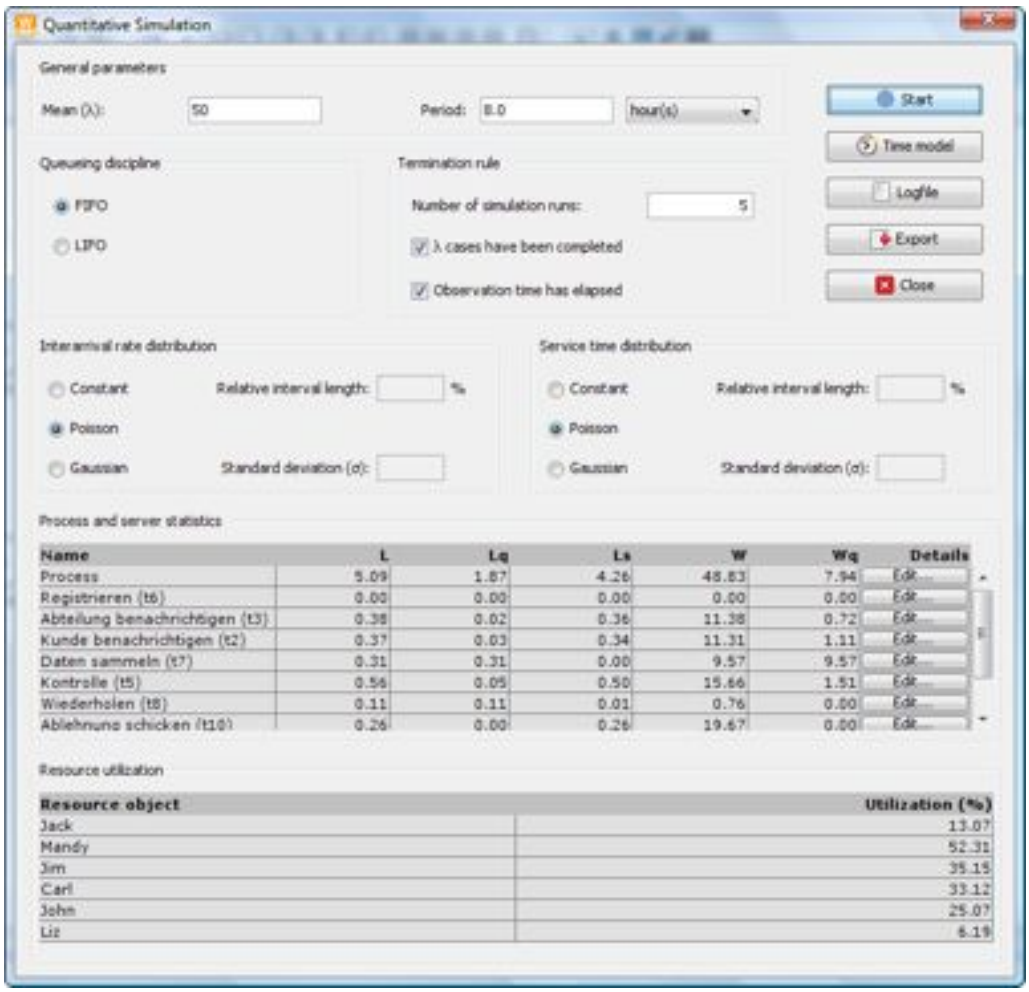


Abbildung 18: Simulation für das Beispiel

5 Implementierung in Java

Gegenstand dieses Kapitels ist die Darstellung, wie die theoretischen Ausführungen der vorhergehenden Kapitel in reale Software umgesetzt werden. Kapazitätsanalyse und quantitative Simulation sollen so, wie dort ausgeführt, implementiert werden. Beide Verfahren werden eingebunden in den ”Workflow Petri Net Designer” (WoPeD), ein open-source Modellierungstool, das an der Berufsakademie Karlsruhe entwickelt wird (vgl. [Land03], [FrLa03]).

Mit WoPeD können Geschäftsprozessmodelle als Workflow-Netze modelliert werden. Diese Modelle lassen sich mit dem Werkzeug validieren - durch eine integrierte Token-Game-Simulation - und verifizieren - mittels einer qualitativen Analyse (vgl. [Frey05],[Eckl06]). Die Workflow-Netze werden in einem standardisierten Format, der Petri Net Markup Language (PNML) gespeichert. PNML baut auf XML auf und läßt sich daher leicht erweitern, um die zusätzlichen Daten für eine quantitative Analyse aufzunehmen.

Die Software-Architektur von WoPeD basiert auf dem Model-View-Controller (MVC) -Paradigma. Das bedeutet, dass die Datenhaltung - das Model - von der graphischen Repräsentation - dem View - getrennt ist und die Beziehung zwischen beiden durch den Controller hergestellt wird. Dadurch können die Model-Ebene und die View-Ebene unabhängig voneinander in verschiedenen Versionen geändert werden. Auf der Controller-Ebene werden die notwendigen Anpassungen gekapselt. Damit ist es möglich, stets einen konsistenten Zustand nach Änderungen zu erreichen.

Daher lassen sich die beiden quantitativen Verfahren reibungslos in das Werkzeug integrieren. Das Top-Level-Paket trägt den Namen *WoPeD-QuantAnalysis*. Der Namensraum für das Projekt lautet `org.woped`. Die quantitative Analyse stützt sich auf die gesamte vorhandene Funktionalität ab. Die grobe Softwarearchitektur des erweiterten WoPeD sieht so aus:

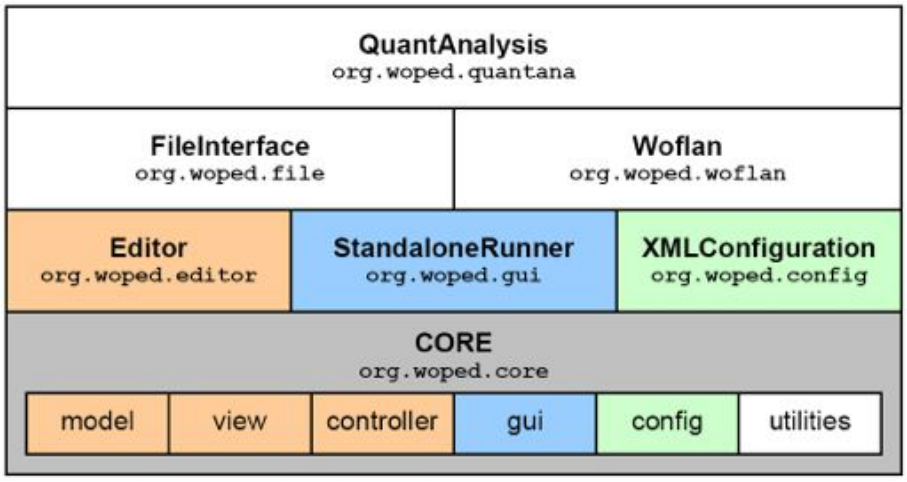


Abbildung 19: Software-Architektur von WoPeD

Das *CORE*-Paket enthält die Basis-Funktionalität, die von allen darüber liegenden Komponenten benötigt wird. Der Editor verwaltet die graphische Darstellung des Petri-Netzes, und zwar Model-, View- und Controller-Ebene. *XMLConfiguration* kümmert sich um die Verwaltung der PNML-Dateien. Im *StandaloneRunner* findet sich der Großteil der GUI und die Klasse `RunWoPeD` mit der `main()`-Methode wieder. *FileInterface* für das Speichern und Laden von Netzen und *Woflan* für die qualitative Analyse bauen darauf auf. Auf der obersten Ebene befindet sich *QuantAnalysis* mit den Unterpaketen für die quantitative Analyse.

Die Klassen zur qualitativen Analyse werden benötigt, um beim Start der Kapazitätsplanung oder der Simulation das zu analysierende Netz auf Korrektheit und Soundness zu überprüfen. Endet der Test negativ, wird die quantitative Analyse nicht ausgeführt und eine Fehlermeldung ausgegeben. Dabei wird vom Satz über die Soundness auf Seite 16 Gebrauch gemacht.

Das Paket `org.woped.quantana` besteht aus weiteren Unterpaketen:

`org.woped.quantana`	
`graph`	Schnittstelle zum Petri-Netz-Modell
`gui`	Sammlung von Dialogen zur Datenein- und Ausgabe
`model`	Datenmodelle für Dialoge
`resourcealloc`	Verwaltung der Ressourcen
`simulation`	Simulationsklassen
`utilities`	Hilfsklassen

Tabelle 3: Paketstruktur von ”QuantAnalysis”

Das Paket `graph` enthält die Klasse `WorkflowNetGraph`. Der Graph kapselt das Workflow-Netz in einer eigenen Datenstruktur für die quantitative Analyse. Dabei handelt es sich um eine Adjazenzliste aus `Node`-Objekten, die jeweils in einer `ArrayList` die Pfeile, implementiert durch Objekte der Klasse `Arc`, zu allen unmittelbaren Nachfolgern speichern. In den Knotenobjekten sind die Identifikationsnummer aus dem Petri-Netz-Modell und der Typ des Knotens (Stelle, einfache Transition, AND-Split, ...) hinterlegt. Die `Arc`-Objekte speichern außerdem die jeweiligen Verzweigungswahrscheinlichkeiten.

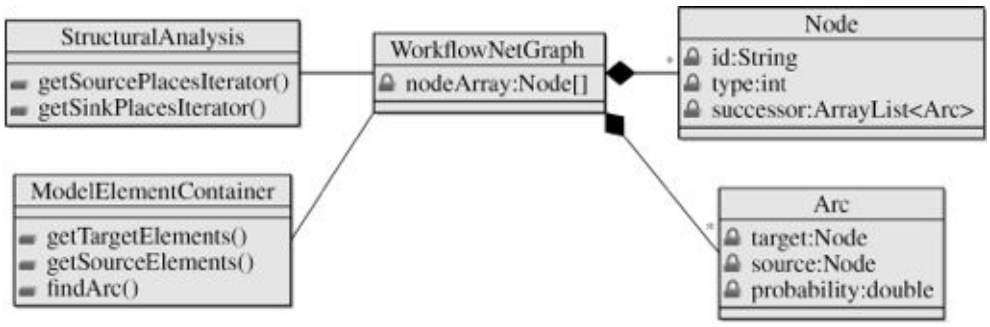


Abbildung 20: Die Schnittstelle "WorkflowNetGraph"

Für die Berechnung stützt sich die Klasse auf die Methoden von `StructuralAnalysis` und `ModelElementContainer` ab. `StructuralAnalysis` liefert die Quelle und Senke des Netzes. Der `ModelElementContainer` entält alle Netzbausteine.

Durch den direkten Zugriff auf das Model werden immer aktuelle Daten verwendet, so dass es keine Diskrepanz zwischen dem Graphen gibt, den der Editor anzeigt und dem WorkflowNetGraph. Die Dialoge für die Kapazitätsplanung ( ) und die Simulation ( ) werden über je einen Button in der Symbolleiste oder durch Auswahl aus dem Analyse-Menü geöffnet. Dies geschieht modal, d.h. es besteht dann kein Zugriff mehr auf den Editor, solange bis die Dialoge wieder geschlossen sind. Damit kann auch während der Analyse keine zwischenzeitliche Änderung am Modell vorgenommen werden.

In beiden Dialogen können die Parameter eingegeben werden, mit denen die Analyse durchgeführt werden soll. Einzelheiten dazu werden in den beiden folgenden Abschnitten vorgestellt. Eine Besonderheit, die bei beiden Verfahren Anwendung findet, ist das `TimeModel`. Damit wird sichergestellt, dass alle Zeitparameter bei den Berechnungen dieselbe Zeiteinheit haben. Die Zeiteinheit kann vom Anwender aus einem Kombinationslistenfeld ausgewählt werden.

Dieser einleitende Teil soll mit einer Bemerkung zur Erzeugung der Zufallszahlen für die Simulation beendet werden. Das Thema rund um die Generierung von (Pseudo-) Zufallszahlen und den stochastischen Tests, um ihre praktische Tauglichkeit zu überprüfen, ist sehr umfangreich. Hinzu kommt, dass aus gleichverteilten Zufallszahlen solche zu erzeugen sind, die einer bestimmten Wahrscheinlichkeitsverteilung genügen. Es gibt Verfahren wie die Inversionsmethode, um das zu erreichen. Die Betrachtung dieses speziellen Aspektes hätte den Rahmen dieser Diplomarbeit weit überschritten[29].

Daher wurde eine open-source-Bibliothek verwendet, die die Generierung von Zufallszahlen verschiedenster Verteilungen leisten kann. Es handelt sich dabei

[29] Als Beispiele für Literaturquellen seien genannt: D. KNUTH. *The Art of Computer Programming*. 3rd ed., vol. II, Seminumerical Algorithms, Addison-Wesley, 1998 sowie [Law07, Kap. 7-8] und [Fish01, Kap. 8-9].

um "Michael Thomas Flanagan's Java Scientific Library" von Dr. Michael Thomas Flanagan, Department of Electronic and Electrical Engineering, University College London[30].

5.1 Algorithmus zur Kapazitätsanalyse

Wie in Kapitel 3 beschrieben, werden für die Kapazitätsplanung außer der Netzstruktur die Ressourcenzuweisung und die Angabe der mittleren Ankunftsrate neuer Fälle benötigt. Außerdem ist der Zeitraum für eine Betrachtungsperiode anzugeben, auf den sich die Ankunftsrate bezieht.

Die Prozesstruktur wird in einer Instanz von `WorkflowNetGraph` verwaltet. Das Ressourcenmodell sowie die Ressourcenzuweisung werden ebenso zentral in einer Instanz der Klasse `ResourceAllocation` des Paketes `org.woped.quantana.resourcealloc` gekapselt.

Dort gibt es je eine Liste für die Ressourcenobjekte, die Gruppen und die Rollen. Hier wird auch aus der Zuordnung von Gruppen und Rollen zu Tasks und von Ressourcen zu Ressourcenklassen die Zuordnung von Tasks zu Ressourcenklassen abgeleitet. Damit kann Schritt 3 der Kapazitätsplanung (Berechnung des Zeitbedarfs pro Klasse) durchgeführt werden.

Die Schritte 2 (Zeitbedarf pro Transition) und 4 (Bestimmung der Anzahl benötigter Ressourcen) sind relativ problemlos. Die Implementierung orientiert sich an den angegebenen Formeln. Der Auslastungsgrad ist für alle Ressourcen gleich und kann im Dialog über einen Schieberegler eingestellt werden. Standardwert ist 80%.

Die eigentliche Herausforderung bei der Implementierung der Kapazitätsanalyse ist die Berechnung der Ausführungshäufigkeiten jedes einzelnen Tasks. Während für die Sequenz und die XOR-Verzweigungen die angegebenen Formeln direkt umgesetzt werden können und ein AND-Split ebenfalls recht einfach zu behandeln ist, machen Schleifen die Analyse äußerst kompliziert.

Das Problem besteht darin, dass ein allgemeingültiges Verfahren - wie es hier angestrebt ist - mit jedem gültigen Workflow-Netz umgehen können muss[31]. In so einem Netz kann aber eine verwickelte Struktur von untereinander abhängigen Schleifen existieren, die eine Berechnung im Zuge einer Tiefen- oder Breitensuche durch den Graphen *unmöglich* machen. In solch einem Fall ist die Kenntnis der gesamten Netzstruktur notwendig, was i.d.R. nur durch ein umfangreiches

[30] Siehe `http://www.ee.ucl.ac.uk/~mflanaga/java/index.html`. Die Bibliothek trägt den Namen *flanagan.jar*. Die Erlaubnis zur nicht-kommerziellen Benutzung ist unter dem genannten Link zu finden.

[31] Das gilt ohne Berücksichtigung der Beschränkungen durch die Komplexität der Speicherauslastung oder der Laufzeit.

Gleichungssystem möglich ist. Im Anhang D werden weitere untersuchte Methoden skizziert, die ebenfalls gescheitert sind.

Die letztlich verwendete Methode ist ein Näherungsverfahren, das auf dem Prinzip der Netzentfaltung beruht (vgl. [McMi95]). Dabei werden beim Traversieren eines gerichteten Graphen ausgehend von einem beliebigen Knoten Kopien der Nachfolgerknoten und der Kanten zu ihnen erzeugt. Das Verfahren wird dann rekursiv auf alle eben kopierten Nachfolgerknoten angewendet. Auf diese Weise entsteht ein neuer Graph, der keine Schleifen mehr enthält. Ohne eine Abbruchbedingung aber wird dieser Graph bei Existenz mindestens einer Schleife unendlich groß.

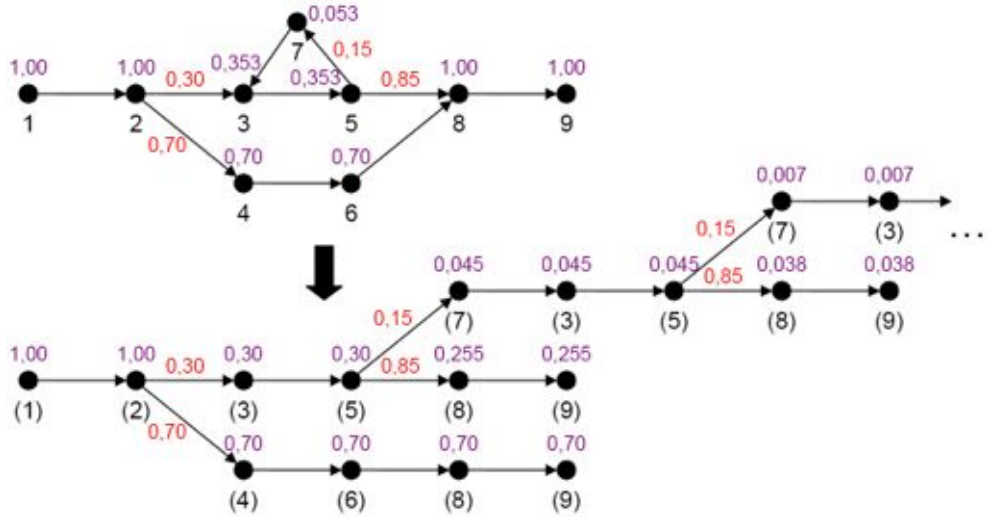


Abbildung 21: Beispiel für eine Netzentfaltung

In der Abbildung ist ein einfaches Netz mit einer XOR-Verzweigung (ohne den Zyklus) und einer Schleife zu sehen[32]. Dafür wurden die ersten Schritte der Entfaltung durchgeführt. Außerdem wurden beispielhaft Verzweigungswahrscheinlichkeiten und die Ausführungshäufigkeiten berechnet. Entlang eines Pfades werden die Wahrscheinlichkeiten aufeinanderfolgender Verzweigungen multipliziert. Daraus folgt, das die Folge der Häufigkeiten in jedem Pfad in Flussrichtung monoton fallend ist[33].

