Technische Universität München
Fakultät für Informatik
Lehrstuhl für Wirtschaftsinformatik (I 17)
Univ.-Prof. Dr. Helmut Krcmar

Evolutionäre Algorithmen zur Performance-Modellierung von Unternehmensanwendungen

Daniel Nico Sebastian Tertilt

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Martin Bichler

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Helmut Krcmar
2. Univ.-Prof. Dr. Hans Michael Gerndt

Die Dissertation wurde am 01.10.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23.04.2014 angenommen.

Zusammenfassung

Evolutionäre Algorithmen zur Performance-Modellierung von Unternehmensanwendungen

Performance-Simulationsansätze für Unternehmensanwendungen benötigen eine Abbildung des Performance-Verhaltens der Softwarekomponenten, um Aussagen über die Performance des Gesamtsystems treffen zu können. Gängige Simulationsansätze modellieren das Komponentenverhalten über deren Ressourcenbedarf. Der Praxiseinsatz zeigt jedoch, dass der Ressourcenbedarf oft nur teilweise oder gar nicht erhoben werden kann. Diese Arbeit geht der Frage nach, inwieweit das Performance-Verhalten der Softwarekomponenten über deren Antwortzeitverhalten abgebildet werden kann. Weiterhin wird untersucht, ob sich evolutionäre Algorithmen für die Modellierung des oft nur in Form von Messwerten vorliegenden Antwortzeitverhaltens eignen.

Mithilfe eines kontrollierten Softwareexperiments wird gezeigt, dass sich die Performance einer Unternehmensanwendung mittels eines existierenden Simulationsansatzes vorhersagen lässt, indem das Komponentenverhalten in Form von Antwortzeit-Komponentenmodellen eingebunden wird. Hierzu wird eine Schnittstelle implementiert, die diese Einbindung von Antwortzeit-Komponentenmodellen ermöglicht. Weiterhin wird ein evolutionärer Algorithmus durch den Prototyp *Mendel* implementiert, der Antwortzeit-Komponentenmodelle auf Basis von Antwortzeitmessdaten generiert. Anhand zahlreicher Untersuchungen wird die Effektivität dieser Modellierungsvariante gezeigt. Validiert wird der vorgestellte Ansatz durch die Experimentdurchführung an einem Realwelt-Szenario.

Diese Arbeit zeigt damit einen Weg auf, die bestehenden Performance-Simulationsansätze für Unternehmensanwendungen näher an die Praxis zu bringen. Durch die Abkehr vom in der Realität schwer zu erlangenden Ressourcenbedarf hin zu zumeist verfügbaren Antwortzeitinformationen wird die Performance-Simulation praktikabel. Weiterhin ermöglicht der Einsatz evolutionärer Algorithmen zur Modellbildung eine Generierung der Antwortzeit-Komponentenmodelle mit minimalem manuellem Aufwand.

Abstract

Ziel: Ziel dieser Dissertation ist die Untersuchung der Einsetzbarkeit von Antwortzeit-Komponentenmodellen zur Performance-Vorhersage von Unternehmensanwendungen. Weiterhin wird untersucht, ob und wie gut sich Antwortzeit-Komponentenmodelle mittels evolutionärer Algorithmen aus Messdaten generieren lassen. Hierzu werden zwei Stränge verfolgt. Zum einen wird der Prototyp *Mendel* eines evolutionären Algorithmus implementiert, der die Modellbildung auf gemessenen Antwortzeitdaten durchführt. Im Zuge der Arbeit wird eine optimale Konfiguration von *Mendel* ermittelt sowie die Approximationsgeschwindigkeit und –genauigkeit analysiert. Zum anderen wird ein existierender Simulationsansatz für die Performance von Unternehmensanwendungen ausgewählt und eine Schnittstelle umgesetzt, um die generierten Komponentenmodelle in die Simulationsmodelle dieses Simulationsansatzes zu integrieren. In einem Anwendungsfall aus der Praxis werden die beiden Stränge zusammengeführt. Im Vergleich der Simulationsergebnisse des konventionellen und erweiterten Simulationsmodells mit dem gemessenen Antwortzeitverhalten der untersuchten Anwendung ergibt sich die Aussage über die Einsetzbarkeit von generierten Antwortzeit-Komponentenmodellen in der Performance-Vorhersage von Unternehmensanwendungen.

Methode: Diese Arbeit basiert auf einem quantitativen Ansatz. Die Untersuchung erfolgt in Form eines kontrollierten Softwareexperiments, bei dem die Einbindung von Antwortzeit-Komponentenmodellen in einen existierenden Performance-Simulationsansatz die veränderliche Variable darstellt. Die Umsetzung des evolutionären Algorithmus und dessen Konfiguration sowie die Auswahl eines geeigneten Simulationsansatzes und die Integration der Komponentenmodelle in den Simulationsansatz bilden die Experimentvorbereitung. Weiterhin umfasst die Experimentvorbereitung die Erstellung des Simulationsmodells des Anwendungsfalls. Die Durchführung der Simulationsläufe selber bildet die Experimentdurchführung. Hierbei bleiben alle Faktoren konstant bis auf die Einbindung der Komponentenmodelle. Der Vergleich der Simulationsergebnisse mit dem gemessenen Performance-Verhalten der untersuchten Anwendung bildet die Experimentauswertung.

Resultate: Es ist feststellbar, dass die Einbindung von Antwortzeit-Komponentenmodellen in die Performance-Vorhersage eine Untersuchung des Performance- und Skalierungsverhalten der untersuchten Unternehmensanwendung ermöglicht. Der Ansatz ist insbesondere dann sinnvoll, wenn für die betrachtete Anwendung oder Teile der Anwendung keine Ressourcennutzungsinformationen erhoben werden können. Auch der Einsatz eines evolutionären Algorithmus zur Modellbildung auf gemessenen Antwortzeitinformationen

erwies sich als effektiv. Aus den gewonnenen Ergebnissen lassen sich folgende Aussagen ableiten:

1) Mittels Antwortzeit-Komponentenmodellen ist eine Performance-Vorhersage für Unternehmensanwendungen auch ohne Ressourcennutzungsinformationen möglich.

2) Evolutionäre Algorithmen eignen sich für die Modellierung von Antwortzeit-Komponentenmodellen auf Messdaten ohne Vorgaben und manuellen Aufwand.

3) Die Konfiguration und Implementierung des evolutionären Algorithmus hat einen großen Einfluss auf sein Approximationsverhalten.

Auswirkungen auf die Praxis: Existierende Simulationsansätze für die Performance-Vorhersage von Unternehmensanwendungen sind in der Entwicklung weit fortgeschritten, und dennoch finden sie in der Praxis kaum Anwendung. Eine der Hauptursachen hierfür ist, dass für ihren Einsatz Informationen wie die Ressourcennutzung von einzelnen Operationen benötigt werden, die in der Regel nicht vorliegen und die mit hohem manuellem Aufwand erhoben werden müssen. Zusätzlich erschwert die aufwändige Normierung und Modellbildung auf Messdaten die Einführung der Simulationsansätze. Diese Arbeit reduziert diese Hindernisse auf zwei Weisen. Zum einen zeigt sie, dass eine Performance-Vorhersage basierend auf Antwortzeitinformationen möglich ist. Antwortzeitinformationen sind ein gängiges Resultat von Last- und Performance-Tests und liegen oft entweder bereits vor oder können mit geringem Aufwand erhoben werden. Zum anderen wird mit dem evolutionären Algorithmus ein Modellierungsansatz vorgestellt, der auf beliebig komplexen Messdatensätzen ohne Vorgaben und manuellem Aufwand Komponentenmodelle erstellt. Die vorliegende Arbeit reduziert damit die Einstiegshürde für den Einsatz von Performance-Simulationsansätzen in der Praxis.

Einschränkungen: Die vorliegende Arbeit demonstriert die Anwendbarkeit des vorgestellten Ansatzes mit dem verwendeten Simulationsansatz und für den vorgestellten Anwendungsfall. Eine Generalisierung geschieht in dieser Arbeit nicht. Weiterhin sind die vorgestellten Prototypen, insbesondere der evolutionäre Algorithmus *Mendel*, ausschließlich für den Einsatz in dieser Arbeit konzipiert. Produktqualität ist bei der Umsetzung explizit nicht angestrebt. Als Metrik für die Performance einer Anwendung wird in dieser Arbeit ausschließlich die Antwortzeit betrachtet. Andere Metriken wie der Durchsatz erfahren keine Betrachtung.

Inhaltsverzeichnis

Liste der Abkürzungen und Akronyme

AA	Arbeitsagentur
ABAP	Advanced Business Application Programming
ADF	Application Development Framework
BA	Bundesagentur für Arbeit
CBSE	Component-based Software Engineering
COMPASS	COMponent-based Parallel System Simulator
CRM	Customer Relationship Management
CS	Composed Service
EA	Evolutionärer Algorithmus
EAI	Enterprise Application Integration
EMPS	Environment for Memory Performance Studies
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
GA	Genetischer Algorithmus
GUI	Graphical User Interface
HEC	High End Computing
HPC	High Performance Computing
ISAM	Index Sequential Access Method
JMX	Java Management Extension
LQM	Layered Queuing Model
LQN	Layered Queuing Network
LQNS	Layered Queuing Network Solver
LQSIM	Layered Queuing Simulator
MOM	Message-orientierte Middleware
OSB	Oracle Service Bus
OWSM	Oracle Web Services Manager
PACE	Performance Analysis and Characterization Environment
PCI	Performance Curve Integration
PCM	Palladio Component Model
QoS	Quality of Service
ROBASO	Rollenbasierte Oberflächenanwendung
SCALA	Scalability Analyzer
SEFF	Service Effect Specification
SOA	Service-orientierte Architektur
SOPM	Service-Oriented Performance Modelling
SSJ	Stochastic Simulation in Java
TPS	Transactions per Second
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
VSAM	Virtual Storage Access Method
WESPE	Web Service Security and Policy Enforcement
WSDL	Web Service Description Language

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

Evolutionäre Algorithmen (EA) und insbesondere deren Spezialform der genetischen Algorithmen (GA) stellen ein mächtiges, der natürlichen Evolution nachempfundenes Mittel zur Lösung komplexer Problemstellungen dar. In der Logistik, im industriellen Design und in zahlreichen anderen Bereichen finden EAs Anwendung auf Probleme, die mit anderen, exakten Methoden nicht mehr oder nur mit unverhältnismäßig hohem Zeit- und Ressourceneinsatz lösbar sind. Die mit EAs lösbaren Probleme lassen sich nach (Eiben/Smith 2003a) in drei Klassen einteilen:

- Optimierungsprobleme sind Probleme, bei denen das Modell bekannt ist sowie das gewünschte Ergebnis, jedoch die zur Erlangung des Ergebnisses notwendigen Eingabewerte unbekannt sind. Das bekannteste Optimierungsproblem ist das Traveling Salesman Problem (Erk/Priese 2009).

- Ein Modellierungs- oder Systemidentifikationsproblem liegt vor, wenn zusammengehörende Ein- und Ausgabedaten bekannt sind, jedoch das Modell nicht, das aus den Eingabedaten die entsprechenden Ausgabedaten generiert. Ein Beispiel für ein Modellierungsproblem ist das Antwortzeitverhalten einer Softwarekomponente oder eines Web Services: liegen die Komponente oder der Service in ausführbarer Weise vor, und verfügt man über valide Eingabedaten, so lässt sich die Antwortzeit ohne weiteres messen. Die Identifikation eines Modells für das Antwortzeitverhalten einer Softwarekomponente ist jedoch eine hoch anspruchsvolle Aufgabe (Chen et al. 2005; Wu/Woodside 2004).