Für den abgebildeten Ausschnitt des entfalteten Netzes gilt für die Summe der Häufigkeiten aller Knoten, die Kopien desselben Originalknotens sind, dass sich diese Summe von unten an den theoretischen Wert für die Häufigkeit des Originalknotens annähert, wenn die Zahl der Entfaltungsschritte gegen unendlich geht. Dies ist in der nachfolgenden Tabelle nochmals zu sehen:

[32] Legende: rote Zahlen = Verzweigungswahrscheinlichkeiten; violette Zahlen = Ausführungshäufigkeiten; schwarze Zahlen = Knotennummer; Zahlen in Klammern ( ) = Knoten-Kopien

[33] Es handelt sich um ein Produkt, dessen Faktoren alle positiv und kleiner als Eins sind.

i-te Kopie	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1,0	1,0	0,3	0,7	0,3	0,7	0,045	0,255	0,255
2			0,045		0,045		0,007	0,7	0,7
3			0,007					0,038	0,038
Summe	1,0	1,0	0,352	0,7	0,345	0,7	0,052	0,993	0,993
theoretisch	1,0	1,0	0,353	0,7	0,353	0,7	0,053	1	1
Fehler (%)	0,0	0,0	0,283	0,0	2,266	0,0	1,887	0,7	0,7

Tabelle 4: Ausführungshäufigkeiten im entfalteten Netz

Damit existiert eine Möglichkeit, den Entfaltungsalgorithmus kontrolliert abzubrechen: Es wird eine Schranke ε für die Häufigkeiten definiert, bei deren Unterschreiten der aktuelle Pfad nicht fortgeführt wird. Wenn es auf diese Weise dann keinen Pfad mehr gibt, der noch Nachfolger produziert, terminiert der Algorithmus. Das Ergebnis ist dann ein Baum, dessen Blätter entweder nicht zu einem Zyklus im Originalgraph gehören (und daher möglicherweise schon vor Erreichen von ε den Pfad beenden) oder deren Häufigkeit so nahe an ε heranreicht, dass mit dem nächsten Knoten ε bereits unterschritten wird.

In der Implementierung beginnt der Traversierungsalgorithmus naheliegenderweise bei dem Knoten im `WorkflowNetGraph`, der der Quelle im Workflow-Netz entspricht. Als Ausführungshäufigkeit des ersten Knotens wird die Ankunftsrate λ verwendet.

Dadurch relativiert sich die Bedeutung von ε als Maß für die Genauigkeit der Ergebnisse. Um eine grobe Abschätzung der Brauchbarkeit der Resultate zu geben, wird der relative Fehler der Ausführungshäufigkeit der Senke angezeigt, da sie außer der Quelle der einzige Knoten ist, für den der Sollwert immer bekannt ist, nämlich λ. Dabei ist zu beachten, dass der Fehler für andere Knoten des Netzes durchaus größer sein kann, wie in der Tabelle 4 für Knoten 5 beispielhaft zu sehen ist.

Als Traversierungsmethode bei der Entfaltung wurde die Breitensuche der Tiefensuche vorgezogen, weil AND-Joins gesondert zu behandeln sind und die Breitensuche sich als effizienter herausgestellt hat. Für AND-Joins gilt nach den Ausführungen zum Schritt 1(c) (Seite 43), dass alle n eingehenden Kanten denselben Wert haben. Es darf aber nur eine Kopie des AND-Join-Knotens erstellt werden, weil sonst $(n-1)$ nachfolgende Pfade in der Entfaltung zu viel wären. Das Ergebnis wäre statt 100% als relative Häufigkeit für die Senke ein Wert von $n \cdot 100\%$. Daraus folgt ein weiteres Abbruchkriterium von Pfaden in der Entfaltung.

Der rekursive Entfaltungsalgorithmus ist im Anhang E zu sehen und erläutert. Der Algorithmus zur Kapazitätsplanung sieht dann so aus:

```
1 private void calculateNumOfRuns() {
2       unfoldNet(graph, lambda, epsilon);
3       Node[] origNet = graph.getNodeArray();
4       for (Key k : unfoldedNet.keySet()) {
5             String id = k.getId();
6             Node n = origNet[graph.getNodeIdx(id)];
7             n.setNumOfRuns(n.getNumOfRuns()+k.getRuns());
8       }
9 }
```

Zuerst wird die Entfaltung durchgeführt. Das entfaltete Netz liegt dann in der Datenstruktur `unfoldedNet` vor. Anschließend werden alle `Key`-Objekte von `unfoldedNet` durchlaufen. Über die in den Schlüsseln gespeicherten Knoten-IDs werden jeweils die Knoten aus dem Originalgraphen bestimmt und ihre Zählerwerte für die Ausführungshäufigkeiten um den zweiten Wert des `Key`-Objektes erhöht. Danach haben alle Knoten ihren Näherungswert für die Ausführungshäufigkeit.

Bevor der gesamte Algorithmus für die Kapazitätsplanung vorgestellt wird, sollen noch einige Datenstrukturen erläutert werden:

- `tasksMatrix`: Tabelle, die die Daten der Tasks speichert:

Spalte 0	Spalte 1	Spalte 2	Spalte 3	Spalte 4
Bedienzeit	Anzahl der Ausführungen pro Fall	Zeitbedarf pro Fall	Anzahl der Ausführungen pro Periode	Zeitbedarf pro Periode

- `resMatrix`: Tabelle, die die Daten der Ressourcenklassen speichert:

Spalte 0	Spalte 1
Gesamter Zeitbedarf pro Ressourcenklasse	Anzahl der dafür benötigten Ressourcen

- `numTrans`: Anzahl der Transitionen (Tasks)
- `numResCls`: Anzahl der Ressourcenklassen
- `numNodes`: Anzahl der Knoten des entfalteten Netzes
- `deviation`: Relativer Fehler, der durch den Abbruch der Entfaltung entsteht
- `currUtil`: gemeinsamer mittlerer Auslastungsgrad aller Ressourcen
- `ResourceAllocation`: Klasse , in der die Gruppen, Rollen und Ressourcen verwaltet werden

- `ResourceClassTaskAllocationTable`: Klasse, die eine Tabelle enthält, in der den Ressourcenklassen die Tasks zugeordnet werden

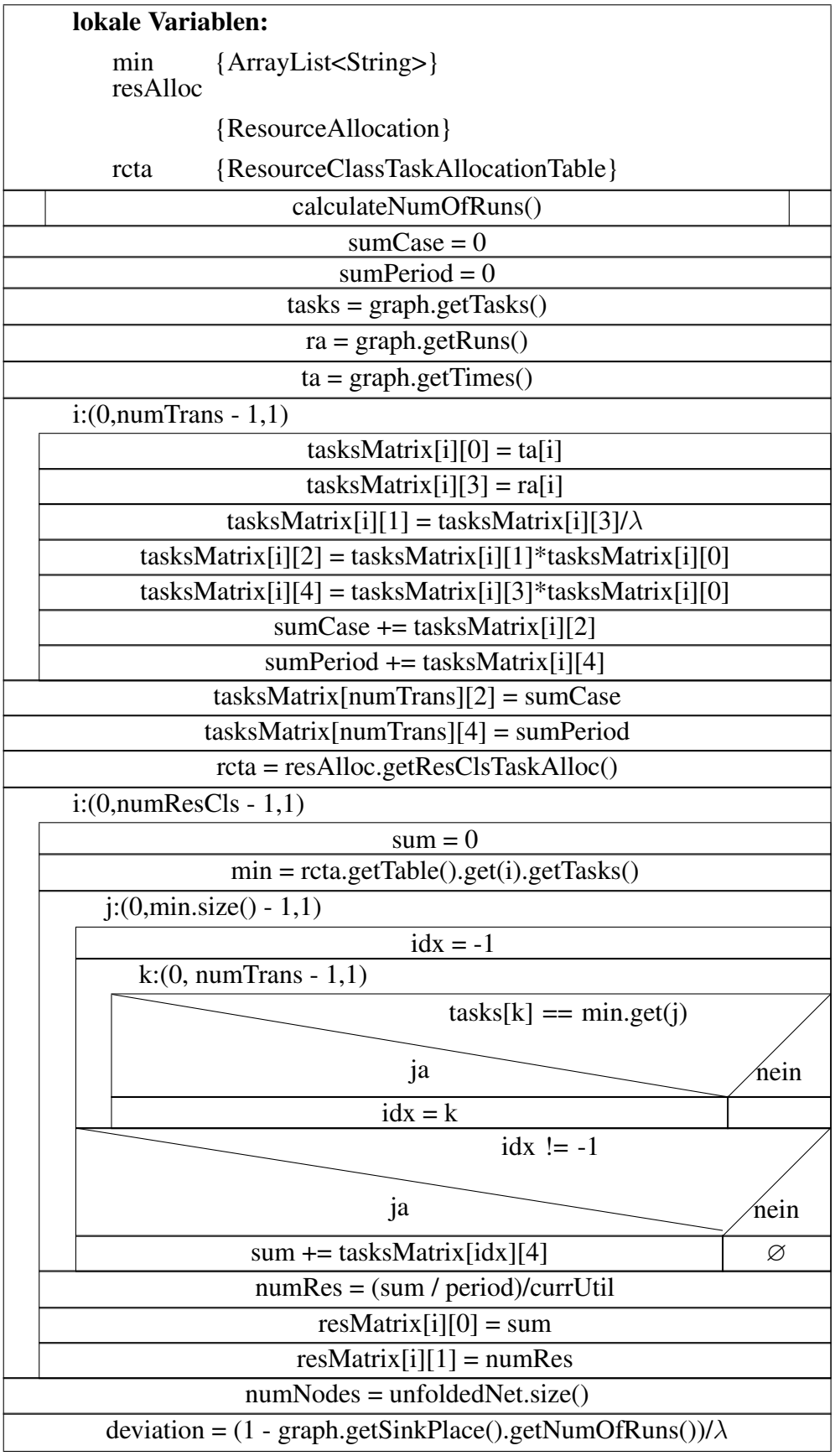
Außerdem werden folgende Methoden verwendet:

- `WorkflowNetGraph.getTasks()`: Gibt ein Array mit den Namen der Tasks zurück
- `WorkflowNetGraph.getRuns()`: Gibt ein Array mit den Ausführungshäufigkeiten in der Reihenfolge von getTasks() zurück
- `WorkflowNetGraph.getTimes()`: Gibt ein Array mit den Bedienzeiten in der Reihenfolge von `getTasks()` zurück
- `WorkflowNetGraph.getSinkPlace()`: Gibt den Endknoten des Graphen = die Senke des Workflow-Netzes zurück
- `ResourceAllocation.getResClsTskAlloc()`: Gibt eine `ResourceClassTaskAllocationTable`-Instanz zurück
- `ResourceClassTaskAllocationTable.getTable().get(i).getTasks()`: Gibt ein Array mit den Namen der Tasks zurück, die der durch `i` spezifizierten Ressourcenklasse zugeordnet sind
- `Node.getNumOfRuns()`: Gibt die Ausführungshäufigkeit eines `Node`-Objektes zurück

Das Struktogramm für die Kapazitätsplanung ist auf der nächsten Seite zu sehen. Abschließend noch ein paar Worte zur Komplexität des Algorithmus: Außer den Methodenaufrufen sind alle Zeilen konstant in der Zeit. Weiterhin gilt: $\aleph[\mathit{getResClsTskAlloc}()] = \aleph[\mathit{getTable}().\mathit{get}(i).\mathit{getTasks}()] = O(1)$, weil die notwendigen Berechnungen bereits bei Erzeugung der Instanz `resAlloc` durchgeführt wurden. Für die Entfaltung werden Initialisierung und Rekursion getrennt betrachtet. Es gilt: $\aleph[\mathit{unfoldNet}()] = O(n)$, wobei n die Anzahl der Knoten ist, und $\aleph[\mathit{runThroughNet}()] = O(|E|+|K|)$ mit $|E| = n$ Anzahl der Knoten und $|K|$ Anzahl der Kanten. Das entspricht dem Aufwand einer Breitensuche.

Damit gilt für die Entfaltung: $\aleph[\mathit{calculateNumOfRuns}()] = O(m) = O(2n+|K|)$, also linearer Aufwand. Für die gesamte Kapazitätsplanung folgt dann die Komplexität $O(m) + O(\mathit{numTrans}) + O(\mathit{numResCls}) \cdot O(\mathit{min.size}()) \cdot O(\mathit{numTrans}) \in O(n^3)$. Das Produkt entsteht durch die Schachtelung der drei Schleifen. Der Aufwand ist polynomial. Er hängt insbesondere von der Anzahl der Tasks, aber auch von der Komplexität der Ressourcenzuweisung ab.

Kapazitätsplanung:



5.2 Algorithmus zur Simulation

Die Implementierung des Simulationsalgorithmus orientiert sich prinzipiell an dem im Kapitel 4 Gesagtem. Es gibt aber einige bedeutende Unterschiede, die erst bei der Betrachtung im Detail auftauchen. Zunächst wird eine statische Sicht auf die Simulation vorgestellt. Darin finden sich die wesentlichsten Komponenten. Danach erfolgt die Darstellung des dynamischen Aspektes, der mittels Interaktion von Simulationsereignissen umgesetzt wird.

Die Hauptfunktionalität der Simulation befindet sich im Paket `org.woped.quantana.simulation`. Der Simulator interagiert aber auch mit einer Vielzahl anderer Klassen aus anderen Paketen. Im Anhang F ist das Paketdiagramm mit den Beziehungen zu sehen.

Die Elemente einer Simulation werden durch einzelne Klassen implementiert. Ihre Beziehung untereinander ist im Klassendiagramm ebenfalls im Anhang F zu finden. Im Einzelnen sind es:

- `Simulator` stellt die Abstraktion einer Simulationsstudie dar und steuert den Ablauf.
- `Server` ist die Bedienstation, an der Fälle (Cases) durch Ressourcen bearbeitet werden.
- `Case` ist ein Fall, der durch den Prozess geroutet wird.
- `Resource` stellt ein Ressourcenobjekt dar.
- `SimEvent` ist die abstrakte Klasse für Simulationsereignisse. Es werden zwei Attribute verwendet: (1) eine Referenz auf den zugeordneten Simulator und (2) ein Zeitstempel. Die Zeit entspricht der Modellzeit, zu der das Ereignis stattfindet.
- In der `eventList` werden die anstehenden Ereignisse gesammelt. Die verwendete Datenstruktur ist eine PriorityQueue, die als nächstes Element immer ein Ereignis mit der kleinsten Zeit zurückgibt. Es ist zu beachten, dass durchaus mehrere Ereignisse denselben Zeitstempel haben. Die Auswahl erfolgt dann beliebig.
- Die `serverList` enthält alle Server. Sie ist als HashMap implementiert, um einen schnellen Zugriff auf ein `Server`-Objekt zu erhalten.
- In der `copiedCasesList` werden alle Fälle gespeichert, von denen zurzeit Kopien im Prozess existieren. Die hier gespeicherten Originale werden nach dem AND-Join wieder benötigt.

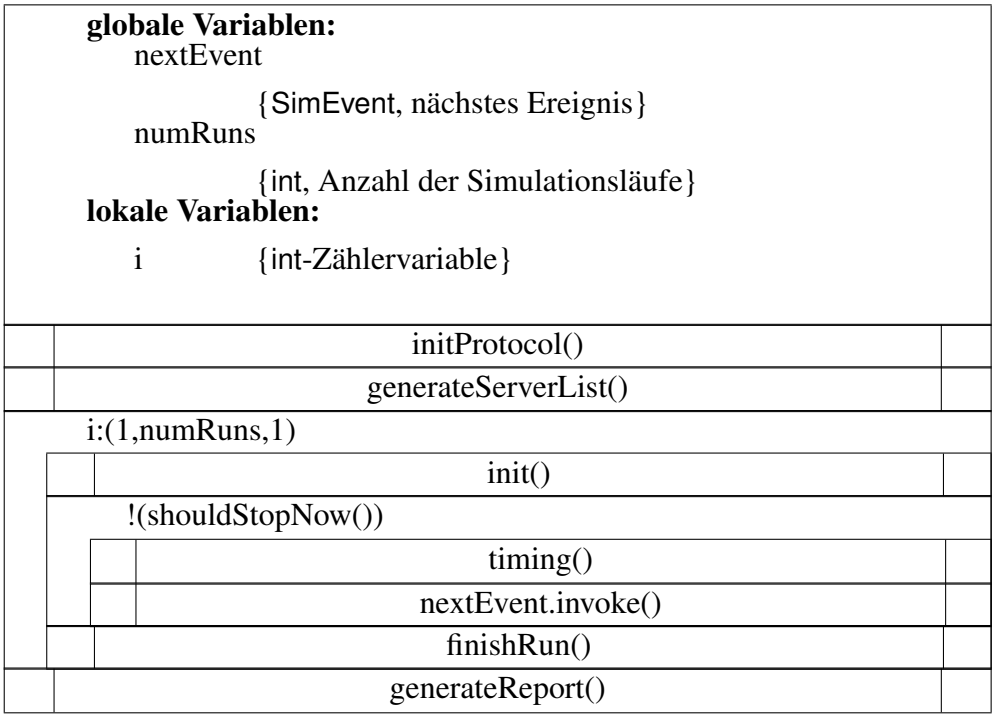
- `ProbabilityDistribution` ist die Abstraktion einer Wahrscheinlichkeitsverteilung. Sie bekommt die vom Anwender eingestellten Parameter und erzeugt daraus Zufallszahlen.
- `CaseGenerator` ist die Quelle, mit der Fälle produziert werden. Er verwendet ein ProbabilityDistribution-Objekt, um die zufälligen Ankunftzeiten zu berechnen.
- `SeedGenerator` erzeugt von der aktuellen Systemzeit des Computers abhängige Startwerte (”seeds”) für alle Zufallszahlengeneratoren. Jeder seed-Wert kommt höchstens einmal vor.

Die Durchführung der Simulation, wie sie hier vorgestellt wird, beachtet keine Einschwingphase. Das wird aus Gründen der Zeit- und Speichereffizienz gemacht, um die Zahl der Berechnungen zu reduzieren. Zudem folgt aus den auf Seite 55 erwähnten Rekursionsformeln, dass zum Zeitpunkt s des ”eigentlichen” Beginns der Simulation (= Ende der Einschwingphase) die aktuelle Warteschlangenlänge $Q(s)$ und die aktuelle Anzahl beschäftigter Ressourcen pro Server $B(s)$ als Startwerte weiterverwendet werden. Die Berechnung der zeitkontinuierlichen Mittelwerte liefert dann keine anderen Ergebnisse.

Lediglich für die Berechnung der zeitdiskreten mittleren Wartezeit müßte die Anzahl abgeschlossener Fälle zum Zeitpunkt s auf Null zurückgesetzt werden. In den Formeln geschieht dies rein rechnerisch, indem $N(s,t) = N(0,t) - N(0,s)$ berechnet wird. Der Zeitpunkt s ist aber von t unabhängig und mit wachsendem t sinkt der Unterschied zwischen $N(s,t)$ und $N(0,t)$. Durch die Wahl einer geeignet großen Periode pro Simulationslauf kann also der Fehler kleingehalten werden.

Der Ablauf einer Simulation stellt sich so dar:

start():



Es wird mit den im Benutzerdialog eingegebenen Parametern ein `Simulator`-Objekt erzeugt. Die Simulation beginnt dann mit Aufruf der Methode `Simulator.start()`. Dann wird das Protokoll initialisert, das fortan alle wesentlichen Schritte der Simulation als LogRecords im Speicher hält. Es ist als Logger implementiert.

Anschließend wird die `serverList` generiert. Das bedeutet, für jede Transition im `WorkflowNetGraph` wird eine `Server`-Instanz erzeugt. Jeder Server erhält zudem eine Liste `successor` der Nachfolger-Server `SuccServer`, wobei die Verzweigungswahrscheinlichkeiten bei mehreren Nachfolgern gespeichert werden. Mit der Methode `gotoNextServer()` kann diesen Wahrscheinlichkeiten entsprechend der nächste Server bestimmt werden, zu dem ein Fall weitergeleitet werden soll.

Anschließend beginnt die Zählschleife für die einzelnen Simulationsläufe. Jeder Lauf simuliert eine Periode. Die Anzahl der Durchläufe ist in `numRuns` gespeichert. Pro Simulationslauf findet eine Initialisierung statt, indem die Methode `init()` aufgerufen wird. Danach wird mit `timing()` das nächste eintretende Ereignis bestimmt und dessen `invoke()`-Methode ausgeführt.

Bei jedem Schleifendurchlauf wird die Abbruchbedingung überprüft. Der Anwender kann wählen zwischen:

- *caseDriven*: Wenn λ Fälle abgeschlossen sind, endet der Simulationslauf.
- *timeDriven*: Wenn die Zeit für eine Periode erreicht oder überschritten wurde, endet der Simulationslauf.

- *both*: Die erste der beiden Bedingungen, die erfüllt wird, beendet den Simulationslauf.

Mit Beendigung eines Simulationslaufes werden die Ergebnisse der Berechnungen gespeichert. Sind alle Simulationsläufe abgeschlossen, wird der Reportgenerator aufgerufen, der die Zusammenfassung der Ergebnisse durch Mittelwertbildung vornimmt. Anschließend erfolgt die Darstellung der Resultate im Benutzerdialog.

Die Methoden `init()` und `timing()` werden durch ihre Struktogramme im Anhang G beschrieben:

Die Aktualisierung der Zustandsvariablen und die Berechnung der Effizienzmessgrößen erfolgt in der `invoke()`-Methode der Simulationsereignisse. Je nach Ereignistyp finden unterschiedliche Berechnungen statt. Die Struktogramme sind im Anhang G zu finden.

Nun wird die Dynamik des Simulationsalgorithmus vorgestellt. Zunächst wird die Menge der Simulationsereignisse weiter ergänzt[34]. Ankunftsereignis (`ArrivalEvent`, AE), Endeereignis (`DepartureEvent`, DP) und Startereignis (`StartServiceEvent`, ST) behalten ihre Bedeutung. Darüberhinaus existieren `BirthEvent` (BE) und `DeathEvent` (DE), um die Ankunft eines Falles im Prozess (Geburt) bzw. sein Verschwinden aus dem Prozess (Tod) zu signalisieren.

Es wurde bei der Implementierung darauf geachtet, dass eine Simulation auch ohne (explizite) Ressourcenzuweisung durchführbar ist. Die Semantik des Geschäftsprozessmodells ändert sich dadurch in der Weise, dass jeder Task von genau einer anonymen Ressource bearbeitet wird. Die maximale Anzahl parallel bearbeiteter Fälle ist damit Eins. Daraus folgt auch, dass die durchschnittliche Ressourcenauslastung in diesem Fall der Serverauslastung entspricht. Gleiches gilt bei Verwendung eines Ressourcenmodells für die Tasks, die nur die Steuerung des Kontrollflusses übernehmen und i.d.R. keine Ressource und keine Bedienzeit zugeordnet bekommen.

Für die Unterscheidung beider Fälle - Ressourcenmodell ja oder nein - wird ein Flag benutzt, dass während der Simulation abgefragt werden kann. Durch diese Unterscheidung ist es notwendig, ein weiteres Simulationsereignis einzuführen: Das `StopServiceEvent` (SP) befreit die zuvor gebundene Ressource bzw. setzt den Zustand des Servers auf ”frei” (idle).

Das dynamische Verhalten der Simulation wird in der folgenden Abbildung nochmal graphisch zusammengefasst:

[34]Es werden von jetzt an die aus dem Englischen abgeleiteten Bezeichnungen, wie sie auch bei der Programmierung verwendet werden, benutzt.

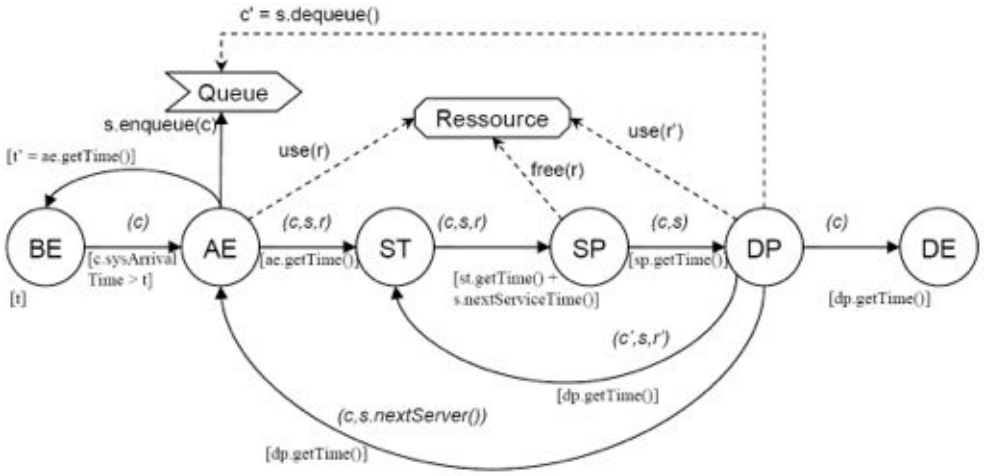


Abbildung 22: Interaktion der Simulationsereignisse

Das erste Ereignis ist ein BirthEvent, welches die Simulation zum Zeitpunkt Null startet und ein ArrivalEvent für den ersten Fall erzeugt. Ein ArrivalEvent erzeugt ein weiteres BirthEvent, so dass weitere Fälle in den Prozess eintreten können. Mit dieser Vorgehensweise ist sichergestellt, dass zu einem Zeitpunkt nur ein Fall ankommt, zugleich aber die Zeit bis zur nächsten Ankunft beliebig klein sein kann. Somit bestehen für die Ankünfte keine künstlichen Restriktionen.

Das ArrivalEvent prüft weiterhin, ob der aktuelle Fall am aktuellen Server sofort in Bearbeitung gehen kann oder in die Warteschlange kommt. Wenn der Server über freie Kapazität verfügt, wird ein StartServiceEvent generiert und ein Ressourcenobjekt für die Bearbeitung gebunden. Das StartServiceEvent bestimmt die Bedienzeit für den übergebenen Fall und erzeugt ein passendes StopServiceEvent.

Durch das StopServiceEvent wird die Bearbeitung beendet und die gebundene Ressource wieder befreit. Im anschließend generierten DepartureEvent wird die Weiterleitung des Falles vorgenommen. Gibt es keinen Nachfolger-Server mehr, verläßt der Fall den Prozess, was durch ein DeathEvent vollzogen wird. Ansonsten wird ein ArrivalEvent für den nächsten Server veranlaßt.

Außerdem wird geprüft, ob die Warteschlange weitere Fälle enthält. Wenn ja und zudem Ressourcen frei sind, werden StartServiceEvents für diese Fälle erzeugt. Die Entnahme aus der Warteschlange erfolgt gemäß der vom Anwender bestimmten Warteschlangendisziplin.

Eine Besonderheit stellt das Routing für nebenläufige Prozesspfade dar. Das wird im StartServiceEvent bzw. im DepartureEvent berücksichtigt. Um nämlich bei der Datenhaltung in den Statistikzählern an verschiedenen Servern für den gleichen Fall nicht in Konflikte zu geraten, werden vom AND-Split-Server Kopien statt des Original-Falles weitergeleitet. Diese Kopien sammeln die Daten zu Bedien- und Wartezeiten und werden am AND-Join-Server gesammelt. Sind al-

le Kopien eines Falles vollständig, werden ihre Daten mit denen des Originals zusammengefaßt. Anschließend wird der Origianl-Fall weitergeleitet.

Die Datenhaltung verläuft nach folgendem Schema:

- beim Split: beginnt zum Zeitpunkt t_s
  - Original c: kumulierte Wartezeit w_c und kumulierte Bedienzeit s_c bisher
  - Kopien $c_1, c_2, ..., c_n$ mit $w_{c_i} = s_{c_i} = 0$ für alle $i \in \{1, 2, ..., n\}$
- während des Splits:
  - c bleibt unverändert
  - c_i kumuliert Werte für w_{c_i} und s_{c_i} während der Simulation
- beim Join: zum Zeitpunkt t_j, wenn alle Kopien vorliegen
  - $\delta s_c = \frac{1}{n} \sum_{i=1}^{n} s_{c_i}$ durchschnittliche Bedienzeit und $\delta w_c = t_j - t_s - \delta s_c$ durchschnittliche Wartezeit
  - für c gilt ab jetzt: $w_c = w_c + \delta w_c$ und $s_c = s_c + \delta s_c$
  - Kopien werden gelöscht

Die Wartezeit vom ersten Eintreffen einer Kopie von c bis zum Eintreffen der letzten Kopie von c (= Beginn der Bedienung am AND-Join-Server) ist Leerzeit des Servers.

6 Bewertung und Ausblick

Die quantitative Analyse von Geschäftsprozessen läßt sich theoretisch sehr gut begründen. Insbesondere wegen ihres formalen Charakters sind Petri-Netze sehr gut zur Modellierung von Geschäftsprozessen geeignet und liefern mit einigen Ergänzungen eine hervorragende Grundlage für ihre Analyse.

Als zwei spezielle Verfahren wurden die Kapazitätsplanung und die Simulation betrachtet. Die inhaltliche Verwandtschaft von Simulation, Markov-Analyse und Warteschlangentheorie wurde verdeutlicht. Es stellte sich heraus, dass Simulation ein mächtiges Werkzeug der quantitativen Analyse ist, das praktisch immer einsetzbar ist, auch wenn analytische Verfahren versagen.

Der Forderung, eine Einheit von Modellierungswerkzeug und Analysetool zu schaffen, wurde durch die Implementierung der beiden ausgewählten Methoden entsprochen. Dazu wurde die Funktionalität von WoPeD entsprechend ergänzt. Aufbauend auf dem verifizierten und mit quantitativen Spezifikationen ergänzten Workflow-Netz wird der zugrundeliegende Graph mit den wesentlichen Informationen gespeichert und der Analyse unterzogen.

Die Kapazitätsplanung ist im Großen und Ganzen recht einfach und der formal definierte Algorithmus läßt sich direkt implementieren. Ein großes Problem ist aber die Berechnung der Häufigkeiten, wie oft jede Transition im Schnitt durchlaufen wird. Hier wurde eine Näherungslösung verwendet, die auf der Technik der Netzentfaltung beruht. Wie jede Approximation hat sie die Nachteile, bei komplexen Problemen sehr aufwendig zu werden und an Qualität in den Ergebnissen einzubüßen.

Einen Algorithmus zur Simulation zu entwickeln, war schwieriger. Die Herausforderung bestand darin, Folgen von Ereignissen zu erzeugen, die die konkreten Pfade von Fällen durch den simulierten Prozess abbilden und dabei die Daten sammeln, die zur Berechnung der quantitativen Messgrößen benötigt werden. Auch hier gab es eine Besonderheit. Die Simulation von Parallelität erfordert die Verwaltung von Kopien eines Falles, die unabhängig voneinander bis zu ihrer Synchronisation vom Prozess bearbeitet werden.

Die vorgestellte und implementierte Methode stellt sicher, dass die Semantik des Prozesses nicht verändert wird. Aus quantitativer Sicht bedeutsam ist die Synchronisation, bei der alle Kopien eines Falles gesammelt und ihre Daten aggregiert werden.

Natürlich ist das Thema "*Quantitative Analyse von Geschäftsprozessen*" damit nicht abgeschlossen. Die Anzahl der Möglichkeiten zur weiteren Entwicklung der Algorithmen ist schlichtweg unbegrenzt:

So bietet es sich beispielsweise an, zu untersuchen, ob und wie das Problem der Ausführungshäufigkeiten von Transitionen analytisch gelöst werden kann. Eine effiziente Implementierung eines solchen Algorithmus würde den Wert des Analysewerkzeugs steigern. Vorstellbar ist die Lösung eines Gleichungssystems ähnlich der Verwendung der ”Maschenregel” in der Elektrotechnik, angewendet auf das kurzgeschlossene Workflow-Netz. Parallelität ist auch hier wieder die Crux.

Eine Verknüpfung der Kapazitätsplanung mit einem System der ad-hoc-Verwaltung personeller oder maschineller Ressourcen würde die Software einen weiteren Schritt in Richtung eines Workflow-Management-Systems bringen.

Sicherlich ist es auch möglich, nach weitergehenden Untersuchungen des Themenkomplexes die Effizienz der Algorithmen bezüglich der Speicherbelegung wie auch des Zeitbedarfs zu verbessern.

Im Moment werden Subprozesse wie gewöhnliche Transitionen behandelt. Die Weiterentwicklung der Algorithmen wird die Betrachtung mehrerer Hierarchieebenen berücksichtigen müssen. Nur so ist eine weitreichende Nutzung möglich. Ebenso wie beim Gesamtnetz bietet sich hier als Strategie die Rekursion an. Bei der niedrigsten Hierarchieebene startet die Berechnung und die Ergebnisse liefern die Eingabedaten für die Kalkulation auf der nächsthöheren Stufe. Die Eigenschaft von Subprozessen, eigenständige Workflow-Netze zu sein, die sound und safe sind, macht das möglich.

Auch die Simulation ist in vielerlei Hinsicht erweiterbar. Die verwendete Implementierung sieht vor, dass jede Simulator-Instanz ein eigenes CaseGenerator-Objekt besitzt. Es ist denkbar, eine Simulationsstudie mit einem ”compound-Simulator” durchzuführen, der verschiedene CaseGenerator-Objekte dazu verwendet, verschiedene Klassen von Fällen bereitzustellen, die einen gemeinsamen Prozess durchlaufen.

Der zeitliche Aufwand wurde bereits als ein Flaschenhals der Simulation identifiziert. Die Parallelsierung der Simulation, d.h. die Verteilung der Berechnungsschritte auf mehrere Prozessoren oder Rechner in einem Netzwerk würde die Effizienz dieser Analysemethode enorm steigern. Dazu ist das Modell in seine konstituierenden ”Logischen Prozesse” zu zerlegen, die dann getrennt, aber nicht unabhängig voneinander simuliert werden (vgl. [Law07, S. 61-64]). Auch hier ist eine korrekte Ereignisfolge ausschlaggabend für die Anwendbarkeit der Methode.

Schließlich bestehen auch Möglichkeiten, die graphische Benutzeroberfläche und Dialoggestaltung derart weiterzuentwickeln, dass die Ausführung einer Simulationsstudie interaktiv unterstützt wird. Modellierung – qualitative Analyse – quantitative Analyse würden damit zu Funktionseinheiten eines Workflows, der

die Entwicklung von Workflows beschreibt.

Dazu gehört auch die Entwicklung eines Frameworks, das getriggerte Aktivitäten verwalten kann. Zurzeit werden in der Simulation nur Transitionen ohne Trigger oder mit Ressourcen-Trigger berücksichtigt. Externe und Zeittrigger sind in späteren Versionen zu ergänzen.

Die Bedeutung quantitativer Analysetechniken wird auch in Zukunft von großer Bedeutung sein. Ihre Integration in Modellierungswerkzeuge und ihre stetige Weiterentwicklung und Verbesserung ist ein wichtiger Schritt zur Effizienzsteigerung. Ich hoffe, mit dieser Diplomarbeit etwas dazu beigetragen zu haben.

Anhang

Anhang A: Durchgehendes Beispiel eines Geschäftsprozesses

Als durchgehendes Beispiel für einen einfachen Geschäftsprozess zur Erläuterung des Haupttextes diene die Abwicklung einer Kundenbeschwerde in einem Dienstleistungsunternehmen nach dem Muster von [AaHe02, S. 81-84, 126-128].

Dazu stelle man sich vor, dass Kunden per Email eine Beschwerde beim Unternehmen abgeben können. Ein Mitarbeiter wertet die Emails grob aus und leitet eine Empfangsbestätigung an den Kunden und eine Bearbeitungsanweisung an die Support-Abteilung weiter. Liegen alle benötigten Daten vor, kann ein entscheidungsbefugter Mitarbeiter prüfen, ob dem Antrag auf finanzielle Rückerstattung stattgegeben wird (in ca. 63% der Fälle) oder nicht (ca. 27%). Bei etwa 10% aller vorgelegten Fälle fehlen spezifische Informationen, die nachträglich vom Kunden angefordert werden müssen. Liegen die Daten vor, kann die Beschwerde erneut zur Entscheidung vorgelegt werden. Ist die Entscheidung gefallen, wird entweder die Zahlung durch die Finanzabteilung veranlaßt oder ein Ablehnungsbescheid an den Kunden gesendet. Anschließend wird der Fall archiviert, womit der Prozess endet.

Damit enthält dieser einfache Prozess sequenzielle, parallele, alternative und sich wiederholende Abschnitte.

Als Grundlage für die Geschäftsprozessmodellierung mit einer ereignisgesteuerten Prozesskette, aber auch als Hintergrund für die anderen erwähnten Modellierungsmethoden werden hier das Organigramm, das Entity-Relationship-Diagramm und der Funktionshierarchiebaum aufgeführt. Die Modelle dienen vor allem der Veranschaulichung und sind daher nur ausschnittsweise dargestellt.