- Bei einem Simulationsproblem liegen sowohl das Modell als auch Eingabedaten vor, und die Ausgabe wird gesucht. Ein Beispiel für ein Simulationsproblem ist die Berechnung des täglichen Wetterberichts, für den sowohl die Eingabedaten (Messwerte wie Luftdruck, Luftfeuchtigkeit und Temperatur) als auch ein über Jahre hinweg entwickeltes Modell vorliegen.

Die Effizienz des Einsatzes von EAs zur Lösung eines Problems ist stark abhängig von der Möglichkeit, das Problem in einem EA abzubilden (De Jong/Spears 1989). Aus diesem Grund finden EAs insbesondere bei der Lösung kombinatorischer Probleme Anwendung, wie beispielsweise dem Vehicle Routing Problem der Logistik (Thangiah et al. 1991; Ochi et al.

1998; Prins 2004). Bei kombinatorischen Problemstellungen ist die Abbildung der Problemstellung in die Struktur des EA einfach, da die betrachteten Elemente von endlicher Anzahl sind und durch eine einfache Kodierung (wie beispielsweise eine Durchnummerierung) im EA abgebildet werden können.

Bei numerischen Problemen ist diese einfache Kodierung nicht gegeben. Diese Art von Problemen besitzt einen Lösungsraum von einer oder mehreren Dimensionen unendlicher Größe. Der effizienten Abbildung des Problems im EA kommt bei numerischen Problemstellungen eine noch größere Bedeutung zu als bei kombinatorischen, sie entscheidet maßgeblich über die erfolgreiche Lösungsfindung des Algorithmus. Neuere Arbeiten wie (Zhong et al. 2004; Parpinelli/Lopes 2011) belegen jedoch den erfolgreichen Einsatz von EAs zur Lösung numerischer Problemstellungen.

Die Modellierung des Antwortzeitverhaltens von Unternehmensanwendungen zur Unterstützung der Performance-Analyse dieser Systeme ist ein solches numerisches Modellierungsproblem.

Unternehmensanwendungen, wie beispielsweise ERP- oder CRM-Systeme, bilden in vielen Unternehmen das Rückgrat zahlreicher Geschäftsprozesse. Diese Systeme verwalten nicht nur geschäftskritische Datenbestände, sie koordinieren auch Geschäftsabläufe von der Lagerbestandsverwaltung über die Projektplanung bis hin zur Personalverwaltung und Buchhaltung. Hierdurch werden die Unternehmensanwendungen selber zu einem geschäftskritischen Bestandteil der Unternehmens-IT, ihre Performance zur Geschwindigkeitsobergrenze der durch sie verwalteten Geschäftsprozesse. Veränderungen an implementierten Unternehmensanwendungen, seien es Software-Updates, Hardwareänderungen oder auch nur Veränderungen im Nutzerverhalten, stellen damit immer einen kritischen Eingriff in den operativen Geschäftsbetrieb dar. Sollte sich die Veränderung negativ auf die Performance der Unternehmensanwendung auswirken, so besteht die Gefahr, dass ganze Geschäftsprozesse verlangsamt oder sogar stillgelegt werden.

Es gibt drei Sichten auf die Performance von Unternehmensanwendungen. Erstens ist dies die rückwirkende Sicht, die Frage „Was ist gewesen?“. Diese Frage kann durch die Analyse von Statistiken und Logfiles beantwortet werden, soweit diese aufgezeichnet wurden. Zweitens ist es die Sicht auf das, „was ist“. Hierunter fällt das klassische Monitoring, also die Darstellung des aktuellen Systemzustands. Und zu guter Letzt ergibt sich die Frage „Was wird sein?“. Diese Fragestellung ist die schwierigste, da sie über das Messen allein hinausgeht.

Der klassische Ansatz, die Frage nach dem zukünftigen Verhalten zu beantworten, ist das Lasttesten. Beim Lasttesten wird das aktuelle oder zukünftig zu verwendende System unter die höchste zu erwartende Last gesetzt und das Verhalten beobachtet, protokolliert und analysiert. Das Lasttesten erfordert jedoch einige Voraussetzungen, die nicht immer, insbesondere nicht bei großen Unternehmensanwendungen, gegeben sind:

1. Die vorauszusagende Anwendung muss (möglichst vollständig) für den Lasttest zur Verfügung stehen.

2. Die für den Lasttest verwendete Hardware muss mit der Hardware des Produktivsystems vergleichbar sein.

3. Der Zustand der Anwendung, beispielsweise die Befüllung mit Daten, muss realitätsnah sein.

4. Ausreichend Last muss produzierbar sein.

5. Die höchste zu erwartende Last muss bekannt sein.

Es ist ersichtlich, dass die Frage nach dem zukünftigen Verhalten einer Unternehmensanwendung einige Fälle beinhaltet, die durch Lasttesten nicht abgedeckt werden können. So ist es nicht immer möglich, große Unternehmensanwendungen, die oft in großen Rechenzentren gehostet werden, für den Lasttest komplett nachzustellen. Weiterhin ist es möglich, dass Teile des vorauszusagenden Systems noch gar nicht vollständig implementiert zu Verfügung stehen. Dies ist beispielsweise der Fall, wenn verschiedene Architekturoptionen untersucht werden sollen. Und schließlich scheitert der Lasttest tatsächlich nicht selten am Hardware- und Implementierungsaufwand der Lasttreiber.

Die genannten Hindernisse können vermieden werden durch den Einsatz von Simulationstechniken wie den Layered Queuing Networks (Franks et al. 2009; Gradl et al. 2009) und dem Palladio Component Model (Becker et al. 2007; Koziolek et al. 2007). Diese Simulationsansätze bieten den Vorteil, dass sie durch Abstraktion einige hinderliche Eigenschaften des Lasttests umgehen können, beispielsweise die Bindung an die Realzeit und das notwendige Vorhandensein der betrachteten Soft- und Hardware. Weiterhin bietet die Simulation die Möglichkeit, Szenarien zu analysieren, die real nicht abgebildet werden können, wie beispielsweise das Ausprobieren verschiedener Hardwarekonfigurationen und

Systemarchitekturen. Das richtige Simulationsmodell vorausgesetzt liefert die Simulation dabei verlässliche Voraussagen.

Doch gerade die Erstellung eines solchen Simulationsmodells für große Unternehmensanwendungen birgt zahlreiche Schwierigkeiten (Jain 1991). In den gängigen Ansätzen zur Simulation der Performance von Softwaresystemen besteht das Simulationsmodell der untersuchten Anwendung aus zwei Ebenen: das Systemmodell, das den Aufrufgraph über die Systemkomponenten darstellt, und die Komponentenmodelle, die das Verhalten der einzelnen Systemkomponenten abbilden, gegebenenfalls ergänzt um ein Ressourcenmodell, welches die zugrunde liegende Hardware beschreibt. Während die meisten Simulationsansätze jedoch ausführlich beschreiben, wie das Systemmodell zu erstellen ist, wird die Erstellung der Komponentenmodelle zumeist dem Anwender überlassen oder durch eine Verteilungsfunktion stark abstrahiert (siehe beispielsweise (Koziolek et al. 2007)). Weiterhin erfordern Simulationsmodelle wie das Palladio Component Model zur Betrachtung von Skalierungseffekten Aussagen über die Ressourcennutzung auf Hardwareebene für jede aufgerufene Operation (vgl. (Becker et al. 2009)). In der Praxis sind diese Ressourcennutzungsinformationen schwer oder gar nicht zu erlangen. Die Hardware, auf der die Unternehmensanwendung installiert wird, steht im Allgemeinen fest, so dass im Rahmen von Last- und Performancetests das Antwortzeitverhalten gemessen und überprüft wird. Doch erlauben Aussagen über das Antwortzeitverhalten der einzelnen Operationen zwar eine Simulation des normalen Verhaltens der Anwendung, jedoch mit den existierenden Simulationsverfahren keine Skalierbarkeitsbetrachtungen, also keine Betrachtung der Veränderung des Verhaltens der Anwendung unter erhöhter Last, da keine begrenzte Ressource das Systemverhalten beeinflusst. Die Aussage, dass eine Operation ein durchschnittliches Antwortzeitverhalten von 15 Millisekunden hat, ermöglicht keine Rückschlüsse über das Verhalten unter Last.

Diese Arbeit erörtert, wie das Antwortzeitverhalten von Komponenten einer Unternehmensanwendung zur Skalierungsbetrachtung des Gesamtsystems herangezogen werden kann. Hierzu wird bestimmt, welche Informationen über das Antwortzeitverhalten der Komponenten gemessen werden müssen, damit daraus Modelle extrahiert werden können. Evolutionäre Algorithmen werden verwendet, um aus diesen Messwerten des Antwortzeitverhaltens Modelle zu bilden. Weiterhin wird analysiert, wie ein bestehender Simulationsansatz für die Performance von Unternehmensanwendungen erweitert werden muss, um die mit den evolutionären Algorithmen gewonnenen Modelle so zu integrieren, dass das Verhalten der betrachteten Unternehmensanwendung unter Last simuliert werden kann. In einem kontrollierten Softwareexperiment wird anhand eines aus der Praxis

stammenden Beispiels untersucht, wie sich die Einbindung von durch einen evolutionären Algorithmus generierten Komponentenmodellen in die Performance-Simulation von Unternehmensanwendungen auswirkt.

Die Arbeit schlägt damit die Brücke zwischen den Lasttests, dem Monitoring und der Simulation. Während im Lasttest und im Monitoring Informationen über das Antwortzeitverhalten der Softwarekomponenten in Form von Einzeldatensätzen erhoben wird erwarten Simulationsansätze (mathematische) Modelle des Komponenten-Performanceverhaltens. Durch den Einsatz evolutionärer Algorithmen schließt die vorliegende Arbeit diese Lücke.

1.2 Annahmen und Forschungsfragen

Gegenwärtige Simulationsansätze für die Performance-Analyse von Unternehmensanwendungen benötigen Informationen über die Ressourcennutzung der Anwendungsoperationen, sowie über die zu Verfügung stehenden Ressourcen, um das Skalierungsverhalten einer Anwendung zu simulieren. Simulationsansätze wie das Palladio Component Model (PCM) ermöglichen eine Simulation rein auf Antwortzeitverhalten, diese beschränkt sich jedoch ausschließlich auf das Verhalten unter Normallast und macht keine Aussagen über das Hochlastverhalten der betrachteten Anwendung. Liegt jedoch, beispielsweise aus einem Lasttest, das Antwortzeitverhalten einer Operation vom Niedrig- bis zum Hochlastbereich in Form von Messwerten vor, und kann dies so in ein Modell überführt werden, dass es von der Simulationsumgebung verarbeitet werden kann, so liegt die folgende erste zentrale Annahme nahe:

Annahme 1: Antwortzeitmodelle für Softwarekomponenten ermöglichen eine Performance-Analyse von Unternehmensanwendungen auf Basis gemessener Antwortzeiten mittels eines existierenden Performance-Simulationsansatzes.