1. Organigramm

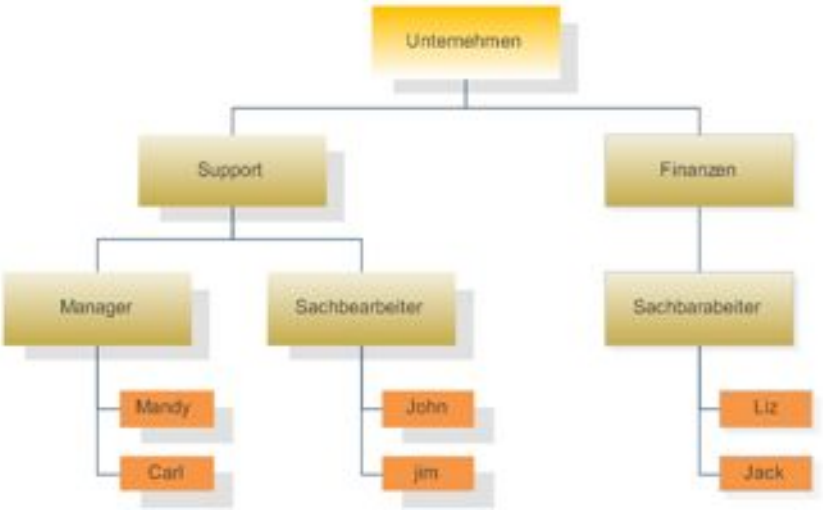


Abbildung 23: Organigramm des Beispielprozesses

2. Eintity-Relationship-Diagramm

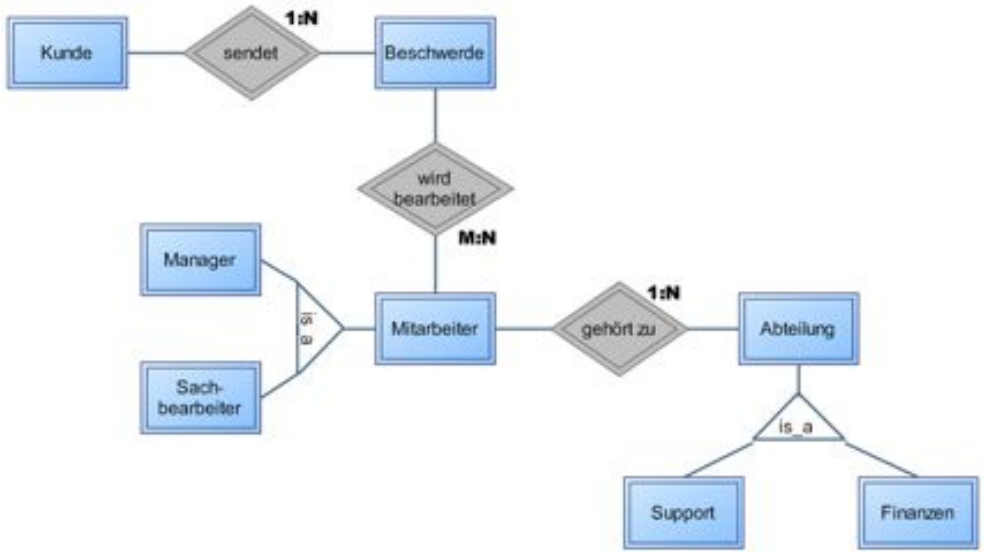


Abbildung 24: Entity-Relationship-Diagramm des Beispielprozesses

3. Funktionshierarchiebaum

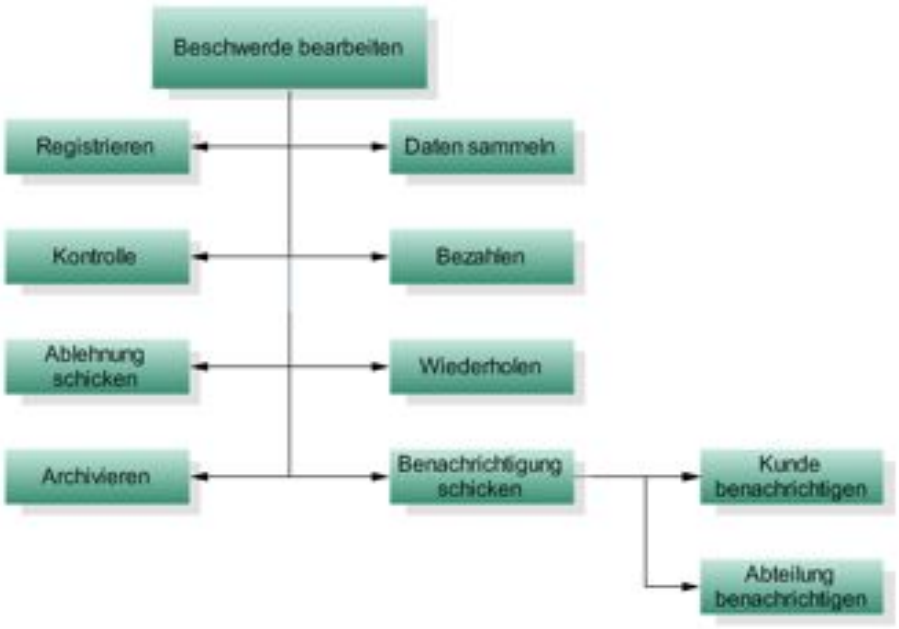


Abbildung 25: Funktionshierarchiebaum des Beispielprozesses

Anhang B: Netztransformationen

Sei $\mathbf{N} = (S, T, F)$ ein Petri-Netz. Ein Netz $\dot{\mathbf{N}} = (\dot{S}, \dot{T}, \dot{F})$ ist **Teilnetz** von $\mathbf{N}$, wenn gilt: $\dot{S} \subseteq S \wedge \dot{T} \subseteq T \wedge \dot{F} = F \cap ((\dot{S} \times \dot{T}) \cup (\dot{T} \times \dot{S}))$.

Die Knotenmenge $R_{\dot{\mathbf{N}},\mathbf{N}}$, definiert durch $R_{\dot{\mathbf{N}},\mathbf{N}} := \{k \in \dot{S} \cup \dot{T} \mid (k\bullet \cup \bullet k) \setminus (\dot{S} \cup \dot{T}) \neq \varnothing\}$, heißt **Rand** von $\dot{\mathbf{N}}$ bezüglich $\mathbf{N}$ (Vor- und Nachbereich jeweils in $\mathbf{N}$).

$\dot{\mathbf{N}}$ heißt **stellenberandet**, wenn $R_{\dot{\mathbf{N}},\mathbf{N}} \subseteq \dot{S}$.

$\dot{\mathbf{N}}$ heißt **transitionsberandet**, wenn $R_{\dot{\mathbf{N}},\mathbf{N}} \subseteq \dot{T}$.

Im Folgenden bezeichne $\dot{\mathbf{N}}$ ein transitionsberandetes Teilnetz von $\mathbf{N}$. Ferner sei $t \notin S \cup T$ eine neue Transition. Die Transformationsabbildung $\xi : (S \cup T) \times (S \cup T) \to (S \cup T \cup \{t\}) \times (S \cup T \cup \{t\})$ mit

$$\begin{aligned}
\xi(k,l) &= (k,l) \quad \text{wenn} \quad k,l \in (S \cup T) \setminus (\dot{S} \cup \dot{T}) \\
\xi(k,l) &= (k,t) \quad \text{wenn} \quad k \in (S \cup T) \setminus (\dot{S} \cup \dot{T}) \wedge l \in R_{\dot{\mathbf{N}},\mathbf{N}} \\
\xi(k,l) &= (t,l) \quad \text{wenn} \quad k \in R_{\dot{\mathbf{N}},\mathbf{N}} \wedge l \in (S \cup T) \setminus (\dot{S} \cup \dot{T})
\end{aligned}$$

stellt die Transformation des Netzes $\mathbf{N}$ zum vergröberten Netz $\hat{\mathbf{N}} = (\hat{S}, \hat{T}, \hat{F})$ her mit $\hat{S} = S \setminus \dot{S} \wedge \hat{T} = (T \setminus \dot{T}) \cup \{t\} \wedge \hat{F} = \xi(F \setminus \dot{F})$. $\hat{\mathbf{N}}$ heißt **Transitions-Vergröberung** von $\mathbf{N}$.

Da ein Teilnetz dem Begriff nach immer ein Teil eines anderen Netzes ist, ist die Definition der Verfeinerung nicht ganz so einfach. Ich beschränke daher diese Operation auf Workflow-Netze, um die es bei der Geschäftsprozessmodellierung vorrangig geht.

Sei $\mathbf{N} = (S, T, F)$ ein Workflow-Netz mit einer ausgezeichneten Transition $t \in T$, die nur eine Eingangsstelle und nur eine Ausgangsstelle besitzt[35]. Sei ferner $\ddot{\mathbf{N}} = (\ddot{S}, \ddot{T}, \ddot{F})$ ein Workflow-Netz mit Quelle i und Senke o. Dann ist das Teilnetz $\dot{\mathbf{N}} = (\dot{S}, \dot{T}, \dot{F}) = (\ddot{S} \setminus \{i,o\}, \ddot{T}, \ddot{F} \setminus \{(k,l) \in \ddot{F} \mid k = i \vee l = o\})$ transitionsberandet.

Das resultierende Netz $\check{\mathbf{N}} = (\check{S}, \check{T}, \check{F})$, welches durch Ersetzung von t durch $\dot{\mathbf{N}}$ entsteht, gemäß:

$$\begin{aligned}
\check{S} &= S \cup \dot{S} \\
\check{T} &= (T \setminus \{t\}) \cup \dot{T} \\
\check{F} &= (F \setminus \{(k,l) \in F \mid k = t \vee l = t\}) \cup \\
&\quad \{(k,l) \in ((\bullet t \times R_{\dot{\mathbf{N}},\ddot{\mathbf{N}}}) \cup (R_{\dot{\mathbf{N}},\ddot{\mathbf{N}}} \times t\bullet) \mid (k \in \bullet t \wedge l \in i\bullet) \vee \\
&\quad (k \in \bullet o \wedge l \in t\bullet))\} \cup \dot{F}
\end{aligned}$$

[35] Sind mehrere Eingangs- resp. Ausgangsstellen zugelassen, kann es u.U. zu einem unerwünschten Verhalten des für t eingesetzten Subprozesses kommen. Insbesondere kann im Allgemeinen die Sicherheit des resultierenden Netzes nicht gewährleistet werden.

heißt **Transitions-Verfeinerung** von $\mathbf{N}$.

Im Hinblick auf die Soundness stellt sich noch die Frage, ob eine Transitions-Verfeinerung ebenfalls sound ist. Es lässt sich zeigen, dass gilt:

> $(\check{\mathbf{N}}, \iota_{\check{\mathbf{N}}})$ist sound und safe genau dann, wenn $(\mathbf{N}, \iota_{\mathbf{N}})$ und $(\ddot{\mathbf{N}}, \iota_{\ddot{\mathbf{N}}})$ sound und safe sind.

Diese Aussage hat van der Aalst für den allgemeinen Fall bewiesen (vgl. [Aals00, Theorem 3, Punkt 4. und Beweis]).

Anhang C: Gesetz von Little

Hier wird in Anlehnung an [FrJo97] kurz der Zusammenhang zwischen den verschiedenen Perspektiven der quantitativen Messgrößen, der durch Little's Gesetz hergestellt wird, dargestellt. Zunächst werden die einzelnen Perspektiven mit ihren charakteristischen Größen aufgeführt.

- *Kundenperspektive*:
  - Anzahl der Kunden: $n > 1$
  - Kunden: $c_1, c_2, ..., c_n$
  - Antwortzeit: $\alpha_{c_i}\ \forall i \in \{1, 2, ..., n\}$
  - Bearbeitungszeit: $\beta_{c_i}\ \forall i$
  - Wartezeit: $\omega_{c_i}\ \forall i$
  - Servicelevel: $\sigma_{c_i} = \frac{\beta_{c_i}}{\delta}\ \forall i$
  - Zusammenhang: ➤ $\alpha_{c_i} = \beta_{c_i} + \omega_{c_i}\ \forall i$
- *Ressourcenperspektive*:
  - Anzahl der Ressourcen: $m > 1$
  - Ressourcen: $r_1, r_2, ..., r_m$
  - Arbeitszeit: $\alpha_{r_j}\ \forall j \in \{1, 2, ..., m\}$
  - Bedienzeit: $\beta_{r_j}\ \forall j$
  - Wartezeit: $\omega_{r_j}\ \forall j$
  - Auslastungsgrad: $\upsilon_{r_j} = \frac{\beta_{r_j}}{\delta}\ \forall j$
  - Zusammenhang: ➤ $\alpha_{r_j} = \beta_{r_j} + \omega_{r_j}\ \forall j$
- *Produktperspektive*:
  Zwischen Kunden- und Ressourcenperspektive existiert ein enger Zusammenhang, da die einzelnen Aktivitäten eines Kundenauftrags von verschiedenen Ressourcen bearbeitet werden. Wird die Zeit für die Bearbeitung einer Aktivität des Kunden c_i durch Ressource r_j mit $\beta_{r_j}(c_i)$ bezeichnet, dann lassen sich alle diese Werte in einer Matrix darstellen:

	c_1	c_2	$\cdots$	c_n	
r_1	$\beta_{r_1}(c_1)$	$\beta_{r_1}(c_2)$	$\cdots$	$\beta_{r_1}(c_n)$	β_{r_1}
r_2	$\beta_{r_2}(c_1)$	$\beta_{r_2}(c_2)$	$\cdots$	$\beta_{r_2}(c_n)$	β_{r_2}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
r_m	$\beta_{r_m}(c_1)$	$\beta_{r_m}(c_2)$	$\cdots$	$\beta_{r_m}(c_n)$	β_{r_m}
	β_{c_1}	β_{c_2}	$\cdots$	β_{c_n}	

  - Zusammenhang:
    - ➤ $\beta_{c_i} = \sum_{j=1}^{n} \beta_{r_j}(c_i)\ \forall i$
    - ➤ $\beta_{r_j} = \sum_{i=1}^{n} \beta_{r_j}(c_i)\ \forall j$

- *Prozessperspektive*:
  - Durchlaufzeit:

$$\begin{aligned} \delta &= max\{\alpha_{c_i} \mid i \in \{1,2,...,n\}\} \\ &= max\{\alpha_{r_j} \mid j \in \{1,2,...,m\}\} \end{aligned}$$

- *Systemperspektive*:
  - Durchsatz: $\tau = \frac{n}{\delta}$

Die folgende Abbildung stellt das o.g. nochmals graphisch dar.

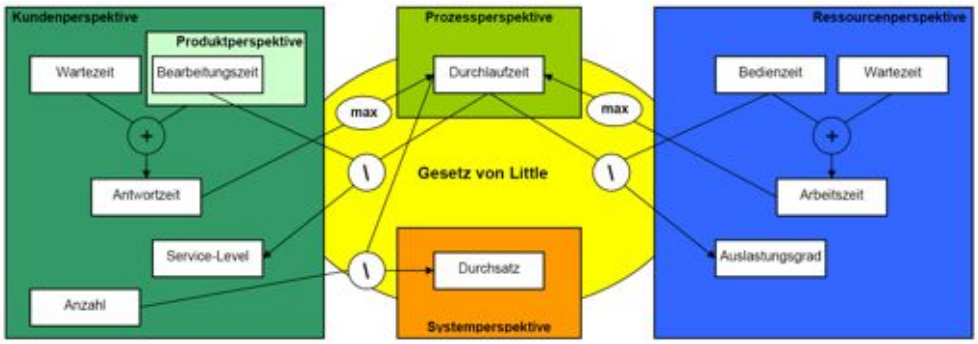


Abbildung 26: Gesetz von Little

Anhang D: Methoden zur Netzanalyse

Im Abschnitt 5.1 wurde bereits angedeutet, dass die Workflow-Eigenschaft eines Petri-Netzes allein zu abstrakt ist, um darauf basierend einen einfachen Algorithmus zu implementieren, der in einem Durchlauf durch das Netz die Ausführungshäufigkeiten berechnet. Verschiedene Möglichkeiten wurden untersucht, sind aber jeweils an einigermaßen komplizierten Beispielnetzen gescheitert. Die Idee der jeweiligen Methode und der Grund des Scheiterns werden nachfolgend skizziert:

Zyklensuche

Das Problem bei der Berechnung entsteht durch die Zyklen. Daher war eine erste Idee, die Zyklen im Netz zu finden. Wenn dann der Eintrittsknoten[36] und der *zugehörige* Verzweigungsknoten bekannt sind, kann die entsprechende Verzweigungswahrscheinlichkeit bis zu der Kante "vorgeschoben" werden, die in den Eintrittsknoten mündet. Dann kann die Ausführungshäufigkeit gemäß Kapitel 3, Schritt 1(e) berechnet werden.

Untersucht wurde dabei auch der Algorithmus von Tarjan zum Auffinden von starken Zusammenhangskomponenten[37]. Dieser eignet sich aber nicht, da er einer Maximalbedingung in dem Sinne genügt, dass er maximal große Zusammenhangskomponenten findet. Diese können dann mehr als eine Schleife enthalten. Eine geringe Abwandlung der Tiefensuche reicht grundsätzlich aber auch aus, um Zyklen zu finden.

Es gibt aber Workflow-Netze, bei denen diese Strategie nicht aufgeht. Verschlungene oder verschachtelte Schleifen lassen sich damit nicht analysieren. Die benötigte Information ist dann nur durch simultanes Lösen mehrerer Gleichungen möglich. Drei Beispiele solcher Netze sind in der nachfolgenden Abbildung zu sehen.

[36] Das ist der Knoten, bei dem man beim Durchlaufen des Graphen landet und der noch weitere Eingangskanten besitzt.

[37] Vgl. R. TARJAN. Depth-first search and linear graph algorithms. SIAM J. Comput., vol 1, S. 146-160, 1972.

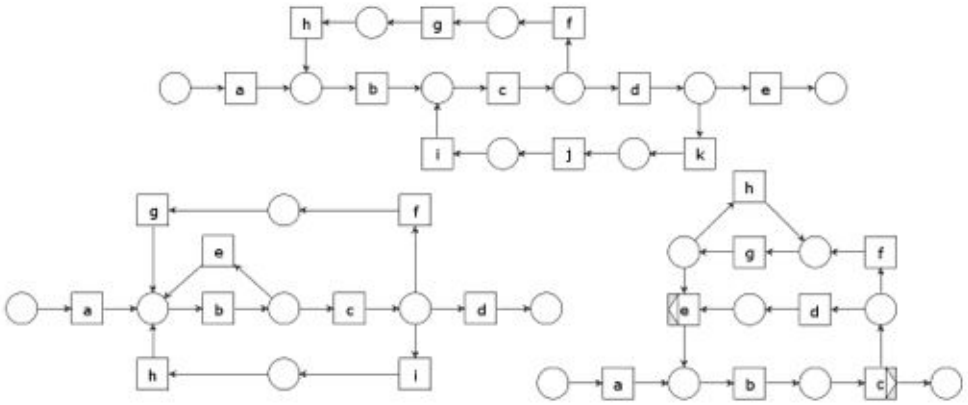


Abbildung 27: Schwer analysierbare Netze

Reguläre Ausdrücke

Eine weitere Strategie in Anlehnung an die ”design patterns” von van der Aalst (vgl. [AaHe02, S. 129]) besteht darin, ein Workflow-Netz formal wie einen regulären Ausdruck zu behandeln. Der Sequenz von *a* und *b* entspricht dann $\mathbf{ab}$, dem XOR $\mathbf{a} + \mathbf{b}$ und der Schleife, bei der *a* mindestens einmal und *b* wiederholt ausgeführt wird, der reguläre Ausdruck $\mathbf{a}(\mathbf{ba})^*$. Die Zyklensuche wäre dann trivial.

Für die drei Netze ergibt sich beispielsweise:

- (oben) $\mathbf{abc}((\mathbf{fghbc})^*\mathbf{d} + \mathbf{d}(\mathbf{kjicd})^*)^+\mathbf{e}$,
- (unten links) $\mathbf{ab}(\mathbf{eb})^*\mathbf{c}((\mathbf{fgb}(\mathbf{eb})^*)^* + (\mathbf{ihb}(\mathbf{eb})^*)^*)\mathbf{d}$ und
- (unten rechts) $\mathbf{abc}((\mathbf{debc})^* + (\mathbf{fg}(\mathbf{hg})^*\mathbf{ebc})^*)^*$.

Parallelität läßt sich damit aber nicht darstellen. Außerdem kommt ein erhöhter Aufwand für die Übersetzung in den regulären Ausdruck hinzu und zu guter letzt besteht dieselbe Schwierigkeit, die im nächsten Punkt erörtert wird.

Schrittweise Verfeinerung

Ähnlich wie bei regulären Ausdrücken besteht diese Idee darin, einfach zu analysierende Teilnetze, durch Transitionen zu ersetzen. Das entspricht Transitions-Vergröberungen. Damit würde ein hierarchisch organisierter schichtenweiser Blick auf das Netz entstehen. Auf der obersten Ebene gäbe es dann nur ein Basis-Netz mit Quelle *i*, einer Transition *t* und Senke *o*. Wenn dann für die Quelle gilt

$\phi(i) = \lambda$, dann gilt auch $\phi(t) = \phi(o) = \phi(i) = \lambda$. Jetzt kann *t* durch das Teilnetz der nächsten Ebene ersetzt werden, gemäß der Definition der Transitions-Verfeinerung.

In dem eingefügten Teilnetz werden für alle Transitionen die Ausführungshäufigkeiten bestimmt. Danach können weitere Transitionen verfeinert werden, bis das ursprüngliche Netz wieder hergestellt ist. Dann ist das Netz vollständig analysiert.

Leider funktioniert dieser Ansatz aber bei den drei Beispielnetzen nicht. Es gibt sogar ein Workflow-Netz mit "Überkreuzungen" (vgl. [AaHe02, S. 116, Abb. oben]). Für so ein Netz scheitert diese Strategie von vorn herein.

Ein weiterer Versuch wäre, ausgehend vom Basis-Netz ein Netz durch eine Folge von Ersetzungen von Transitionen durch die "design patterns" darzustellen, wie es in [AaHe02, S. 110-118] beschrieben ist. Für diese Musternetze können fertige Lösungen angewendet werden, in denen nur der Parameter für die jeweilige Ankunftsrate eingegeben werden muss. Dann läßt sich auch das Netz mit Überkreuzungen als für sich allein analysiertes Muster verwenden.

Das Problem bei diesem Vorschlag ist aber, dass man erst beweisen muss, dass die von van der Aalst angegebene Liste von "design patterns" vollständig ist, in dem Sinne, dass sich damit jedes Workflow-Netz, das sound und safe ist, aufbauen läßt. Dieser Beweis steht noch aus.

Aufgrund all dieser Schwierigkeiten ist letztlich die Entscheidung zugunsten der Implementierung einer Näherungslösung gefallen. Sie wird im Abschnitt 5.1 ab Seite 61 vorgestellt.

Anhang E: Entfaltungsalgorithmus

Der Algorithmus zur Netzentfaltung bekommt eine Instanz des Netzgraphen und die Werte für die Ankunftsrate λ und die Abbruchschranke ε übergeben. Das entfaltete Netz wird in der *globalen* Variablen `unfoldedNet` gespeichert. Es ist als *HashMap* implementiert, um die Geschwindigkeit beim Bilden der Summen der Ausführungshäufigkeiten pro (Original-) Knoten zu erhöhen.

Zunächst erfolgt der Aufruf der Methode `unfoldNet()`, die die Initialisierung vornimmt und dann mit der rekursiven Methode `runThroughNet()` die Entfaltung startet.

```
1  private void unfoldNet(WorkflowNetGraph g, double l,
       double e) {
2         unfoldedNet.clear();
3         for (Node n : g.getNodeArray()) {
4               if (n.isJoinReached())
5                     n.setJoinReached(false);

7               n.setNumOfRuns(0);
8         }

10        Node s = g.getStartPlace();
11        Node n = new Node(s.getId(), s.getName());
12        n.setTempRuns(lambda);
13        Key start = new Key(n.getId(), lambda);
14        unfoldedNet.put(start, n);

16        LinkedList<NodePair> queue = new LinkedList<NodePair
             >();
17        queue.addLast(new NodePair(s, n));
18        runThroughNet(queue);
19 }
```

```
1  // rekursive Funktion mit Breitensuche
2  private void runThroughNet(LinkedList<NodePair> q) {
3         if (!(q.isEmpty())) {
4               NodePair np = q.removeFirst();

6               for (Arc a : np.getFirst().getSuccessor()) {
7                     Node m = a.getTarget();
8                     if (!m.isJoinReached()) {
9                           double val = a.getProbability()
```

```
10                                          * np.getSecond().
                                              getTempRuns();
11                              if (!(val < epsilon)) {
12                                  Node y = new Node(m.getId(),
                                        m.getName());
13                                  y.setTempRuns(val);
14                                  if (m.isAndJoin())
15                                      m.setJoinReached(true)
                                            ;
16                                  np.getSecond().getSuccessor
                                        ().add(
17                                              new Arc(y, a.
                                                  getProbability
                                                  ()));
18                                  Key k = new Key(m.getId(),
                                        val);
19                                  unfoldedNet.put(k, y);

21                                  if (!(containsElement(q, m))
                                        )
22                                      q.addLast(new NodePair
                                          (m, y));
23                              }
24                          }
25                  }

27                  runThroughNet(q);
28          }
29  }
```

Erläuterungen zur Methode `unfoldNet()`:

Zeilen 2 bis 8: Initialisierung der Datenstruktur `unfoldedNet`, Flags werden gelöscht.

Zeilen 10 bis 12: Erstellen einer Kopie der Startstelle und Initialisierung der Häufigkeit mit λ. Das Kopieren von Knoten ist notwendig, da sonst bei erneutem Durchlaufen eines Pfades im Originalnetz die letzte Häufigkeit gelöscht würde.
Das einfache Aufsummieren beim Durchlaufen des Originalgraphen scheitert an den AND-Joins innerhalb von Schleifen. Da nur der Wert von einem Pfad summiert werden darf

und alle anderen parallelen Pfade des entsprechenden AND-Splits vor dem Join enden müssen, wird ein Flag gesetzt, dass die erfolgte einmalige Summation anzeigt. Beim Durchlauf durch die Schleife würde eine erneute Ankunft beim Join gar nicht mehr gezählt und der Pfad abgebrochen werden, was zu einem Fehler führt.

Zeile 13: Definition einer neuen `Key`-Instanz. `Key`-Objekte speichern die aktuelle Häufigkeit zusammen mit der Identifikationsnummer eines Knotens. Sie dienen als eineindeutige Schlüssel zur Identifizierung einer Knotenkopie in der HashMap `unfoldedNet`.

Zeile 14: Das Paar (<Schlüssel>, <Kopie>) für den Startknoten wird in `unfoldedNet` gespeichert.

Zeile 16: Für die Breitensuche wird die Warteschlange `queue` benötigt. Sie wird hier als verkettete Liste implementiert und initialisiert. Ihre Elemente sind vom Typ `NodePair`. Diese Objekte stellen die Beziehung zwischen Original und Kopie her.

Zeile 17: Das erste Knotenpaar wird in die Warteschlange eingefügt.

Zeile 18: Aufruf von `runThroughNet()` mit `queue` als Parameter.

Erläuterungen zur Methode `runThroughNet()`:

Zeilen 3 bis 4: Nur wenn mindestens ein Element in der übergebenen Warteschlange enthalten ist, wird die Methode ausgeführt. Das erste Knotenpaar wird aus der Warteschlange entnommen.

Zeilen 6 bis 7: Für alle Nachfolgerknoten des Originalknotens aus dem entnommenen Knotenpaar wird Folgendes ausgeführt: `m` bezeichne einen Nachfolger

Zeilen 8 bis 10: Wenn `m` keine AND-Join-Transition ist, die bereits einmal erreicht wurde, dann speichert `val` die aktuelle Häufigkeit von `m`, berechnet als das Produkt der Übergangswahrscheinlichkeit und der Häufigkeit der letzten Kopie des Vorgängers von `m`. Ist `m` eine AND-Join-Transition und wurde bereits erreicht, passiert nichts und der nächste Nachfolger wird behandelt oder die Schleife verlassen.

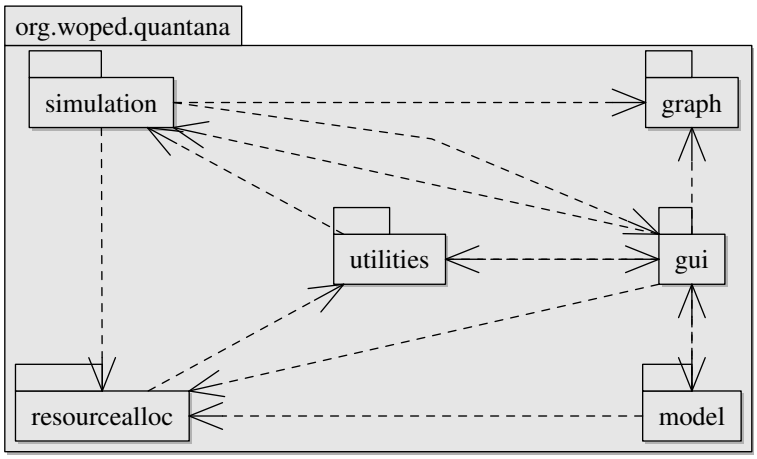
Zeilen 11 bis 19: Wenn `val` nicht kleiner ist als ε, dann wird eine Kopie `y` von `m` angelegt. Wenn `m` außerdem eine AND-Join-Transition ist, wird sie als erreicht markiert. Die Kopie aus dem aktuellen Knotenpaar erhält `y` als Nachfolger. Es wird ein neuer Schlüssel für `m` und `val` erzeugt und dieser zusammen mit `y` in `unfoldedNet` gespeichert.

Zeilen 21 bis 25: Wenn `m` nicht schon in der Warteschlange enthalten ist, wird das neue Knotenpaar (`m,y`) aufgenommen und nach Abarbeiten aller anderen Nachfolger die Schleife verlassen.

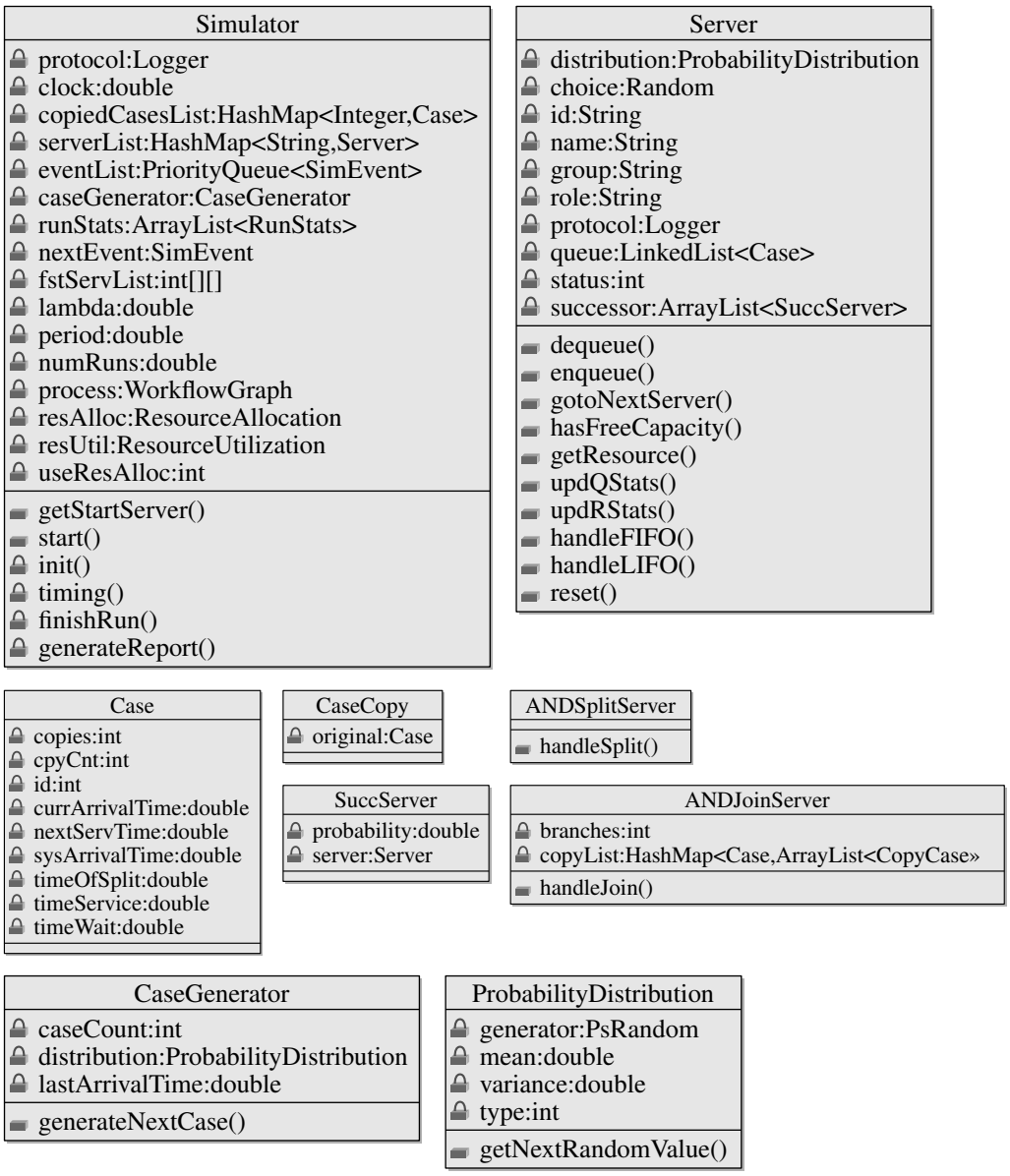
Zeile 27: Hier erfolgt der rekursive Aufruf der Methode. Ist die Warteschlange leer, endet der Algorithmus.

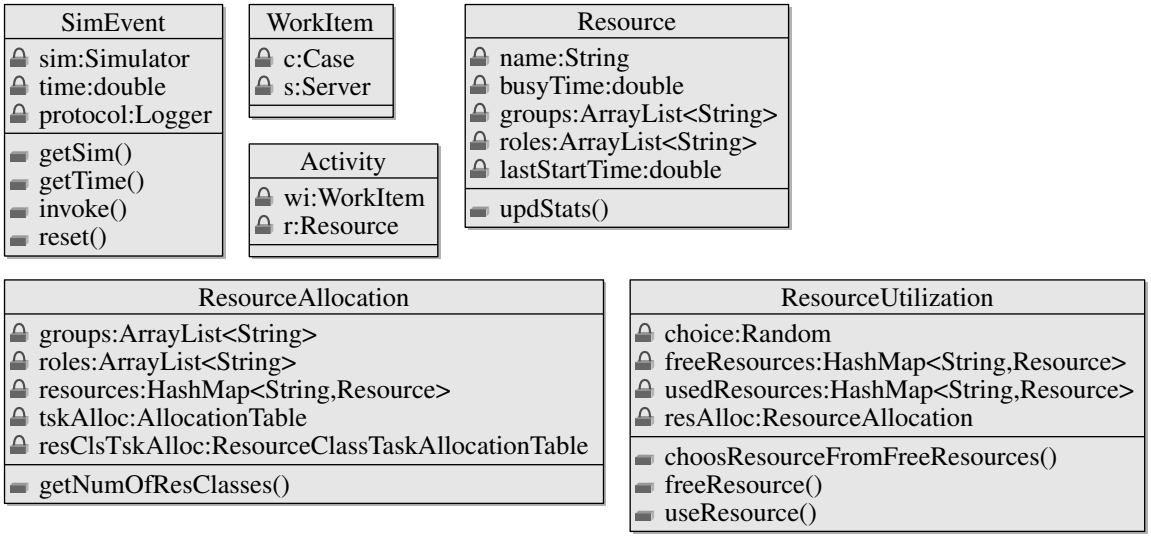
Anhang F: UML-Diagramme zur Simulation

Paketdiagramm von `org.woped.quantana`:

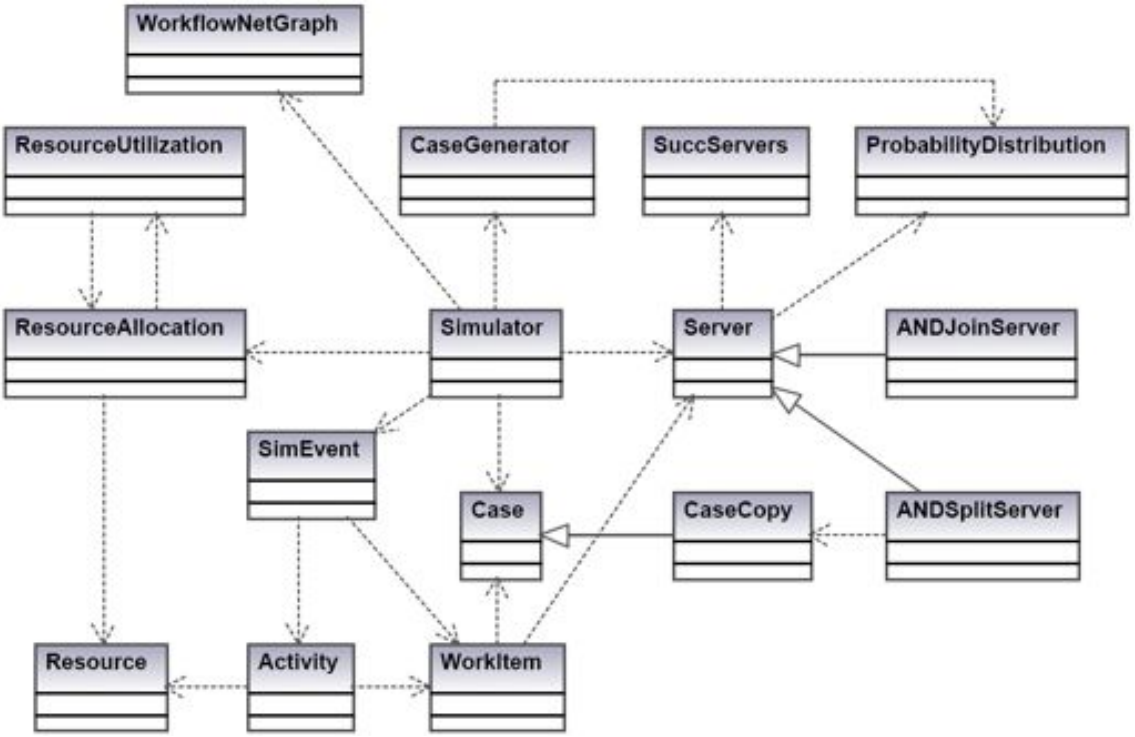


Die wichtigsten Klassen mit ihren Attributen und Methoden sind:





Klassendiagramm (Ausschnitt):

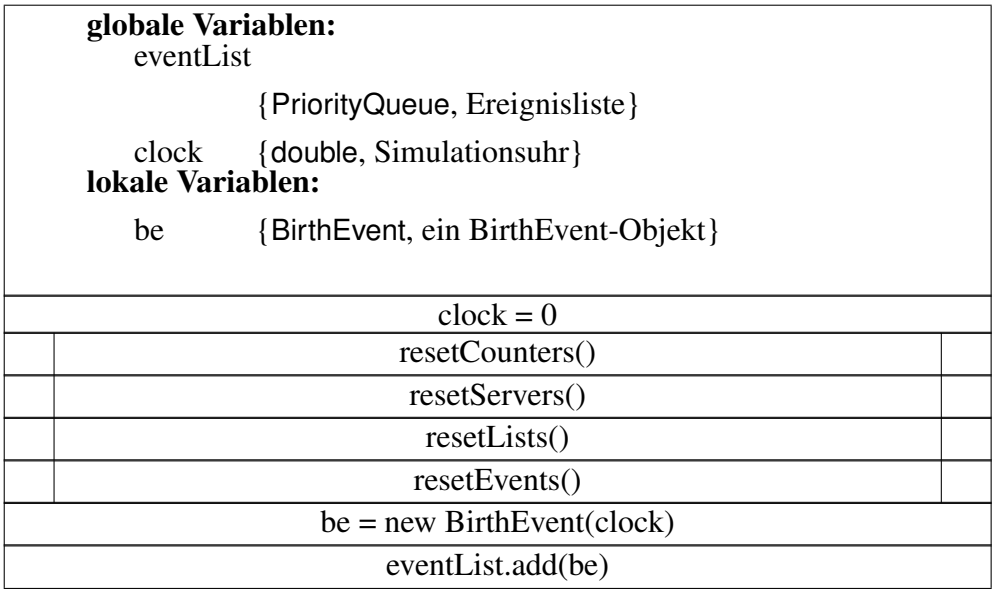


Anhang G: Struktogramme zur Simulation

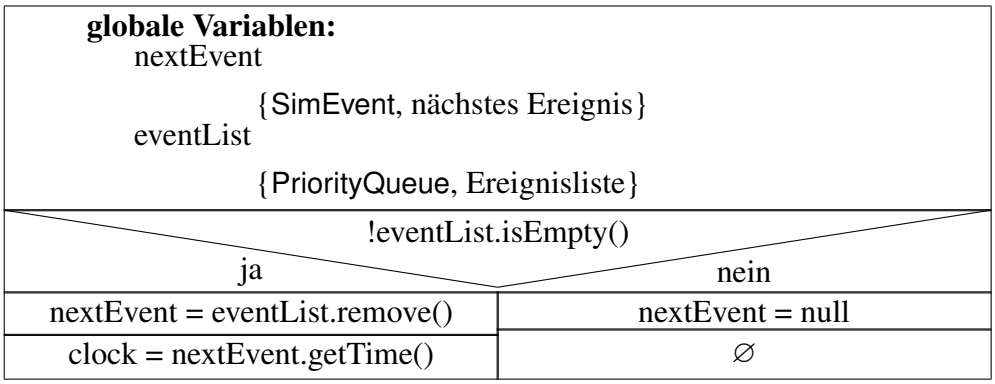
Auf den folgenden Seiten sind die wesentlichsten Methoden aus dem Simulationspaket aufgeführt. Die Bezeichnungen weiterer Methoden oder von Variablen sollten selbsterklärend sein. Die Verwendung von Struktogrammen zur Darstellung von Java-Code wurde aus Gründen der besseren Übersicht erwogen.

Nach der Darstellung der Methoden `init()` und `timing()` werden die `invoke()`-Methoden der Simulationsereignisse vorgestellt.

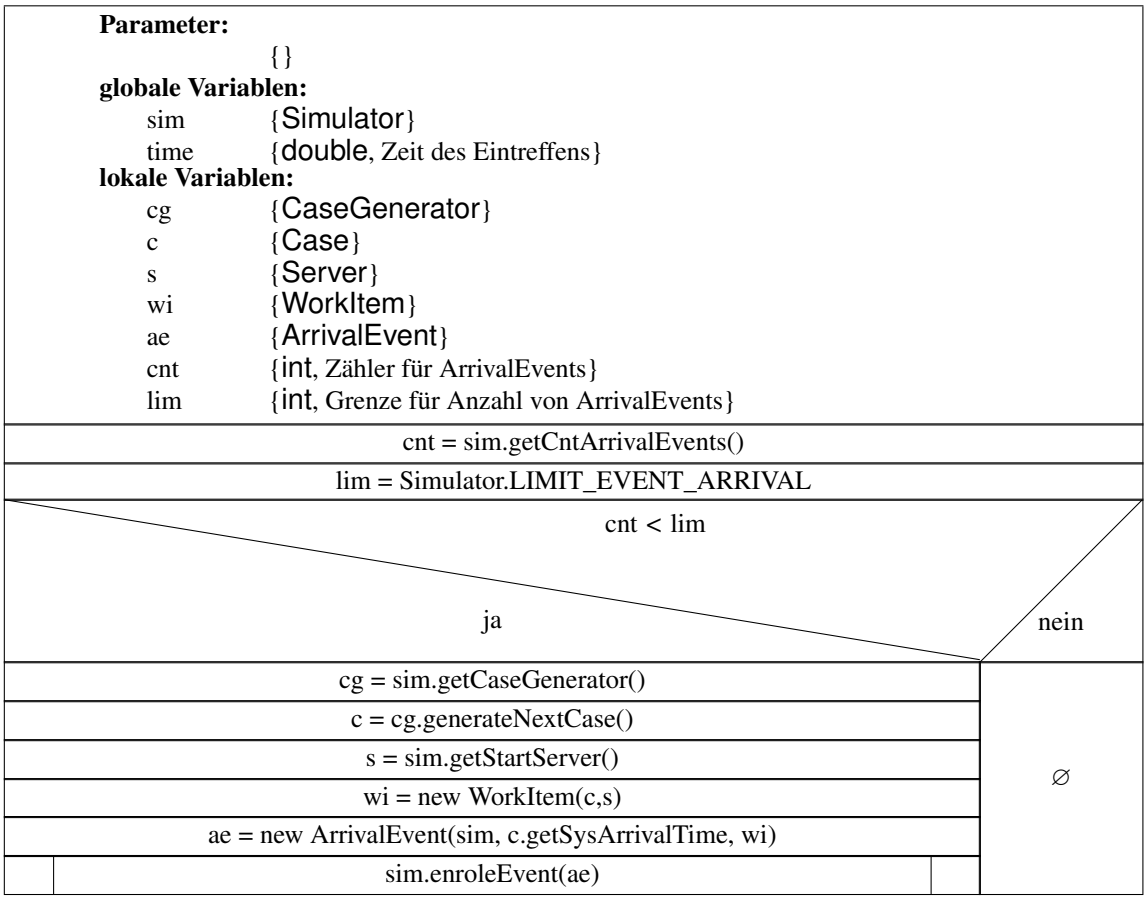
init():



timing():



BirthEvent.invoke():



ArrivalEvent.invoke():

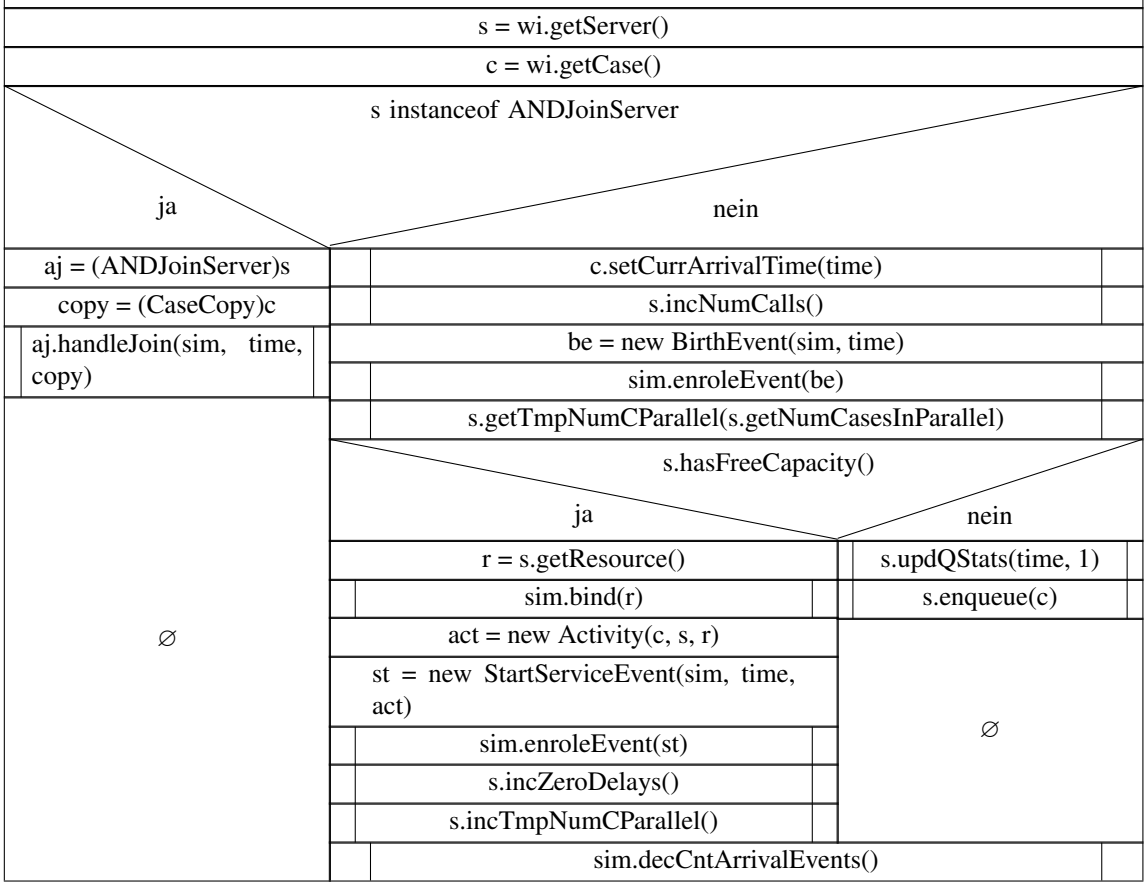
Parameter:
{}

globale Variablen:

sim	{Simulator}
time	{double, Zeit des Eintreffens}
wi	{WorkItem}

lokale Variablen:

s	{Server}
c	{Case}
be	{BirthEvent}
r	{Resource}
act	{Activity}
st	{StartServiceEvent}
aj	{ANDJoinServer}
copy	{CaseCopy}



StartServiceEvent.invoke():

Parameter:
{}

globale Variablen:

sim	{Simulator}
time	{double, Zeit des Eintreffens}
act	{Activity}

lokale Variablen:

s	{Server}
c	{Case}
r	{Resource}
sp	{StopServiceEvent}
depart	{double}
wait	{double}
serv	{double}

s = act.getServer()

c = act.getCase()

r = act.getResource()

s.setStatus(Server.STATUS_BUSY)

s.updRStats(time, 1)

wait = time - c.getCurrArrivalTime()

s.incWaitTime(wait)

s.getWaitTimes().add(new Double(wait))

wait > s.getMaxWaitTime()

ja	nein
s.setMaxWaitTime(wait)	∅

c.addWaitTime(wait)

r != null

ja	nein
r.setLastStartTime(time)	∅

serv = s.getNextServTime()

depart = time + serv

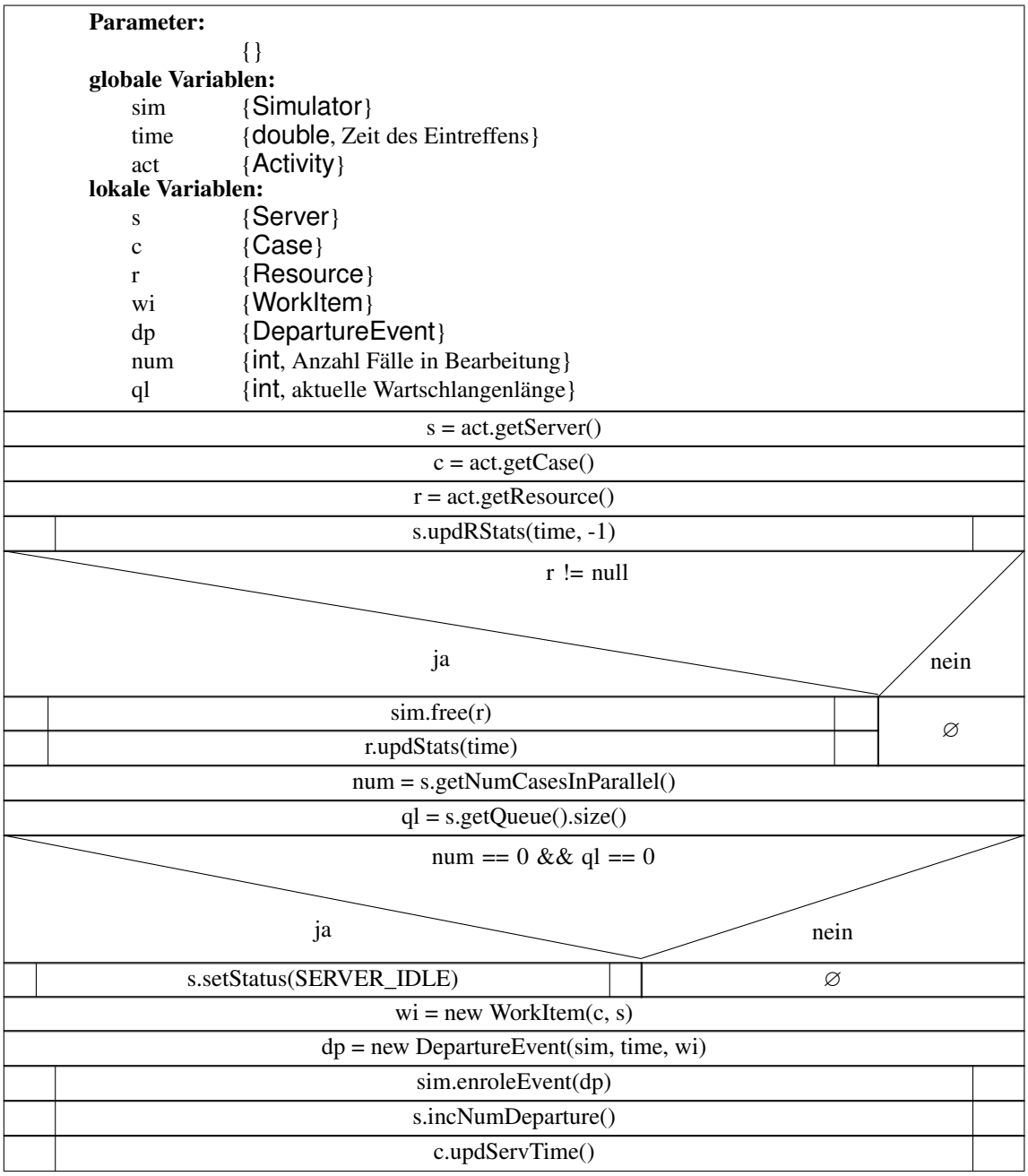
c.setNextServTime(serv)

sp = new StopServiceEvent(sim, depart, act)

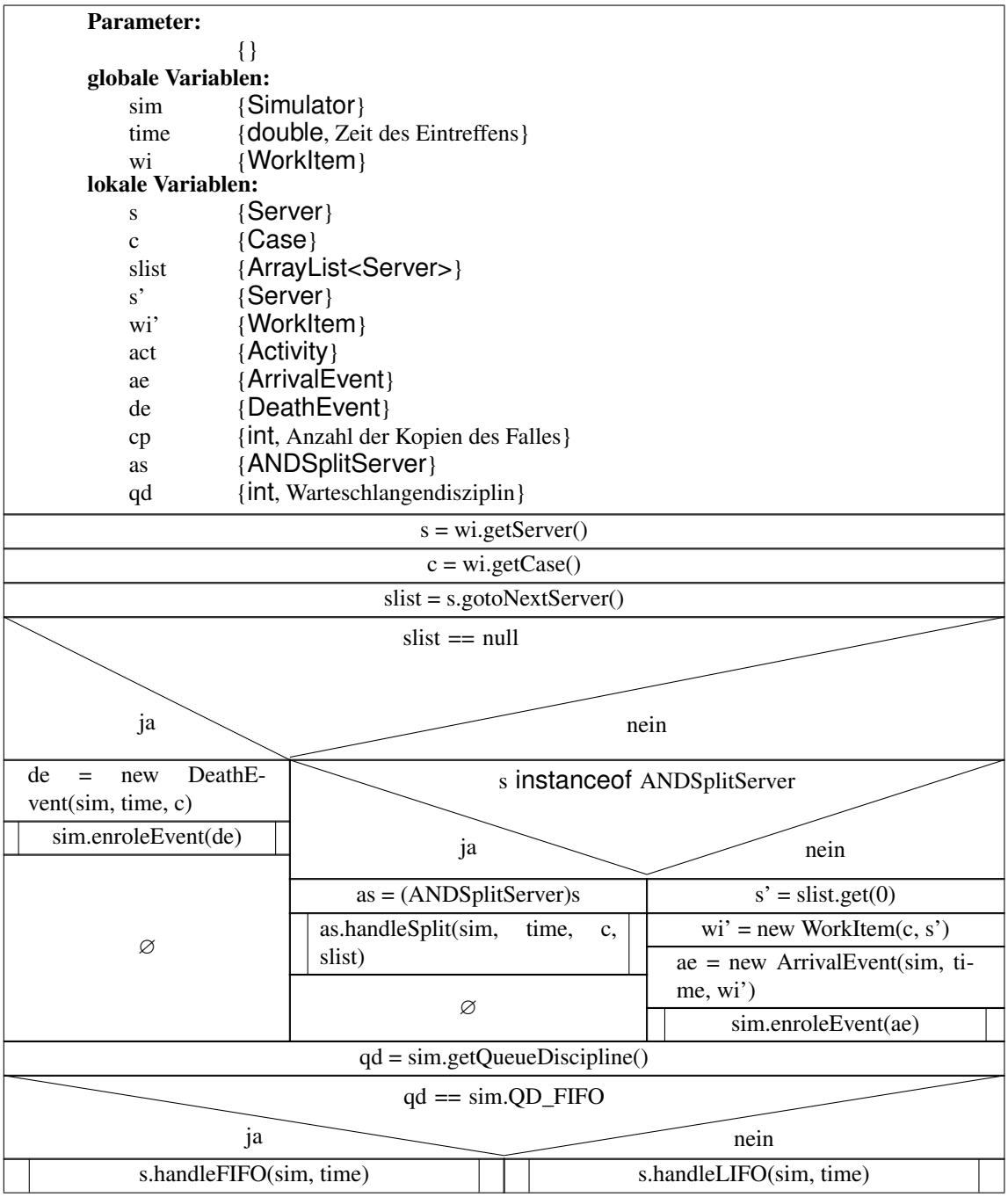
sim.enroleEvent(sp)

s.incNumAccess()