Bei komplexen Unternehmensanwendungen ist eine manuelle Erstellung der Antwortzeitmodelle aus den Performance-Messwerten nicht möglich. Auch der Einsatz von mathematischen Regressionsansätzen ist auf großen, vieldimensionalen Datensätzen ohne Annahmen über die Struktur der Daten schwierig. Evolutionäre Algorithmen als Repräsentant des maschinellen Lernens versprechen eine Modellierung multidimensionaler Datensätze ohne manuellen Aufwand. Hieraus ergibt sich die folgende zweite Annahme:

Annahme 2. Durch evolutionäre Algorithmen können Antwortzeitmodelle für Softwarekomponenten automatisch aus Messdaten generiert werden.

Zur Überprüfung dieser Annahmen sind drei Schritte notwendig:

1. Ein geeigneter Simulationsansatz ist zu wählen, in dem die Performance einer Unternehmensanwendung simuliert werden kann.

2. Ein evolutionärer Algorithmus ist zu entwickeln sowie ein Verfahren, um die Ergebnisse des evolutionären Algorithmus in die Simulation zu integrieren.

3. Die Simulation eines repräsentativen Anwendungsfalls ist einmal ohne und einmal mit den generierten Antwortzeitmodellen der Komponenten durchzuführen und die Ergebnisse sind mit realen Messwerten der simulierten Unternehmensanwendung zu vergleichen.

Bei der Performance-Analyse auf Antwortzeitverhalten ist eine wichtige Beschränkung im Vergleich zur Performance-Analyse auf Basis der Ressourcennutzungen zu betonen. Performance-Analysen auf Basis von Antwortzeitverhalten können immer nur Prognosen über das Verhalten auf derselben Hardware erstellen, auf welcher das Antwortzeitverhalten gemessen wurde. Eine Betrachtung des Performance-Verhaltens auf veränderter Hardware ist auf diese Weise nicht möglich. Die Projekterfahrung zeigt jedoch, dass Performance-Analysen zumeist zur Überprüfung von Architekturentscheidungen verwendet werden, während die zu Verfügung stehende Hardware bereits feststeht, so dass sich für den Projektkontext hierdurch keine größere Einschränkung ergibt.

Die Modellierung des Antwortzeitverhaltens von Anwendungskomponenten und ihrer Operationen ist, wie eingangs beschrieben, ein numerisches Problem. Ein mathematisches Modell muss identifiziert werden, welches die gemessenen Werte möglichst gut abbildet. Jedoch wirken mehrere Einflüsse auf das Antwortzeitverhalten einer Komponente ein, neben der Anzahl der parallelen Zugriffe auch Faktoren wie der Typ und die Größe der Übergabeparameter. Diese Faktoren sind von Operation zu Operation unterschiedlich, semantisches Wissen ist notwendig um entscheiden zu können, welche Faktoren für welche Operation in Betracht kommen. Diese Entscheidung wird jedoch bereits zum Messen des Antwortzeitverhaltens getroffen, so dass bei der Modellierung davon ausgegangen werden muss, dass die gemessenen Faktoren die relevanten sind.

Um den Ablauf der Arbeit zu strukturieren, werden drei Forschungsfragen herangezogen. Forschungsfrage 1 bildet den theoretischen Kontext dieser Arbeit. Zu ihrer Beantwortung werden existierende Simulationsansätze für das Performance-Verhalten von

Unternehmensanwendungen vorgestellt und analysiert. Die Beantwortung der zweiten Forschungsfrage bildet die Experimentvorbereitung des kontrollierten Softwareexperiments und umfasst die Implementierung und Konfiguration der für das Experiment benötigten Experimentumgebung. Die dritte Forschungsfrage dient zur Überprüfung des entwickelten Ansatzes an einer realen Unternehmensanwendung. Sie stellt die Experimentdurchführung dar und liefert die Überprüfung der zuvor getroffenen Annahmen.

Forschungsfrage 1: Welcher in der Literatur beschriebene Simulationsansatz für das Performance-Verhalten von Unternehmensanwendungen eignet sich für die Performance-Analyse auf Basis des Antwortzeitverhaltens der Anwendungskomponenten?

Im Rahmen der Bearbeitung der ersten Forschungsfrage erfolgt eine Literaturanalyse zur Identifikation der in Wissenschaft und Industrie eingesetzten Simulationsansätze für das Performance-Verhalten von Unternehmensanwendungen. Der Fokus liegt bei der Literaturanalyse auf tatsächlich um- und eingesetzten Simulationsansätzen, die zudem verfügbar sind. Eine Voranalyse der Literatur ergab, dass zahlreiche Verfahren beschrieben werden, die wenigsten jedoch in einer aktuellen Implementierung verfügbar sind. Diese Ansätze werden in der Literaturanalyse kurz vorgestellt, helfen jedoch bei der Durchführung des kontrollierten Softwareexperiments nicht. Bei der Untersuchung der tatsächlich verfügbaren Simulationsansätze für das Performance-Verhalten von Unternehmensanwendung wird zudem ein Blick auf die Integrierbarkeit von Antwortzeitmodellen geworfen.

Das Ziel der Beantwortung der ersten Forschungsfrage ist eine Übersicht über die existierenden Simulationsansätze für das Performance-Verhalten von Unternehmensanwendungen. Weiterhin wird im Zuge der Bearbeitung der ersten Forschungsfrage ein Simulationsansatz ausgewählt, der als Teil der Experimentumgebung zur Überprüfung der oben getroffenen Annahmen und zur Beantwortung der folgenden Forschungsfragen dient.

Forschungsfrage 2 beschäftigt sich mit der Generierung von Antwortzeit-Komponentenmodellen durch evolutionäre Algorithmen sowie deren Einbindung in die Simulation des Performance-Verhaltens von Unternehmensanwendungen.

Forschungsfrage 2: Wie kann ein evolutionärer Algorithmus in die Simulation der Performance einer Unternehmensanwendung eingebunden werden und wie muss der evolutionäre Algorithmus konfiguriert sein?

Die Beantwortung der zweiten Forschungsfrage erfolgt als Teil eines kontrollierten Softwareexperiments, welches die Forschungsfragen 2 und 3 umspannt. Die Beantwortung der zweiten Forschungsfrage bildet zugleich die Experimentvorbereitung, indem die technischen Bedingungen für die Durchführung des kontrollierten Softwareexperiments geschaffen werden. Hierzu wird prototypisch ein evolutionärer Algorithmus implementiert, der basierend auf Performance-Messdaten Antwortzeit-Komponentenmodelle generiert. Diese Modelle werden in den bei der Beantwortung von Forschungsfrage 1 ausgewählten Simulationsansatz integriert. Durch eine Untersuchung der optimalen Konfiguration des evolutionären Algorithmus für die Modellierung des Antwortzeitverhaltens basierend auf Messdaten wird sichergestellt, dass die während der Experimentdurchführung gewonnenen Ergebnisse nicht durch Fehlkonfigurationen verfälscht werden.

Das Ziel der Beantwortung der zweiten Forschungsfrage ist die Umsetzung eines Prototyps eines evolutionären Algorithmus zur Modellierung von Antwortzeit-Komponentenmodellen, sowie die optimale Konfiguration des evolutionären Algorithmus. Durch die Umsetzung des Prototyps erfolgt die Überprüfung der zweiten Annahme. Zudem erfolgt die Implementierung einer Schnittstelle zur Einbindung der generierten Komponentenmodelle in die ausgewählte Simulationsumgebung. Der entwickelte Prototyp dient als Basis für das kontrollierte Softwareexperiment und damit der Beantwortung der folgenden Forschungsfrage 3.

Forschungsfrage 3: Wie wirkt sich der Einsatz von generierten Komponentenmodellen auf die Performance-Simulation von Unternehmensanwendungen im Fallbeispiel aus?

Die dritte Forschungsfrage widmet sich der Überprüfung der ersten Annahme. Anhand einer Fallstudie an einer realen Unternehmensanwendung wird überprüft, in wieweit die Einbindung von automatisch generierten Antwortzeit-Komponentenmodellen das Simulationsergebnis verändern. Die Beantwortung dieser Forschungsfrage bildet die Durchführung des kontrollierten Softwareexperiments. Durch einen Vergleich der Performance-Simulationsergebnisse des Anwendungsfalls mit und ohne den Einsatz der Komponentenmodellen jeweils mit gemessenen Performance-Werten ergibt sich eine Aussage darüber, ob sich das Simulationsergebnis durch die Verwendung von Antwortzeit-Komponentenmodellen zum Positiven oder Negativen verändert.

Das Ziel der Beantwortung der dritten Forschungsfrage liegt in der Gewinnung einer Aussage über die Ergebnisveränderung des verwendeten Simulationsansatzes durch den Einsatz automatisch generierter Antwortzeit-Komponentenmodelle. Zudem werden die

Einsetzbarkeit des entwickelten Ansatzes im industriellen Umfeld demonstriert und Hindernisse aufgezeigt.

1.3 Struktur der Arbeit

Die drei aufgestellten Forschungsfragen werden genutzt, um die Arbeit zu strukturieren. Die einzelnen Kapitel tragen zur Beantwortung je einer der Forschungsfragen bei.

Kapitel 2 beginnt mit einer Einführung in die zum Verständnis dieser Arbeit notwendigen Grundlagen. In diesem Kapitel wird ausgeführt, weshalb eine Performance-Analyse von Unternehmensanwendungen notwendig ist, und welche grundlegenden Aufgabenstellungen sich bei der Performance-Analyse von Unternehmensanwendungen ergeben. Anschließend erfolgt eine Einführung in die Verwendung von Simulation zur Analyse der Performance einer Unternehmensanwendung. Gefolgt wird diese Einführung von der Vorstellung von Komponentenmodellen. Komponentenmodelle, insbesondere Antwortzeit-Komponentenmodelle, bilden ein Kernelement dieser Arbeit, da ihre Anwendung die veränderliche Variable im kontrollierten Softwareexperiment bildet. Zur Schaffung eines einheitlichen Verständnisses werden Komponentenmodelle definiert und ihre Eigenschaften und Limitationen vorgestellt. Im Anschluss erfolgt die Einführung in die Theorie evolutionärer Algorithmen sowie ihrer Verwendung zur Modellapproximation. Im Detail wird in diesem Abschnitt auf die Unterform der genetischen Algorithmen eingegangen, da diese Unterform in der vorliegenden Arbeit verwendet wird, sowie auf die Konfiguration des Algorithmus. Die Konfiguration wird im späteren Verlauf dieser Arbeit eine wichtige Rolle einnehmen. Letztendlich wird das Kapitel durch eine Erläuterung der grundlegenden Konzepte und Eigenschaften von Unternehmensanwendungen und Service-orientierten Architekturen abgeschlossen. Beide Konzepte sind elementar für das Verständnis des in Kapitel 5 beschriebenen Anwendungsfalls.

In Kapitel 3 erfolgt eine Analyse der verwandten Forschungsarbeiten zu den Themen *Performance-Analyse von Unternehmensanwendungen* sowie *Maschinelles Lernen zur Modellierung des Performanceverhaltens von Softwarekomponenten*. Mithilfe der Literaturanalyse wird aufgezeigt, welche Forschung im Umfeld der vorliegenden Arbeit bereits durchgeführt wurde, und wie sich die vorliegende Arbeit in diesen Kontext einbettet. Durch eine umfassende Analyse wird klar herausgearbeitet, welche Forschungslücke durch diese Arbeit gefüllt wird. Die Literaturanalyse dient zudem noch einem weiteren Zweck – der Auswahl eines geeigneten Simulationsansatzes als Basis für das durchzuführende kontrollierte Softwareexperiment. Neben der Darstellung des wissenschaftlichen Kontextes

dient die in Kapitel 3 durchgeführte Literaturanalyse damit zudem der Beantwortung der ersten Forschungsfrage.

In Kapitel 4 erfolgten die Beschreibung der Implementierung des evolutionären Algorithmus sowie die Integration der durch den Algorithmus generierten Komponentenmodelle in die ausgewählte Simulationsumgebung. Das Kapitel beschreibt die Experimentvorbereitung des kontrollierten Softwareexperiments. Hierzu gehören neben der Implementierung auch die Ermittlung einer optimalen Konfiguration des evolutionären Algorithmus sowie eine Betrachtung verschiedener möglicher Fehlerwertberechnungen zur Ermittlung der Fitness einzelner Individuen des Algorithmus. Die Beantwortung der zweiten Forschungsfrage geschieht in diesem Kapitel. Als Forschungsmethodik kommt zur Beantwortung der zweiten Forschungsfrage Prototyping zum Einsatz. Sowohl der evolutionäre Algorithmus als auch die Integration in die Simulationsumgebung wird prototypisch implementiert und bildet damit einen Teil der Experimentumgebung.

Kapitel 5 beschreibt die Anwendung der Performance-Simulation auf einen Realwelt-Anwendungsfall in Form einer Fallstudie. Als Anwendungsfall wurde der Terminierungsprozess des von der Bundesagentur für Arbeit umgesetzten Projektes ROBASO gewählt. In diesem Kapitel wird der Anwendungsfall vorgestellt sowie das Vorgehen zur Messung des Performance-Verhaltens der einzelnen Komponenten des Anwendungsfalls. Anschließend erfolgt eine Beschreibung der Anwendung des evolutionären Algorithmus zur Modellierung der Komponentenmodelle der vom Terminierungsprozess verwendeten Services sowie der Erstellung der Simulationsmodelle für den Anwendungsfall. Die Simulation ohne und mit der Einbindung der Komponentenmodelle wird anschließend dargestellt, gefolgt von der Interpretation und Bewertung der Ergebnisse. Das in Kapitel 5 beschriebene Vorgehen dient der Beantwortung der dritten Forschungsfrage.

Kapitel 6 fasst die Arbeit und deren Ergebnisse zusammen und bietet einen Ausblick zu den weiteren möglichen Schritten in der Forschung, die im Anschluss an diese Arbeit unternommen werden können.

Einen Überblick über den Zusammenhang der Forschungsfragen und der Kapitel liefert die Abbildung 1.1.

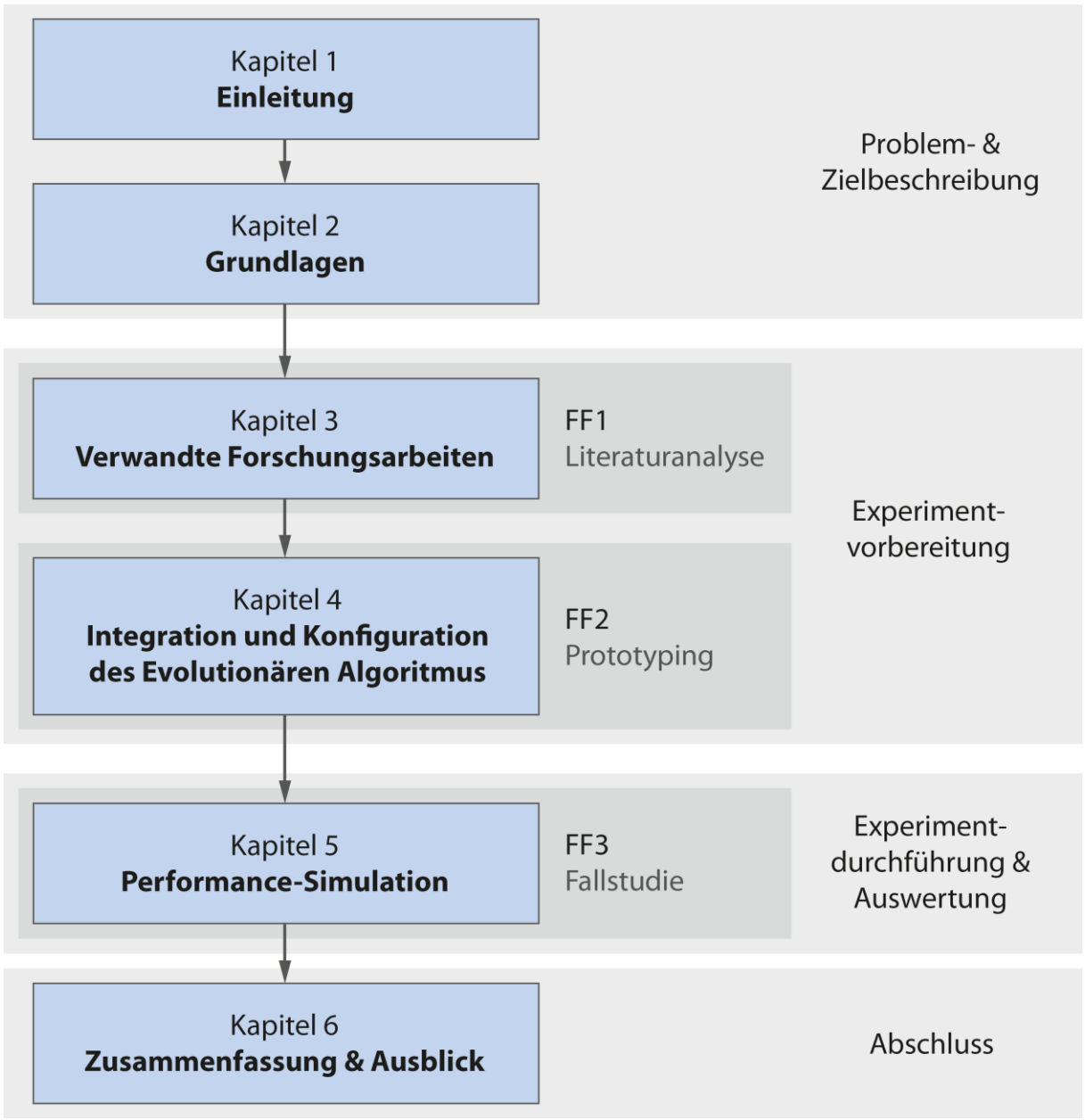


Abbildung 1.1: Struktur der Arbeit. Quelle: Eigene Darstellung.

2 Grundlagen

2.1 Abstrakte Problemstellung

Unternehmensanwendungen bilden aufgrund ihrer Komplexität und den kontinuierlichen, durch wirtschaftliche oder rechtliche Veränderungen verursachten Anpassungen ein herausforderndes Anwendungsgebiet für Performance-Analysen (Becker et al. 2009; Wert 2011). Zu den Schwierigkeiten bei der Performance-Analyse von Unternehmensanwendungen zählen die meist komplexe, aus vernetzten Anwendungen unterschiedlichsten Alters bestehende Systemlandschaft, implementiert in unterschiedlichen Technologien und Programmiersprachen, und die meist teilweise oder vollständig fehlende Dokumentation. Hinzu kommen oft organisatorische Barrieren: aus operativen oder sicherheitspolitischen Gründen ist ein Zugriff auf die produktiven Middleware-Komponenten oder Betriebssysteme nur eingeschränkt oder gar nicht möglich, ebenso das Instrumentieren der relevanten Softwarekomponenten. Auch die Möglichkeit des Vermessens der Softwarekomponenten zum Erlangen der für die meisten Simulationsansätze notwendigen Ressourcenverbräuche der Softwarekomponenten ist oft nicht gegeben. Produktive Softwaresysteme dürfen im Allgemeinen zur Sicherstellung des operativen Betriebs nicht instrumentiert werden, und produktivnahe Testumgebungen sind, gerade bei großen Anwendungen, nur selten vorhanden und von den zahlreichen Stakeholdern eines Softwareprojekts stark umkämpft. Zudem liegen, gerade bei großen, verteilten Anwendungen, verwendete Softwarekomponenten oft in dem Verantwortungsbereich anderer Abteilungen, oder werden extern zugekauft, wodurch ein Zugriff zusätzlich erschwert wird.

Für die Performance-Analyse einer großen Unternehmensanwendung bedeutet dies, dass zwar im Normalfall Architekturinformationen vorliegen, die einzelnen Komponenten jedoch als Black Box betrachtet werden müssen. Die Performance-Analyse muss demnach auf den aus Monitoring und Lasttest gewonnenen, extern verfügbaren Performance-Informationen (im Allgemeinen Antwortzeitinformationen) geschehen, nicht wie in den Performance-Simulationsansätzen gefordert auf Ressourcennutzungsinformationen (Becker et al. 2007; Franks et al. 2005). Dieses Vorgehen bringt Einschränkungen mit sich, beispielsweise wird die Möglichkeit der Simulation des Effekts von Hardwareänderungen begrenzt. Tatsächlich zeigt die Projekterfahrung jedoch, dass durch die Verwendung von Antwortzeitinformationen die Performance-Analyse großer, verteilter Unternehmensanwendungen erst ermöglicht wird.

Existierende Performance-Simulationsansätze sehen die Verwendung von Antwortzeitinformationen jedoch nicht oder nur beschränkt vor (vgl. (Wert 2011)). Die abstrakte, in dieser Arbeit behandelte Problemstellung gliedert sich damit in zwei Teile:

- Die Erstellung von Antwortzeitmodellen für Softwarekomponenten aus gemessenen und in ihrer Struktur vorab nicht bekannten Antwortzeitinformationen einer Softwarekomponente mit geringem manuellem Aufwand.

- Die Einbindung dieser Antwortzeitmodelle in einen existierenden Performance-Simulationsansatz.

Im Rahmen der vorliegenden Arbeit wird eine Lösung für diese Problemstellung vorgestellt.

2.2 Performance-Analyse durch Simulation

Neben der Messung stellt die Simulation die häufigste Anwendung zur Performance-Analyse von Softwaresystemen dar. Anders als die Messung erlaubt die Simulation eine Voraussage für die Zukunft unter veränderten Bedingungen. In diesem Kapitel wird die Simulation als Werkzeug für die Performance-Analyse vorgestellt. Hierzu wird der Einsatz der Simulation motiviert, anschließend werden verschiedene Simulationsansätze erläutert. Im nächsten Schritt werden die Ziele der Simulation sowie Voraussetzungen und Schritte zur Durchführung der Simulation definiert. Abschließend erfolgt eine Analyse der Limitationen der Simulation und der durch sie erzielbaren Ergebnisse.

2.2.1 Motivation zur Nutzung von Simulation zur Performance-Analyse

Die Simulation ermöglicht es, Situationen abzubilden, die sich nicht oder nur mit erheblichem Aufwand real nachbilden lassen. So wird Simulation insbesondere in Bereichen eingesetzt, in denen das reale Nachstellen einer zu analysierenden Situation erhebliche finanzielle oder andere Kosten (wie beispielsweise die Verschmutzung der Umwelt) mit sich bringt, oder das vorherzusagende Element nicht durch den Menschen veränderbar ist oder verändert werden soll. Klassische Anwendungsgebiete für die Simulation sind die Wettervorhersage (Johnson et al. 1996) und der Nahverkehr (Boel/Mihaylova 2006).