StopServiceEvent.invoke():



DepartureEvent.invoke():



DeathEvent.invoke():

Parameter:	
	{}
globale Variablen:	
sim	{Simulator}
time	{double, Zeit des Eintreffens}
c	{Case}
lokale Variablen:	
cTime	{double, Prozess-Durchlaufzeit}
sim.incAvgProcessWaitTime(c.getTimeWait())	
sim.incAvgProcessServiceTime(c.getTimeService())	
cTime = time - c.getSysArrivalTime()	
sim.incAvgProcessCompletionTime(cTime)	
sim.incFinishedCases()	

Es folgen die Struktogramme einiger Hilfsmethoden, die von den Ereignissen verwendet werden und besondere Bedeutung für die Ablauflogik der Simulation haben:

- generateNextCase(): erzeugt neue Fälle mit Ankunftsrate λ
- getStartServer(): wählt aus der Menge möglicher Anfangsserver einen aus
- getResource(): wählt aus der Menge freier Resourcen eine aus
- updQStats(): führt die Warteschlangenstatistik pro Server
- updRStats(): führt Statistik über die Server-Auslastung
- handleSplit(): erzeugt Kopien eines Falles bei einem AND-Split
- handleJoin(): führt Kopien eines Falles bei einem AND-Join zusammen
- handleFIFO(): holt wartende Fälle aus der Warteschlange nach FIFO-Prinzip
- handleLIFO(): dto., aber nach LIFO-Prizip
- gotoNextServer(): bestimmt den nachfolgenden Server
- hasFreeCapacity(): prüft, ob der Server einen weiteren Fall bearbeiten kann

CaseGenerator.generateNextCase():

Parameter:
{}

globale Variablen:

caseCount {int}

lastArrivalTime {double}

distribution {ProbabilityDistribution}

lokale Variablen:

next {Case}

time {double}

next = new Case(++caseCount)

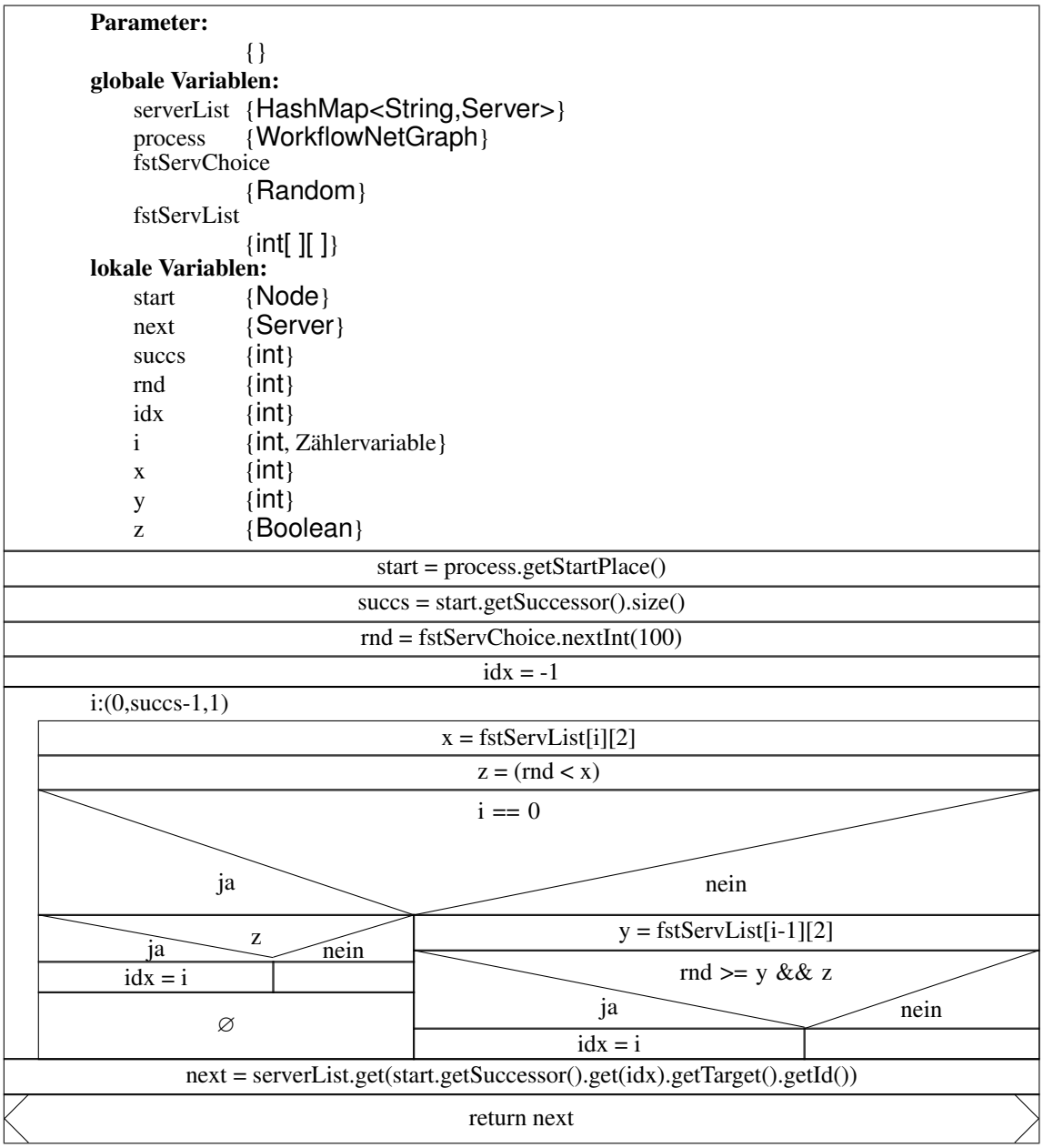
time = distribution.getNextRandomValue()

lastArrivalTime += time

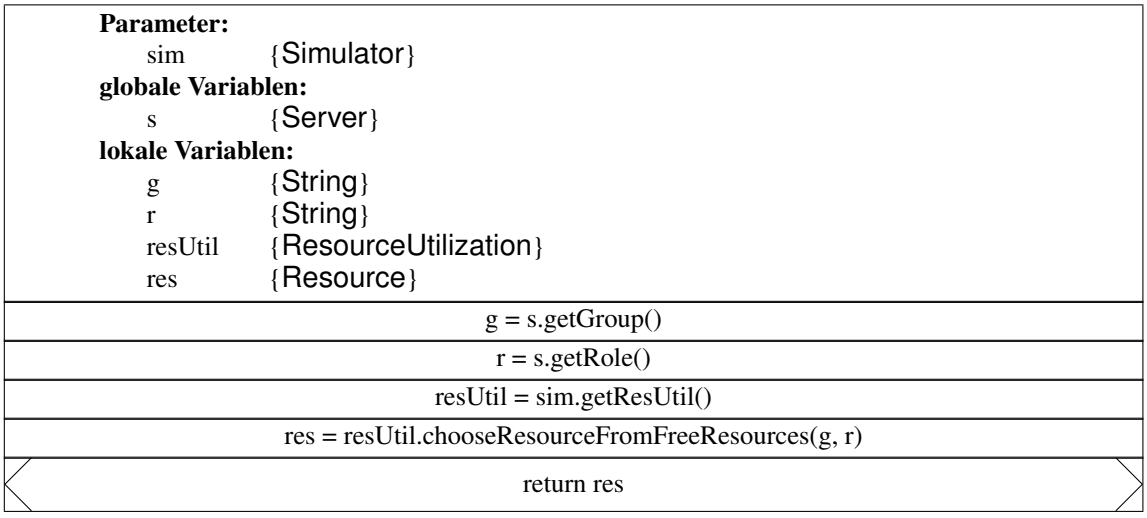
next.setSysArrivalTime(lastArrivalTime)

return next

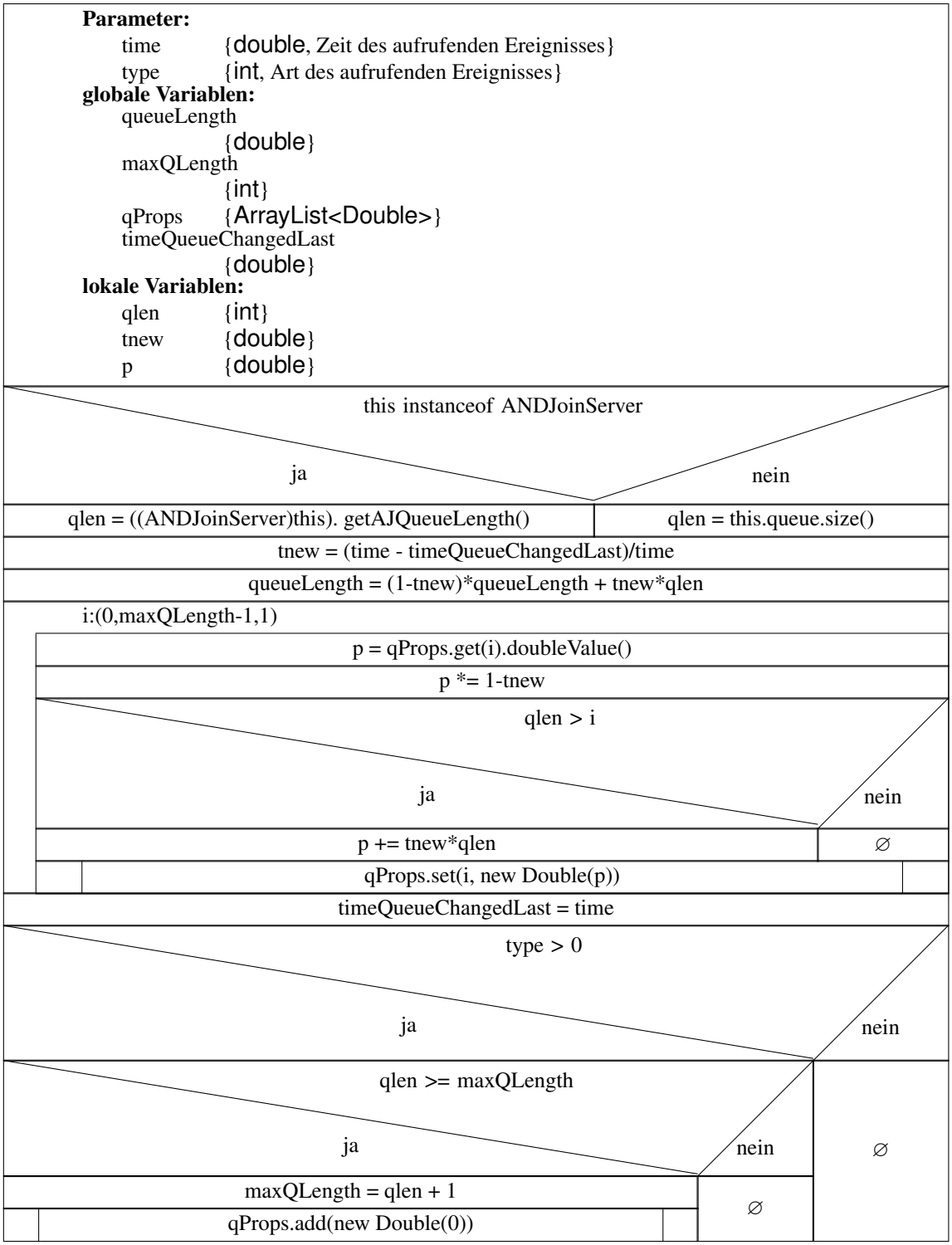
Simulator.getStartServer():



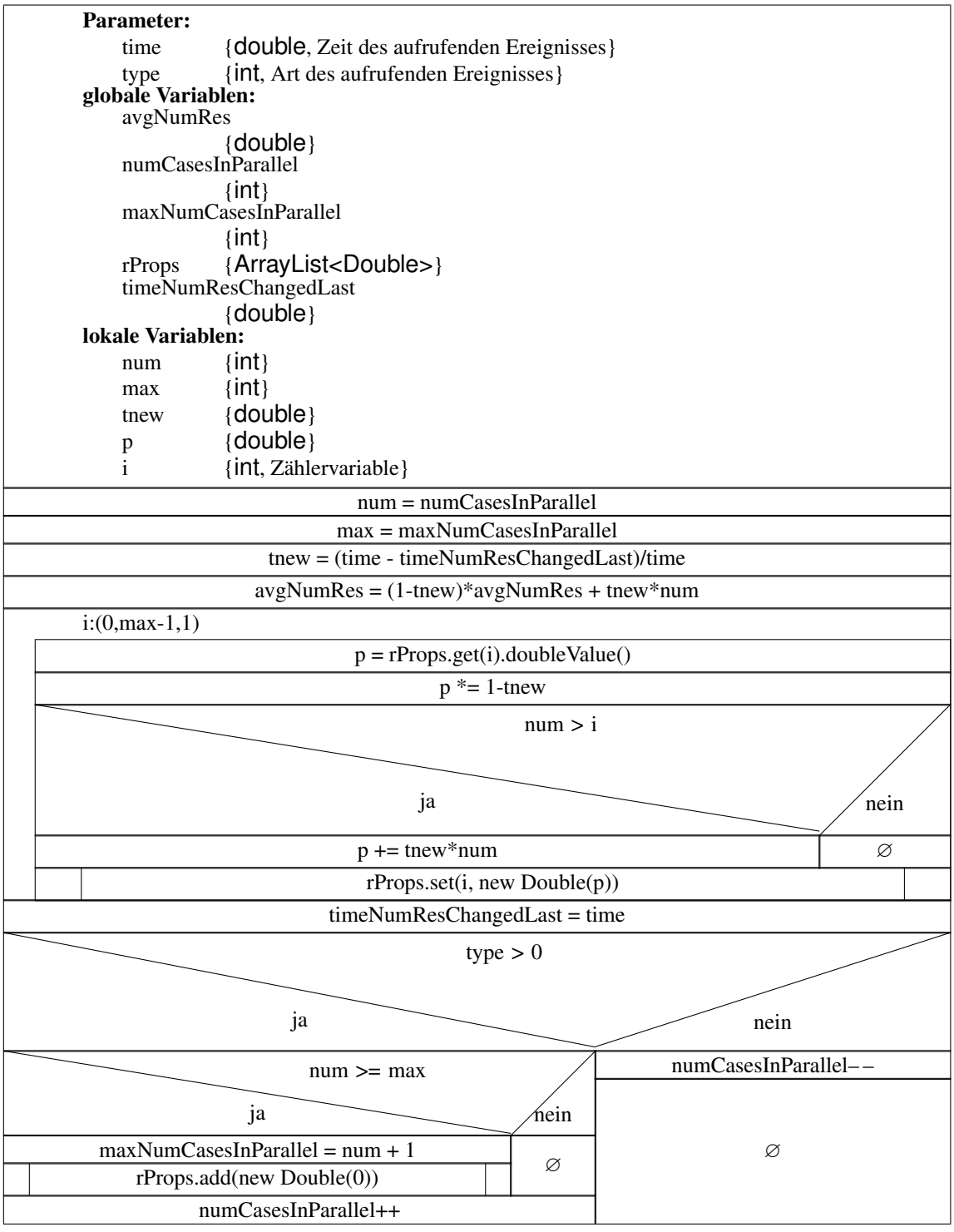
Server.getResource():



Server.updQStats():



Server.updRStats():



ANDSplitServer.handleSplit():

Parameter:

sim	{Simulator}
time	{time}
c	{Case}
l	{ArrayList<Server>}

globale Variablen:

s	{ANDSplitServer}

lokale Variablen:

cg	{CaseGenerator}
s'	{Server}
cc	{Case}
ae	{ArrivalEvent}
copies	{int}
cnt	{int}
i	{int, Zählervariable}

sim.addToCopyList(c)

copies = l.size()

c.setCopies(copies)

c.setCpyCnt(0)

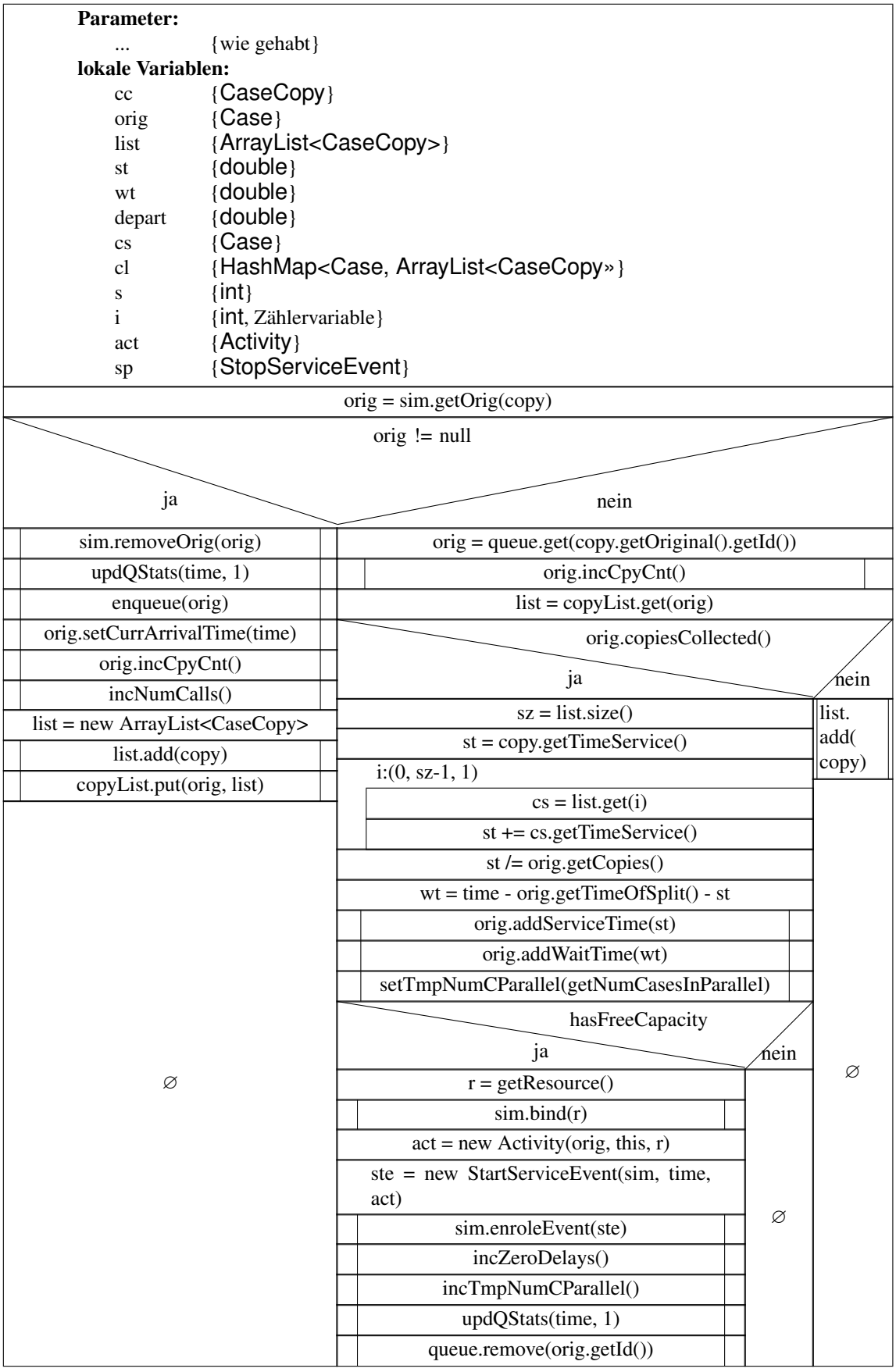
c.setTimeOfSplit(time)

cg = sim.getCaseGenerator()

i:(0,copies-1,1)