Große Unternehmensanwendungen weisen ähnliche Eigenschaften auf wie die genannten Anwendungsgebiete. Ihr Aufbau ist komplex (so umfasst das SAP ERP-System über 160.000 Einzelprogramme (Bögelsack 2011)), und die Interaktion zwischen den Einzelkomponenten ist zu umfangreich, als dass sie von einem Menschen nachvollzogen werden könnte. Zudem beeinflussen zahlreiche Faktoren (Eingabewerte, Art und Anzahl der

CPU, freier Hauptspeicher, Art und Anzahl der I/O-Schnittstellen, Datenbestand der Anwendung und viele mehr) ihr Verhalten.

Simulation ist ein adäquates Mittel, um dieser Komplexität Herr zu werden. Hierbei unterstützt die Simulation die Reduktion der Komplexität mittels zweier Eigenschaften:

- **Abstraktion** – Ein Simulationsmodell erlaubt die Abstraktion von für die Performance-Analyse nicht relevanten Teilen eines betrachteten Systems. Die Abstraktion kann auf zwei Weisen erfolgen: zum einen können Systemkomponenten zusammengefasst oder nur grob beschrieben werden (beispielsweise durch eine einfach Verteilungsfunktion), die für die aktuelle Analyse nicht relevant sind. Hierdurch wird es dem Simulations-Anwender ermöglicht, bei der Modellierung nicht relevanter Komponenten Aufwand zu sparen, und diesen auf die Erstellung eines möglichst korrekten Modells der relevanten Komponenten zu verwenden.

- **Sub-Modellierung** – Ist ein System oder eine Komponente zu komplex, um ihr Performance-Verhalten durch die Angabe von Ressourcennutzungsinformationen oder Antwortzeitmodelle darzustellen, so kann sie im Simulationsmodell in Sub-Komponenten heruntergebrochen werden. Dieses Verfahren ist iterativ, so kann eine Sub-Komponente erneut durch Sub-Sub-Komponenten und deren Zusammenspiel abgebildet werden. Die Komplexität der durch Ressourcennutzungsinformationen oder Antwortzeitmodelle darzustellenden Komponenten nimmt durch diese Sub-Modellierung stark ab, die Wiederverwendbarkeit hingegen zu. Die gängigen Simulationsmodelle orientieren sich diesbezüglich größtenteils an dem Component-based Software Engineering (Heineman/Councill 2001; Szyperski et al. 2002).

Simulations-Ansätze bieten einige Vorteile gegenüber anderen Performance-Analysetechniken wie dem Messen:

- **Vermeidung realer Gefahren** – Nicht erst die Katastrophe von Tschernobyl beweist, dass Untersuchungen am realen Objekt zu realen Gefahren und Schäden führen können. Auch wenn die globalen Gefahren durch Versuche an Softwaresystemen geringer sind können aus dem Ruder laufende Untersuchungen an Produktivsystemen gravierende Folgen für ein Unternehmen haben, bis hin zum Konkurs. Bei der Simulation bestehen, durch den Verzicht auf die Verwendung produktiver Systeme, diese Gefahren nicht.

- **Untersuchung von nicht nachstellbaren Situationen** – Die Simulation erlaubt es, Situationen zu untersuchen, die nicht real nachgestellt werden können oder sollen. So ist es mittels der Simulation ohne weitere möglich, das Verhalten eines Softwaresystems beim Zugriff von 100.000 Nutzern zu analysieren. In der Praxis ist es im Allgemeinen schwer bis gar nicht möglich, eine solche Zahl an Testnutzern zu akquirieren. Auch Szenarien wie der Ausfall eines Teiles der Hardware können auf diese Weise dargestellt werden.

- **Zeitraffer** – Mittels Simulation besteht die Möglichkeit, die simulierte Zeit deutlich zu beschleunigen. So können Prozesse, die in der Realität Jahre oder Jahrhunderte benötigen, in der Simulation innerhalb eines kurzen Zeitraums durchgespielt werden. In der Software-Performance-Analyse werden selten Zeiträume von Jahrhunderten betrachtet, doch die Simulation des Nutzerverhaltens über den Zeitraum von einem oder zwei Jahren ist für die Betrachtung des Systemverhaltens oft von Bedeutung.

- **Reduktion der Kosten für Untersuchungen** – Der Nach- oder Umbau eines Softwaresystems zur Durchführung einer Untersuchung verursacht erhebliche Kosten durch Hardware, Software und Administration. Bei der Simulation entstehen zwar ebenfalls Kosten für die Erstellung der Simulationsmodelle, diese sind jedoch im Allgemeinen niedriger, insbesondere, wenn mehrere verschiedene Untersuchungen am Stück durchgeführt werden.

- **Wiederholbarkeit** – Sind die Simulationsmodelle für ein Softwaresystem einmal erstellt, so kann die Simulation ohne großen Aufwand beliebig oft wiederholt werden. Werden Analysen über längere Zeiträume hinweg durchgeführt, so können die erstellten Simulationsmodelle archiviert und jederzeit reaktiviert werden. Die Durchführung einer weiteren Simulation nach der Erstellung der Simulationsmodelle ist mit geringem Aufwand jederzeit möglich.

Die genannten Vorteile der Simulation sind mit dieser Liste nicht abgeschlossen, sondern bilden eine Auswahl, die speziell auf die Simulation der Software-Performance gerichtet ist.

Allerdings bringt der Einsatz von Simulation zur Performance-Analyse auch einige Herausforderungen und Nachteile mit sich:

- **Hoher Aufwand in der Erstellung des Simulationsmodells** – Die Erstellung eines Simulationsmodells für die Performance-Analyse einer Unternehmensanwendung

erfordert in den gängigen Simulationsansätzen hohen manuellen Aufwand (Kappler et al. 2008; Brosig et al. 2009). Erste Ansätze zur automatischen Generierung zumindest eines Teils des Simulationsmodells befinden sich in der Entwicklung (z.B. (Chen et al. 2005; Brosig et al. 2011)), dennoch müssen auch diese mit hohem Aufwand manuell ergänzt und verfeinert werden.

- **Auswahl einer passenden Granularität der modellierten Komponenten** – Die Auswahl einer passenden Detailtiefe bei der Modellierung eines komplexen Systems ist nach Jain (Jain 1991) eine der größten Herausforderungen. Bildet das Simulationsmodell das zu analysierende System zu grobgranular ab, so werden die Komponenten des simulierten Systems und ihre Relationen nicht genau genug repräsentiert, da zu viele Annahmen getroffen werden müssen. Bildet das Simulationsmodell das betrachtete System hingegen zu feingranular ab, so erhöhen sich die Modellentwicklungszeit, die Fehlerrate und der Aufwand für die Validierung des Modells. Weiterhin erhöht eine zu feine Granularität die benötigten Ressourcen und die Zeit zur Durchführung der Simulation insbesondere großer Systeme.

- **Auswahl der korrekten Eingabe- und Umgebungsparameter** – Ist eine zu analysierende Unternehmensanwendung korrekt und in passender Granularität in einem Simulationsmodell abgebildet, so beeinflusst die Auswahl der Eingabe- und Umgebungsparameter maßgeblich die Korrektheit des Simulationsergebnisses. Eingabeparameter werden zumeist als Workload definiert: der Workload gibt an, wie viele Anwender wann mit welchen Eingabeparametern auf welche Funktion des betrachteten Systems zugreifen. Hierbei ist sicherzustellen, dass der Workload der realen Nutzungssituation möglichst nahe kommt, da eine Simulation unter realitätsfernen Bedingungen auch voraussichtlich realitätsferne Ergebnisse liefert. Ebendies gilt auch für die Umgebungsparameter. Dies sind beispielsweise simulierte Datenbankfüllstände, Netzwerklasten und Hardwareumgebungen.

- **Validierung des Simulationsmodells** – Ein Simulationsmodell kann auf zwei Weisen falsch sein. Die erste Weise ist das, was man bei der Programmierung einen „Bug“ nennt: eines oder mehrere der zahlreichen Sub-Modelle oder deren Kommunikation wurde fehlerhaft abgebildet. Da die meisten Elemente eines Simulationsmodells manuell angelegt werden ist dies häufig der Fall. Die zweite Weise ist zumeist deutlich gravierender – das Modell wurde korrekt erstellt, doch es entspricht nicht dem realen, zu simulierenden System. Ein solcher Fall tritt beispielsweise dann auf, wenn Messwerte falsch oder nicht im erforderlichen Umfang

erhoben wurden, oder wenn fälschlicherweise Standard-Verteilungsfunktionen für Komponentenverhalten zum Einsatz kamen. Eine Validierung des Simulationsmodells ist notwendig, um diese beiden Fälle auszuschließen. Da zum heutigen Stand keine automatisierten Validierungsverfahren für Performance-Simulationsmodelle vorliegen ist mit der Validierung ein erheblicher manueller Aufwand verbunden. Oft ist eine komplette Validierung zudem nicht möglich, beispielsweise, wenn sich das betrachtete System noch in der Design- oder Entwicklungsphase befindet.

- **Vertrauen in die Ergebnisse** – Ingenieure glauben das, was sie mit ihren eigenen Augen sehen (Ludewig/Lichter 2007a). Simulationsergebnisse haben deshalb oft den Ruf, realitätsfern zu sein. Eine umfangreiche und nachvollziehbare Validierung des Simulationsmodells bildet eine gute Basis für die Schaffung von Vertrauen in die Simulationsergebnisse. Dennoch ist oft gutes zwischenmenschliches Geschick notwendig, um Skeptiker von der Validität von Simulationsergebnissen zu überzeugen – insbesondere dann, wenn diese unerfreuliche Aussagen (wie beispielsweise eine schlechte Performance eines neu entwickelten Softwaresystems) enthalten.

Wann immer die Komplexität eines Systems eine gewisse Größe erreicht oder die Kosten, Gefahren oder Auswirkungen eine gewisse Dimension annehmen, überwiegen die Vorteile der Simulation die Nachteile. Bei der Performance-Analyse von Unternehmensanwendungen ist beides gegeben. Die Komplexität und die mit der Unternehmensanwendung verbundenen Kosten verhindern andere Maßnahmen zur Sicherstellung der Performance, wie beispielsweise vollständige Lasttests, so dass mithilfe konventioneller Verfahren wie dem Last- und Stresstest nur stichprobenartig geprüft werden kann. Auch das Verändern der Unternehmensanwendungen zu Analysezwecken ist aus Gründen der Sicherstellung des produktiven Betriebs in den meisten Fällen nicht möglich. Simulation ist, ergänzend zu den stichprobenartigen Lasttests, damit das Mittel der Wahl zur Analyse des Performance-Verhaltens einer Unternehmensanwendung.

2.2.2 Simulationsansätze

(Jain 1991) folgend gibt es vier Typen von Simulation, die für eine wissenschaftliche Arbeit im Bereich der Computerwissenschaften in Frage kommen:

- **Emulation** – Eine Simulation, die Hardware oder Firmware abbildet. Ein Betriebssystem-Emulator simuliert einen Typ von Betriebssystem auf einem anderen

(zum Beispiel Wine, (WineHQ.org 2012)), ein Prozessor-Emulator simuliert ein Instruction Set eines Prozessors auf einem anderen.

- **Monte-Carlo-Simulation** – Eine statistische Simulation oder eine Simulation ohne Zeitachse wird Monte-Carlo-Simulation genannt. Monte-Carlo-Simulation wird für die Abbildung von Wahrscheinlichkeitsphänomenen eingesetzt, die über die Zeit konstant bleiben. Die Monte-Carlo-Simulation wird zudem eingesetzt, um nicht wahrscheinlichkeitstheoretische Probleme (primär der Analysis) mit den Methoden der Wahrscheinlichkeitstheorie zu lösen.

- **Trace-Driven-Simulation** – Eine Trace-Driven-Simulation ist eine Simulation, die einen Trace als Eingabewert hat. Ein Trace ist eine zeitlich geordnete Menge von Ereignissen auf einem Realsystem. Trace-Driven-Simulation wird zumeist eingesetzt, um Computersysteme zur Laufzeit zu tunen oder zu konfigurieren. Speicherverwaltungsalgorithmen, CPU-Scheduling-Algorithmen, Cache-Analysealgorithmen und Algorithmen zur dynamischen Allokation von Festplattenspeicher sind klassische Beispiele für den Einsatz von Trace-Driven-Simulation.

- **Discrete-Event-Simulation** – Eine Simulation, die ausschließlich auf einem Modell diskreter Ereignisse und einem diskreten Status eines Systems basiert, nennt man Discrete-Event-Simulation. Die Discrete-Event-Simulation steht im Gegensatz zur Continuous-Event-Simulation, in der die Zustandsübergänge kontinuierlich erfolgen und die oft in der Chemie angewandt wird. In der Computerwissenschaft findet die Discrete-Event-Simulation ihre Anwendung, da sich Computersysteme aufgrund ihrer Struktur ausschließlich in diskreten Schritten verändern können.

In allen gängigen Performance-Simulationsansätzen (vgl. Teilkapitel 3.3) findet die Discrete-Event-Simulation Anwendung.

2.2.3 Ablauf des Simulationsprozesses

(Bungartz et al. 2009) definieren den Begriff „Simulation“ auf zwei Arten. Im weiteren Sinn ist Simulation der „Gesamtkomplex der Vorausberechnung oder des Nachstellens eines bestimmten Szenarios“. Dieser Gesamtkomplex setzt sich aus sechs Schritten zusammen, den die Autoren als die „Simulationspipeline“ bezeichnen (siehe Abbildung 2.1).

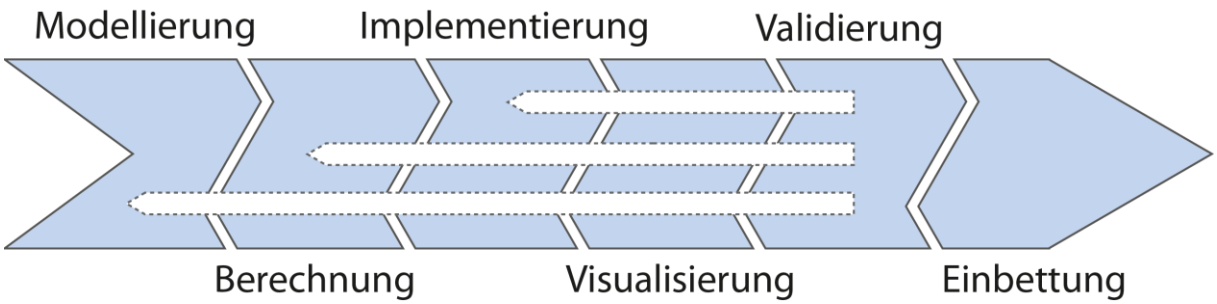


Abbildung 2.1. Die Simulationspipeline. Quelle: Eigene Darstellung, nach (Bungartz et al. 2009).

- **Modellierung** – Ergebnis der Modellierung ist das Simulationsmodell. Dieses umfasst eine vereinfachte formale Beschreibung eines geeigneten Ausschnitts des betrachteten Systems.

- **Berechnung** – Die Simulation im engeren Sinne. Im Zuge der Berechnung wird das Simulationsmodell geeignet aufbereitet (z.B. diskretisiert), und effiziente Algorithmen werden ermittelt, um das Modell zu lösen.

- **Implementierung** – Im Rahmen der Implementierung erfolgt die Umsetzung der vorab ermittelten Algorithmen in Programmcode.

- **Visualisierung** – Die Visualisierung umfasst nicht nur die Darstellung der Simulationsergebnisse, sondern insbesondere auch deren Interpretation.

- **Validierung** – Die Simulation im weiteren Sinn umfasst zahlreiche potentielle Fehlerquellen, wie beispielsweise das Simulationsmodell, die gewählten Algorithmen oder die Interpretation der Ergebnisse. Eine Validierung der Simulationsergebnisse, beispielsweise durch einen Vergleich gegen andere Modelle oder Experimentergebnisse, ist zwingend notwendig, um die Ergebnisse belastbar und brauchbar zu machen.

- **Einbettung** – Simulation erfolgt im Allgemeinen nicht zum Selbstzweck, sondern ist in einen Kontext, beispielsweise in ein Entwicklungs- oder Produktionsprojekt, eingebettet. Zu diesem Kontext müssen Schnittstellen geschaffen werden, um sowohl die benötigten Eingabedaten in die Simulation als auch die Simulationsergebnisse zurück fließen lassen zu können.

Aus der Validierung heraus gibt es, ähnlich dem Software Engineering, Rückschlüsse auf alle vorhergehenden Schritte. Hierdurch ergibt sich, dass einige Schritte der Simulationspipeline mehrfach durchlaufen werden.

Die existierenden Simulationsansätze für Unternehmensanwendungen unterstützen die Schritte Modellierung, Berechnung, Implementierung und teilweise Visualisierung. Hierbei erfolgt die Implementierung vollautomatisiert (aus der Modellierung heraus), während die Schritte Modellierung, Berechnung und Visualisierung werkzeugunterstützt von Anwendern durchgeführt werden.

Im engeren Sinn bezeichnet die Simulation lediglich den zentralen Schritt, die Berechnung des Simulationsergebnisses. In dieser Arbeit werden beide Bedeutungen parallel genutzt, wobei auf eine Differenzierung verzichtet wird, soweit die Bedeutung implizit klar ist.

2.2.4 Ziele der Simulation

Ziel der Simulation des Performanceverhaltens eines Softwaresystems ist immer die Voraussage des Performance- und Skalierungsverhaltens des betrachteten Systems unter Bedingungen, die in der Realität (zu einem gegebenen Zeitpunkt) nicht betrachtbar sind. Für das Fehlen des realen Beobachtungsobjekts kann es verschiedene Gründe geben:

- **Das System ist noch nicht (fertig) implementiert** – Das zu betrachtende System befindet sich noch in der Entwicklung und ist noch nicht (vollständig) betriebsbereit.

- **Eine Veränderung kann am existierenden System nicht durchgeführt werden** – Die Auswirkung einer Hardware- oder Softwareänderung auf ein Produktivsystem ist nicht vollständig vorhersagbar. Die Simulation soll die Auswirkungen der Änderung vorwegnehmen, so dass entschieden werden kann, ob die Änderung durchgeführt wird.

- **Das System kann nicht unter Hochlast gesetzt werden** – Ein Produktivsystem kann oft nicht einfach unter Hochlast gesetzt werden, um seine Lastgrenzen auszuloten. Hierdurch würde es eben in den Zustand versetzt, welcher durch die Ermittlung der Lastgrenzen vermieden werden soll. Zudem ist es bei großen Systemen schwer möglich, künstlich ausreichend Last zu erzeugen, um die Systeme an ihre Grenzen zu bringen.

Aufgrund dieser Einschränkungen ist das Performance-Verhalten insbesondere großer Unternehmensanwendungen oft weitgehend unbekannt.

Das vorab genannte Ziel der Simulation lässt sich detaillieren in mehrere separate Ziele, welche stark vom Stand des betrachteten Systems im Softwarelebenszyklus abhängen. Die folgenden Abschnitte erläutern diese Ziele.

2.2.4.1 Architektur- und Testunterstützung zur Entwicklungszeit

Das simulierte System befindet sich zum Zeitpunkt der Simulation noch in der Entwicklungsphase. Dies kann bedeuten, dass es vollständig oder teilweise noch nicht in ausführbarer Form vorliegt. Zumindest ein (möglichst vollständiger) Entwurf ist jedoch Voraussetzung für eine Simulation zur Entwicklungszeit. Eine Betrachtung zur Entwicklungszeit kann hilfreich sein, um Performanceschwachstellen in der Architektur frühzeitig zu entdecken und zu beseitigen. Zudem erlaubt die Simulation eine Identifikation von potentiellen Bottlenecks und performanceunkritischen Komponenten, wodurch eine Fokussierung von Last- und Performance-Tests ermöglicht wird.

2.2.4.2 Betrachtung der Auswirkungen von verändertem Anwenderverhalten

Das zu simulierende Softwaresystem existiert bereits und ist gegebenenfalls auch schon im produktiven Einsatz. Nun soll simuliert werden, wie sich das System verhält, wenn sich das Zugriffsverhalten der Anwender ändert. Zum einen kann diese Veränderung des Anwenderverhaltens die Anzahl an Anwendern betreffen, die gleichzeitig oder mit geringer zeitlicher Verzögerung auf das System zugreifen, zum anderen kann es auch die Zugriffspfade und Zugriffshäufigkeiten der bestehenden Anwender betreffen. Durch die Simulation können Aussagen über Lastobergrenzen für das betrachtete System getroffen werden, welche insbesondere bei sprunghaften Anstiegen der Nutzerzahlen, wie sie beispielsweise bei Firmenzusammenführungen entstehen, oder bei starken Veränderungen des Zugriffsverhaltens, beispielsweise durch Enterprise Application Integration-Projekte (EAI), relevant werden.

2.2.4.3 Betrachtung der Auswirkungen von Softwareänderungen

Im Laufe des Softwarelebenszyklus wird ein Softwaresystem durch Wartungs- und Erweiterungsarbeiten verändert. Bei einer großen Unternehmensanwendung betreffen diese Änderungen zumeist nicht das Gesamtsystem, sondern einzelne Komponenten. Dennoch können Änderungen an einzelnen Komponenten Auswirkungen auf die Performance des Gesamtsystems haben. Dies ist beispielsweise dann der Fall, wenn die Änderung an der Komponente eine Veränderung des Nutzungsverhaltens dieser Komponente auf einer gemeinsam genutzten Ressource (wie beispielsweise einer CPU oder einem Pool von

Datenbankverbindungen) zur Folge hat. Im Rahmen der Simulation erfolgt eine Vorhersage des Performanceverhaltens des Gesamtsystems unter Berücksichtigung der veränderten Komponente, um noch vor Einbindung dieser veränderten Komponente etwaige Performance-Probleme kompensieren zu können.

2.2.4.4 Betrachtung der Auswirkungen von Hardwareänderungen

Günstige Hardwarepreise führten in den vergangenen Jahren zu einer massiven Hardwareunterfütterung von Unternehmensanwendungen, um Performanceproblemen vorzubeugen. Dieses Vorgehen basiert jedoch auf den zwei irrigen Annahmen, dass jedes Performanceproblem mittels zusätzlicher Hardware gelöst werden kann, und dass eine mehr oder weniger gleichmäßige Verteilung der Hardware sinnvoll ist. (Wilhelm 2003) zeigt am Beispiel SAP, dass Unternehmen auf diese Weise einen Hardwareoverhead von bis zu 50 Prozent betreiben.