- s' = l.get(i)
- cnt = cg.getCaseCount() + 1
- cg.setCaseCount(cnt)
- cc = new CaseCopy(cnt, c)
- ae = new ArrivalEvent(sim, time, new WorkItem(c, s))
- sim.enroleEvent(ae)

ANDJoinServer.handleJoin():



Server.handleFIFO():

Parameter:

sim	{Simulator}
time	{time}

globale Variablen:

s	{Server}
tmpNumCParallel	{int}

lokale Variablen:

cap	{Boolean}
qe	{Boolean}
r	{Resource}
c	{Case}
act	{Activity}
st	{StartServiceElement}

tmpNumCParallel = s.getNumCasesInParallel()

cap = s.hasFreeCapacity()

qe = s.getQueue().isEmpty()

s.updQStats(time, -1)

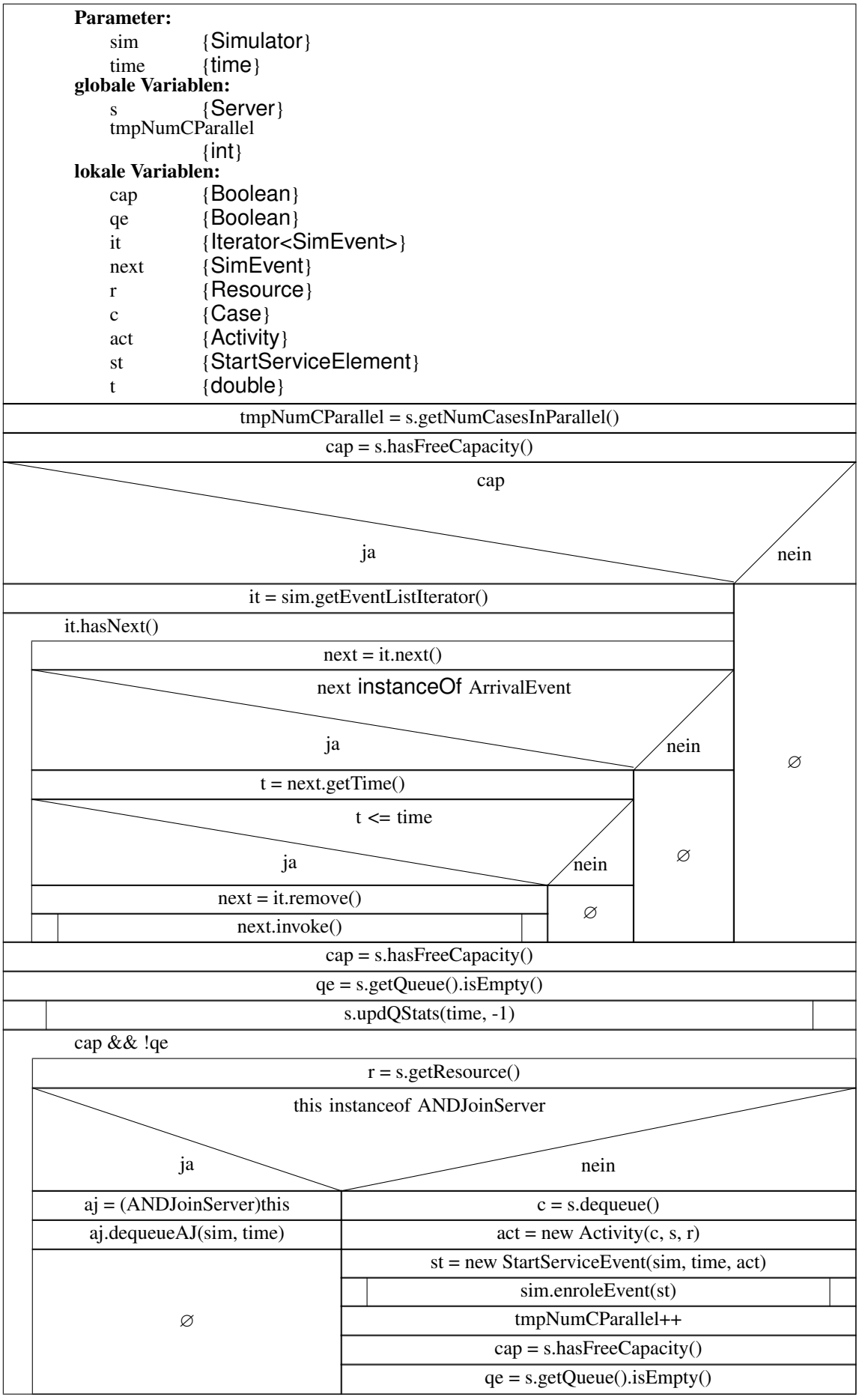
cap && !qe

r = s.getResource()

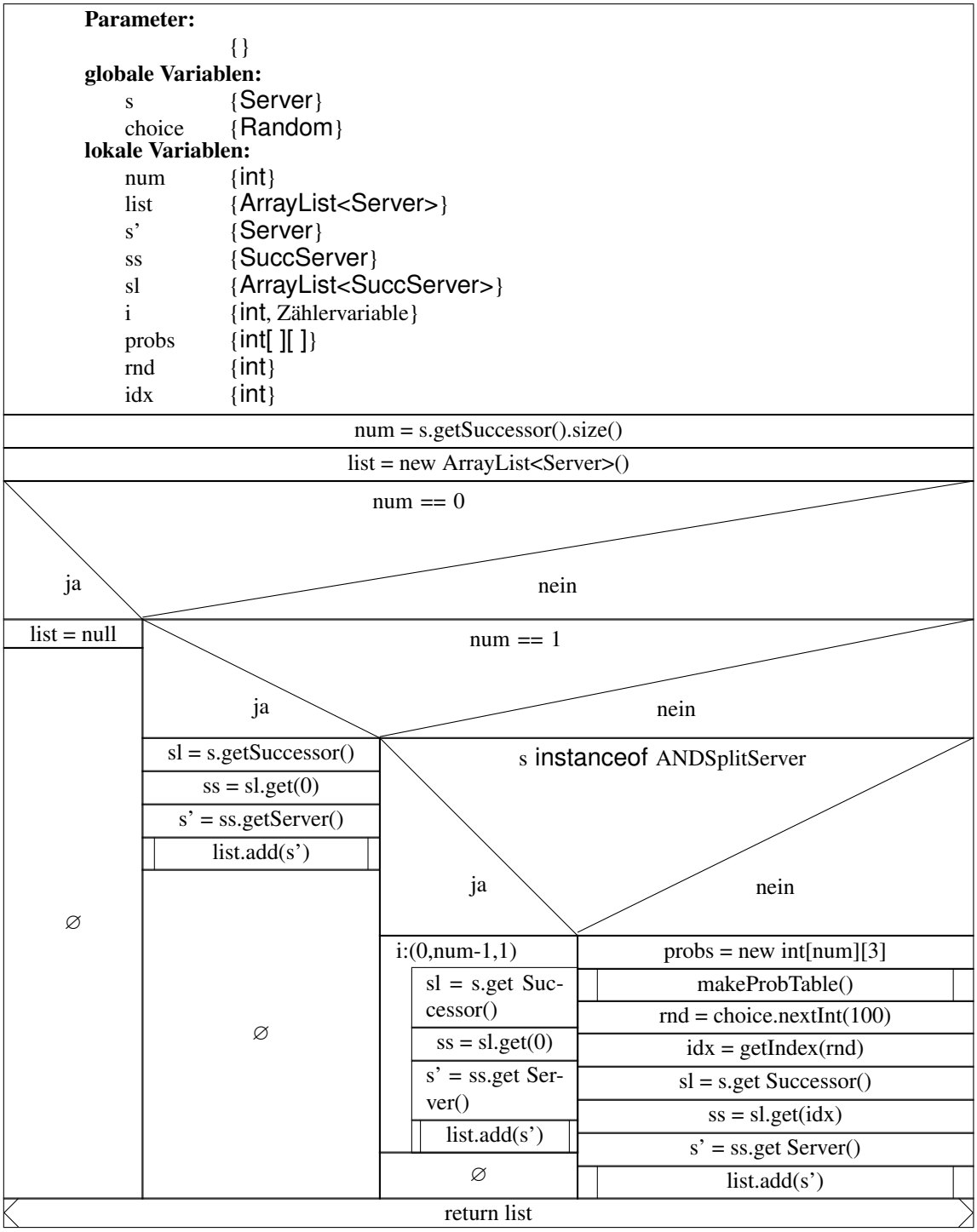
this instanceof ANDJoinServer

ja	nein
aj = (ANDJoinServer)this	c = s.dequeue()
aj.dequeueAJ(sim, time)	act = new Activity(c, s, r)
∅	st = new StartServiceEvent(sim, time, act)
	sim.enroleEvent(st)
	tmpNumCParallel++
	cap = s.hasFreeCapacity()
	qe = s.getQueue().isEmpty()

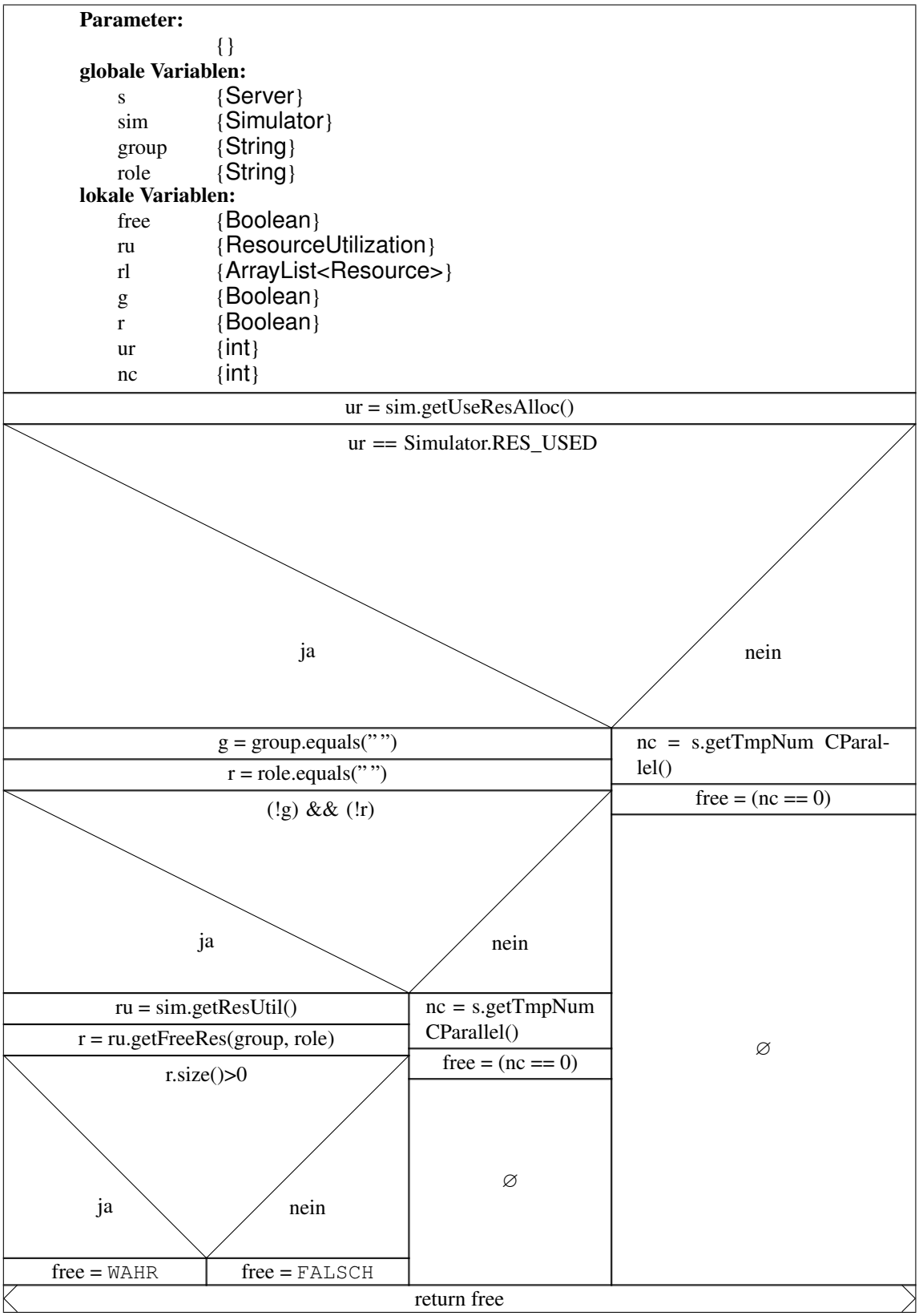
Server.handleLIFO():



Server.gotoNextServer():



Server.hasFreeCapacity():



Literatur

[AaDe02] W. M. P. VAN DER AALST, J. DESEL, E. KINDLER. *On the semantics of EPCs: A vicious circle*. In M. Nüttgens and F. J. Rump, ed.: Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), 71-79, Trier, Germany, 2002.

[AaHe02] W. M. P. VAN DER AALST, K. M. VAN HEE. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge (Mass.), London, 2002.

[AaHo98] W. M. P. VAN DER AALST AND A. H. M. TER HOFSTEDE. *Verification of Workflow Task Structures: A Petri-net-based Approach*. Information Systems, 25 (1) : 43-69, 2000.

[Aals00] W. M. P. VAN DER AALST. *Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques*. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, Business Process Management: Models, Techniques, and Empirical Studies, volume 1806 of Lecture Notes in Computer Science, pages 161-183. Springer-Verlag, Berlin, 2000.

[Baum96] B. BAUMGARTEN. *Petri-Netze: Grundlagen und Anwendungen*. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford, 1996.

[Broc03] J. VOM BROCKE. *Referenzmodellierung: Gestaltung und Verteilung von Konstruktionsprozessen*. Logos, Berlin, 2003.

[Eckl06] A. ECKLEDER. *Semantikprüfung von Geschäftsprozessmodellen*. Studienarbeit BA Karlsruhe, 2006.

[Fish01] G. S. FISHMAN. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer, New York, 2001.

[Frey05] T. FREYTAG. *WoPeD - Workflow Petri Net Designer*. BA Karlsruhe, 2005.

[FrJo97] H. M. FRANKEN, H. JONKERS AND M. K. DE WEGER. *Structural and quantitative perspectives on business process modelling and analysis*. In A.R. Kaylan and A. Lehmann (eds.), Proc. 11th European Simulation Multiconference, Istanbul, Turkey, pages 595-599, June 1997.

[FrLa03] T. FREYTAG, S. I. LANDES. *PWFtool - a Petri net workflow modelling environment*. Proceedings of the Workshop "Algorithmen und Werkzeuge für Petrinetze" (AWPN), Eichstätt, 2003.

[GrBa04] P. GRÄSSLE, H. BAUMANN, P. BAUMANN. *UML 2.0 projektorientiert*. Galileo Press, Bonn, 2004.

[GuSo06] H. GUMM, S. SOMMER. *Einführung in die Informatik*. Oldenbourg, München, 2006.

[HaSt04] C. RICHTER-VON HAGEN, W. STUCKY. *Business-Process- und Workflow-Management: Prozessverbesserung durch Prozess-Management*. B.G. Teubner, Wiesbaden, 2004.

[HiLi05] F. S. HILLIER, G. J. LIEBERMAN. *Introduction to Operations Research*. McGraw-Hill, New York, 2005.

[JoFr96] H. JONKERS AND H. M. FRANKEN. *Quantitative modelling and analysis of business processes*. In A.G. Bruzzone and E.J.H. Kerckhoffs (eds.), Simulation in Industry: Proc. 8th European Simulation Symposium, vol. I, Genoa, Italy, pages 175-179, Oct. 1996.

[KaKB05] U. KASTENS, H. KLEINE BÜNING. *Modellierung: Grundlagen und formale Methoden*. Hanser, München, Wien, 2005.

[KoHu00] K. KOSMIDIS AND G. HUSZERL. *UML-Extensions for Quantitative Analysis*. In Proceedings UML 2000 Workshop Dynamic Behaviour in UML Models: Semantic Questions. LMU München, Institut für Informatik, 2000.

[Land03] S. I. LANDES. *Entwicklung eines Modellierungstools für Workflow-Petrinetze*. Diplomarbeit BA Karlsruhe, 2003.

[Law07] A. M. LAW. *Simulation Modeling and Analysis*. McGraw-Hill, New York, 2007.

[McMi95] K. L. MCMILLAN. *A technique of a state space search based on unfolding*. Formal Methods in System Design, 6 (1):45-65, 1995.

[RuAa06] N. RUSSELL, W. M. P. VAN DER AALST, A. H. M. TER HOFSTEDE AND P. WOHED. *On the suitability of uml 2.0 activity diagrams for business process modelling*. In APCCM 06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling, 2006.

[Sche96] A.-W. SCHEER. *ARIS-House of Business Engineering: Von der Geschäftsprozeßmodellierung zur Workflow-gesteuerten Anwendung;*

vom Business Process Reengineering zum Continuous Process Improvement. In: A.-W. Scheer (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Nr. 133, Saarbrücken, Universität des Saarlandes, 1996.

[Sche97] A.-W. SCHEER. *Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse*. Springer, Berlin, Heidelberg, 1997.

[Stac73] H. STACHOWIAK. *Allgemeine Modelltheorie*. Springer, Wien, 1973.

[OMGI07] OMG: *UML Infrastructure Specification*, Version 2.1.1., `http://www.omg.org/docs/formal/07-02-06.pdf`, 2007.

[Vojn97] T. VOJNAR. *Various Kinds of Petri Nets in Simulation and Modelling*. In J. Stefan, editor, Proceedings of 31st Spring International Conference on Modelling and Simulation of Systems MOSIS'97, Vol. 1, pages 227-232, Hradec nad Moravicí, Czech Republik, MARQ, Ostrava, April 1997.

[Wins04] W. L. WINSTON. *Operations Research: Applications and Algorithms*. Thomson, Belmont, 2004.

Eidesstattliche Erklärung

Christian Czech

872462

Hiermit erkläre ich, dass ich diese Arbeit selbständig abgefasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Essen, 24. Juli 2007

Unterschrift