Mit der Simulation der Auswirkungen von Hardwareänderungen werden drei Ziele verfolgt. Zum einen unterstützen die Resultate das Upsizing – sie erlauben Aussagen darüber, an welchen Stellen neue Hardware eine Steigerung der Leistungsfähigkeit des betrachteten Softwaresystems bringt, und ob sich die Leistung überhaupt durch zusätzliche oder stärkere Hardware steigern lässt. Zum anderen unterstützt die Simulation jedoch auch das Downsizing. Oftmals besitzen Unternehmensanwendungen ein großes Einsparungspotential aufgrund durch fehlerhaftes Upsizing ineffizient eingesetzter Hardwareressourcen. Ein durch die Simulation unterstütztes Downsizing der Hardware nicht-performancekritischer Komponenten erlaubt oft eine Reduzierung der Betriebskosten. Zu guter Letzt unterstützt die Simulation auch Hardwaremigrationsprojekte, beispielsweise eine Migration auf eine neue Supercomputergeneration oder auf den Hardwarestack eines anderen Anbieters.

2.2.4.5 Betrachtung der Auswirkungen von Middleware-Änderungen

Unternehmensanwendungen basieren oft auf Standardanwendungen, sogenannter Middleware. Unter Middleware versteht man Anwendungen, die einen oder mehrere Basisdienste bereitstellen, selber jedoch nicht explizit eine Geschäftsaktivität oder einen Geschäftsprozess abbilden. Gängige Middleware-Systeme sind Application Server und Message Buses (siehe Abschnitt 2.6.2.4). Solche Middleware wird von verschiedenen namhaften Herstellern angeboten (siehe beispielsweise (IBM 2012) oder (Oracle 2012c)), zudem gibt es etablierte Open-Source-Systeme auf dem Markt, beispielsweise (Red Hat 2012).

Aus technischen, finanziellen oder unternehmenspolitischen Gründen kann für ein Unternehmen die Notwendigkeit entstehen, den Anbieter oder die Version der verwendeten

Middleware zu wechseln. Auch wenn die Middleware-Systeme dieselben Schnittstellen implementieren haben sie aufgrund ihrer internen Umsetzung oft ein vollständig unterschiedliches Performance-Verhalten. Ein Wechsel der Middleware, oft sogar bereits eine Rekonfiguration, hat eine erhebliche Auswirkung auf die darauf betriebene Unternehmensanwendung. Die Simulation unterstützt solche Middleware-Änderungen durch eine proaktive Identifikation möglicher Performanceprobleme, welche oft bereits durch Konfigurationsänderungen behoben werden können.

2.2.5 Modellbildung – wie tief bildet man das System ab?

Eine Simulation eines Softwaresystems kann immer maximal so genau sein wie das in der Simulation verwendete Modell der Komponenten, welche das System aufbauen. Durch zufällige Negierung zweier oder mehrerer Fehler ist zuweilen auch eine höhere Genauigkeit erzielbar, solch zufällige Effekte sind jedoch nicht planbar und dürfen deshalb auch nicht berücksichtigt werden.

Der Detailgrad, zu welchem das betrachtete System nachgebildet werden muss, hängt stark von den unter 2.2.4 beschriebenen Zielen der Simulation ab. Ist eine Schwachstellenuntersuchung durchzuführen, beispielsweise zur Unterstützung der Testkonzeption wie unter 2.2.4.1 beschrieben, so ist zumeist eine Modellierung des Komponentenverhaltens mittels der Angabe eines durchschnittlichen oder maximalen Antwortzeitverhalten ausreichend. Ein solcher Detailgrad reicht offensichtlich nicht, um eine Betrachtung der Auswirkungen von Hardwareveränderungen wie unter 2.2.4.4 beschrieben vorauszusagen. Eine detaillierte Modellierung der Hardwarenutzung jeder einzelnen Operation jeder Komponente ist hier unumgänglich.

Der Detailgrad, zu welchem ein Systemverhalten modelliert wird, hat einen großen Einfluss auf den für die Simulation zu betreibenden Aufwand. Für eine Skalierungsbetrachtung auf Basis des Ressourcennutzungsverhaltens jeder Operation bedarf es einer möglichst exakten Modellierung der Ressourcennutzung jeder Operation unter allen Kombinationen von möglichen Einflussfaktoren (insbesondere der Typen von Eingaben sowie der Korrelationen zu anderen Operationen bzw. Komponenten). Eine solche Modellierung eines komplexen Systems erzeugt ohne weiteres einen Aufwand von mehreren Personenmonaten.

Weiterhin ist der Grad der Modellierungsgenauigkeit bei Unternehmensanwendungen stark abhängig von der Verfügbarkeit bzw. Messbarkeit der benötigten Informationen. Der Zugriff auf produktiv betriebene Unternehmensanwendungen für Messungen ist in Unternehmen im Allgemeinen aus Sicherheitsgründen stark eingeschränkt, produktivnahe Mess- und

Testumgebungen stehen aus Kostengründen oft gar nicht oder nur stark begrenzt zu Verfügung. Bei der Planung der Simulation ist deshalb zwingend zu berücksichtigen, welche Performancedaten überhaupt erlangt werden können.

Die Entscheidung der Detaillierung des Simulationsmodells ist demnach eine kritische Abwägung und stark abhängig vom Ziel der Simulation, dem vertretbaren Aufwand sowie der Datenverfügbarkeit. Sie bereits zu Beginn der Simulationsplanung zu treffen ist damit zwingend notwendig für eine erfolgreiche Planung und Durchführung der Simulation.

2.2.6 Voraussetzungen für eine erfolgreiche Simulation

Um eine Simulation erfolgreich durchführen zu können müssen einige Voraussetzungen erfolgt sein. Negiert man Jains „Common Mistakes in Simulation“ (Jain 1991), so lassen sich daraus folgende notwendigen Voraussetzungen ableiten:

- **Verfügbarkeit eines adäquaten Simulations-Frameworks** – Ein Simulations-Framework unterstützt die Modellerstellung und führt die Simulation auf dem Modell aus. Ein adäquates Simulations-Framework erlaubt eine effiziente Modellierung und liefert die benötigten Resultate in akzeptabler Zeit und unter der Verwendung akzeptabler Ressourcen.

- **Ausreichend Zeit und Ressourcen** – Wie bereits in Teilkapitel 2.2.1 ausgeführt bedarf die Simulation in ihrem gesamten Prozess eine große Menge an Zeit und Ressourcen. Stehen diese nicht zu Verfügung, so besteht die Gefahr, dass das Simulationsprojekt nicht, oder, meist noch schlimmer, nur teilweise durchgeführt wird. Ähnlich der Softwareentwicklung leidet hierunter zumeist die Validierung, so dass zwar Simulationsergebnisse gewonnen werden, deren Zuverlässigkeit jedoch nicht sichergestellt wird.

- **Klar definiertes Ziel** – Um mittels Simulation ein verlässliches Ergebnis zu erlangen muss das mit der Simulation verfolgte Ziel klar definiert werden. Die Aussage, dass die „Performance der Anwendung XY“ simuliert werden soll ist keine klare Zieldefinition, zahlreiche Fragen sind offen: Was versteht man unter Performance? Unter welchen Rahmenbedingungen soll XY untersucht werden? Jain definiert, dass die durch die Simulation verfolgten Ziele dem SMART-Prinzip (Doran 1981) folgen sollen, demnach spezifisch, messbar, akzeptiert, realistisch und terminierbar sein müssen. Ein klar definiertes Ziel wäre beispielsweise die Simulation des

Antwortzeitverhaltens der Anwendung XY bei 1.000 parallelen Nutzerzugriffen auf die Funktion Z, wenn alle Rahmenbedingungen gleich bleiben.

- **Ausreichend Erfahrung** – Wie bereits ausgeführt erfordert die Simulation einer Unternehmensanwendung zahlreiche manuelle Schritte. Diese Schritte, wie das Erstellen des Simulationsmodells und dessen Validierung, erfordern umfangreiche Kenntnisse der Simulationsmethodik, aber auch der Softwarearchitektur und des zu analysierenden Systems. Jain folgend benötigt man für ein erfolgreiches Simulationsprojekt Erfahrung in vier Bereichen: Projektleitung, Modellierung und Statistik, Programmierung und Wissen über das zu simulierende System. Zur erfolgreichen Durchführung des Simulationsprojekts müssen all diese Erfahrungen im Team verfügbar sein. Geleitet wird das Team nach Jain am besten von demjenigen, der sich in allen vier Bereichen auskennt.

- **Mitarbeit der Anwender** – Sind die Mitglieder des Simulationsteams nicht zugleich die Nutzer des zu simulierenden Systems (was oft nicht der Fall ist), so ist eine regelmäßige Abstimmung des Simulationsmodells mit den Anwendern zwingend notwendig. Dokumentationen des untersuchten Systems wie beispielsweise Architekturdokumente geben im Allgemeinen einen guten Überblick, doch nur die Abstimmung mit Anwendern, Systemarchitekten und Komponentenverantwortlichen erlaubt die vollständige und korrekte Modellierung einer großen Unternehmensanwendung. Zudem sind solche Systeme kontinuierlichen Veränderungen ausgesetzt (Heineman/Councill 2001). Diese Veränderungen sind oft nicht dokumentiert und werden erst durch eine Auseinandersetzung mit den technischen und operativen Nutzern des Systems enthüllt. Letztendlich werden die Anwender benötigt, um ein realistisches Nutzerverhalten abzubilden.

Die vorab genannten Punkte lassen sich um weitere, aus der praktischen Erfahrung gewonnene Voraussetzungen ergänzen:

- **Verfügbarkeit von aktuellen Architekturinformationen des betrachteten Systems** – Die Analyse des Aufbaus des untersuchten Systems kann aus Gründen des Aufwands nicht rein durch die Befragung von Anwendern erfolgen. Zur Erstellung des Simulationsmodells müssen Architekturinformationen in Form von Architekturskizzen und Dokumentation vorliegen.

- **Verfügbarkeit von aktuellen Messwerten der Systemkomponenten** – Um das Performanceverhalten der Softwarekomponenten abbilden zu können werden (aktuelle) Messwerte des Ressourcennutzungs- oder Antwortzeitverhaltens der Komponenten benötigt. Diese Messwerte werden entweder aus Last- und Performancetests oder aus Monitoring- und Log-Informationen der Komponenten ermittelt.

- **Verfügbarkeit von Nutzungsinformationen** – Informationen über das Nutzerverhalten erlauben die Simulation eines realitätsnahen Anwendungsfalls. Hierbei ist zum einen die Anzahl der parallelen Nutzerzugriffe und ihre Verteilung über den simulierten Zeitraum hinweg relevant, zum anderen aber auch das Nutzungsverhalten der einzelnen Nutzer auf dem System, also die aufgerufenen Funktionen, die Think Times (also die Zeit, die ein Nutzer benötigt, um nachzudenken oder externe Aufgaben zu erfüllen) sowie die Prozesse, die der Nutzer auf dem System ausführt. Diese Informationen können zum einen, soweit vorhanden, durch statistische Analyse von Log-Informationen aus den vergangenen Prozessdurchläufen gewonnen werden, zum anderen durch Interviews mit den Fachabteilungen.

- **Verfügbarkeit von Kontextinformationen** – Kontextinformationen sind Informationen über den Zustand und die Umgebung des betrachteten Systems. Dies sind beispielsweise Füllstände von Datenbanken, aber auch Informationen darüber, welche anderen Systeme parallel auf der gleichen Hardware ausgeführt werden wie das zu simulierende System. Weiterhin fallen unter die Kontextinformationen Latenzen von Netzwerk- und Middleware-Komponenten. Die Relevanz des Kontexts wird bei der Performance-Analyse häufig unterschätzt und führt dann zu starken Abweichungen der Simulationsergebnisse zum Produktivsystem, beispielsweise weil dort eine Datenbank parallel von mehreren, in der Simulation nicht beachteten, Anwendungen verwendet wird.

All die genannten Voraussetzungen müssen erfüllt sein, um verlässliche Simulationsergebnisse zu erhalten. Einige Voraussetzungen können kompensiert werden – beispielsweise lässt sich das Fehlen von Architekturdokumenten oder Beschreibungen von Nutzerverhalten durch Interviews ausgleichen. Generell ist dabei jedoch mit erheblichem Aufwand zu rechnen. Das Erlangen von validen Messwerten zur Modellbildung hat sich in der Praxis als schwerwiegende Herausforderung herausgestellt, da oft organisatorische Schranken überwunden werden müssen.

2.2.7 Limitationen der Simulation

Die Zuverlässigkeit der Simulationsergebnisse ist beschränkt durch die verwendeten Komponentenmodelle. Ein Simulationsergebnis macht nur solange verlässliche Aussagen, wie die Ober- und Untergrenzen der Komponentenmodelle nicht über- bzw. unterschritten wurden (siehe hierzu Teilkapitel 2.3.2). Verlässt die Simulation bei nur einer Komponente den validierten Modellraum, so ist eine Verfälschung des Simulationsergebnisses nicht auszuschließen. Die Bedeutung der Verletzung einer Komponentenmodellgrenze ist abhängig von der Größe des Übertritts sowie von der vermuteten möglichen Abweichung. Letztere ist im Allgemeinen nicht zu bestimmen, so dass eine Verletzung der Komponentenmodellgrenzen möglichst zu vermeiden ist.

Alle betrachteten Simulationsmodelle (vgl. Teilkapitel 3.3) erlauben die Simulation über die validierten Komponentenmodellgrenzen hinaus. Je nach Implementierung wird das Modell entweder extrapoliert, oder der Grenzwert wird für alle über die Grenze hinaus abgerufenen Werte angenommen. Beide Lösungen sind problematisch – nimmt man beispielsweise die Dimension „Anzahl paralleler Zugriffe“, die sich auf die Antwortzeit einer Komponente auswirkt. Besitzt diese eine konstante Obergrenze, so bedeutet dies für die Komponente, dass sie, ihrem Modell folgend, bis zu dieser Obergrenze eine (linear, quadratisch oder exponentiell) steigende Antwortzeit in Abhängigkeit der Anzahl paralleler Aufrufe hat, ab dem Grenzpunkt jedoch eine konstante Antwortzeit behält. Nimmt man jedoch eine extrapolierte Antwortzeitkurve an, und verläuft die Kurve bis zu dem Grenzwert beispielsweise linear steigend, so impliziert das extrapolierte Modell, dass sich dieses lineare Antwortzeitverhalten bis ins Unendliche so fortsetzt. (Bögelsack 2011) zeigt, dass beide Modelle falsch liegen.

Da die gängigen Performance-Simulationsframeworks die Grenzwerte der Komponentenmodelle nicht berücksichtigen ist es notwendig, diese Prüfung manuell durchzuführen, bevor die Simulationsergebnisse verwendet werden. Eine Betrachtung der Anwendungs-Performance über die Komponentenmodellgrenzen hinweg ist durch Simulation nicht möglich.

2.2.8 Zusammenfassung

Die vorherigen Abschnitte zeigen die grundlegenden Prinzipien der Performance-Analyse mittels Simulation. Im ersten Abschnitt wird gezeigt, dass die Simulation zur Untersuchung komplexer Softwaresysteme wie beispielsweise Unternehmensanwendungen zahlreiche Vorteile mit sich bringt, wie die Möglichkeit der Abstraktion und die Sub-Modellierung zur Reduktion der Komplexität, aber auch die Nachstellbarkeit nicht am Realsystem messbarer Situationen. Dem gegenüber gestellt werden die Nachteile der Simulation, insbesondere der

hohe Aufwand zur Erstellung der Simulationsmodelle und das benötigte Wissen zur Sicherstellung der Korrektheit. Die Motivation zur Nutzung der Simulation erfolgt durch die Darstellung, weshalb gerade bei komplexen Unternehmensanwendungen die Vorteile überwiegen.

Im nächsten Schritt folgt die Vorstellung verschiedener Arten von Simulationen sowie des Ablaufs der für die Performance-Analyse hauptsächlich eingesetzten Discrete-Event-Simulation. In diesem Abschnitt wird auf die Simulationspipeline von (Bungartz et al. 2009) eingegangen, die die Simulation im Großen wie im Kleinen beschreibt. Fortgeführt wird die Beschreibung der Simulation durch die Darstellung möglicher Simulationsziele und der Ausführung, dass eine detaillierte Definition der Ziele eines Simulationslaufs bereits vor dem Lauf wichtig ist, um den Aufwand für die Simulation auf das Nötigste zu begrenzen.

Auf die Ziele folgend wird betrachtet, wie tief ein simuliertes System abgebildet werden muss. In diesem Abschnitt wird ausgeführt, dass die Granularität des Simulationsmodells stark vom durch die Simulation angestrebten Ziel abhängt. Umgekehrt sind auch nur die Ziele durch die Simulation erreichbar, die im Simulationsmodell abgebildet werden können. An dieser Stelle geht der Abschnitt nahtlos in den folgenden über, in dem die Voraussetzungen für eine erfolgreiche Simulation, wie die Verfügbarkeit eines geeigneten Simulations-Frameworks, ausreichend Zeit und Ressourcen und die Mitarbeit der Anwender vorgestellt werden.

Abgeschlossen wird das Teilkapitel durch die Aufführung von Limitationen der Simulation. In diesem Abschnitt wird aufgeführt, dass Komponentenmodelle nur in dem Rahmen valide Aussagen liefern, in dem ihnen Messwerte zugrunde liegen. Verlässt ein in der Simulation verwendetes Komponentenmodell den Bereich der zugrundeliegenden Messwerte, so gefährdet dies die Aussagekraft des gesamten Simulationsergebnisses. Existierende Simulations-Frameworks berücksichtigen diese Beschränkung nicht, so dass sie manuell geprüft werden muss. Die Beschäftigung mit Komponentenmodellen im letzten Abschnitt dieses Teilkapitels bildet die Überführung in das folgende Teilkapitel: Komponentenmodelle.

2.3 Komponentenmodelle

Komponentenmodelle bilden das Verhalten einer in der Simulation verwendeten Softwarekomponente abhängig von gegebenen Eingabedaten in einem definierten Kontext ab. Die vorliegende Arbeit konzentriert sich auf die Performance-Betrachtung, so dass es sich bei den in dieser Arbeit besprochenen Komponentenmodellen generell um Performancemodelle handelt, um Modelle also, die das Antwortzeitverhalten einer

Komponente beschreiben. Komponentenmodelle finden in der Simulation ihren Einsatz, um ein detailliertes Verhalten einer für das Ergebnis relevanten Komponente darzustellen. Vorgaben zum Detaillierungsgrad des Simulationsmodells sind in Teilkapitel 2.2.5 getroffen.

2.3.1 Aussage eines Komponentenmodells

Ein Komponentenmodell beschreibt das Antwortzeitverhalten einer Komponente unter gegebenen Rahmenbedingungen und Eingabewerten. Softwarekomponenten sind im Allgemeinen komplex, so dass Komponentenmodelle eine Approximation des tatsächlichen Verhaltens darstellen. Das Komponentenmodell liefert unter gegebenen Eingabedaten, wie beispielsweise die Anzahl der parallelen Zugriffe auf die Komponente, die aufgerufene Operation oder die Art des Aufrufs (lesend oder schreibend), eine zu erwartende Antwortzeit der Komponente zurück. Das Komponentenmodell berücksichtigt dabei Korrelationen zwischen den Eingabedaten.

2.3.2 Limitationen eines Komponentenmodells

Das Verhalten einer Softwarekomponente ist im Allgemeinen hoch komplex, die Menge möglicher Eingabewerte und Kontextzustände (wie beispielsweise der Zustand einer internen Datenbank oder die Bearbeitung paralleler Prozesse auf der CPU) so umfangreich, dass es nicht vollständig gemessen werden kann. Bei der Erstellung von Komponentenmodellen trifft man auf die gleichen Probleme wie beim Softwaretest – ein vollständiger Test bzw. eine vollständige Vermessung aller möglichen Kombinationen aus Eingabedaten und Kontextinformationen ist aus Gründen des Aufwands nicht möglich (Ludewig/Lichter 2007b). Das Set an Messwerten, aus dem die Komponentenmodelle erzeugt werden, ist demnach immer unvollständig, die Komponentenmodelle damit auch. Durch geeignete Modellierungs- und Interpolationsmaßnahmen ist diese Unvollständigkeit jedoch soweit reduzierbar, dass sie sich auf die Genauigkeit des Simulationsergebnisses nicht negativ auswirkt. Dieser Zustand ist erreicht, sobald die Modellierungsungenauigkeit unter den erwarteten Messfehler fällt.

2.3.2.1 Genauigkeit

Die Genauigkeit eines Komponentenmodells hängt zum einen von der Art der Generierung des Modells ab, zum anderen von den Daten, die dem Modell zugrunde liegen. Liegen dem Komponentenmodell wenige Messdaten zugrunde, so ist es möglich, ein Modell zu erzeugen, welches diese Messdaten exakt abbildet. Dieses Modell existiert zu jedem Datensatz und ist eindeutig. Als Beweis hierzu dient die numerische Mathematik, die zeigt, dass es zu jeden n+1 paarweise verschiedenen Datenpunkten exakt ein Interpolationspolynom n-ten Grades existiert, das an den vorgegebenen Stützstellen mit den vorgegebenen Stützwerten überein stimmt (Schwarz/Köckler 2004). Zur tatsächlichen

Durchführung dieses als Polynominterpolation bekannten Verfahrens liefert die numerische Mathematik zahlreiche Verfahren, beispielsweise die Lagrangesche Interpolationsformel (Munz/Westermann 2009) und den Newtonschen Interpolations-Algorithmus (Schwarz/Köckler 2004), auf die in dieser Arbeit nicht weiter eingegangen wird. Komponentenmodelle, die mittels der Polynominterpolation erstellt werden, haben keinen Modellfehler.

Es ist jedoch ersichtlich, dass die Polynominterpolation nur auf kleinen Datensätzen tatsächlich durchgeführt werden kann. Bereits auf Datensätzen von nur 50 Messwerten liefert die Polynominterpolation ein Interpolationspolynom vom Grad 49. Liegen mehrere hundert oder tausend Messwerte vor, so ist die Polynominterpolation nicht durchführbar, da die Komplexität der resultierenden Polynome ihre weitere Verwendung beispielsweise als Komponentenmodelle unmöglich macht. In diesem Fall kommen Approximationsalgorithmen zum Einsatz, welche die Komplexität der resultierenden Polynome reduzieren, indem sie Abweichungen von den Messwerten, also Modellfehler, akzeptieren.

In wieweit ein Modellfehler eines Komponentenmodells akzeptabel ist hängt stark von dem Einsatzgebiet des Komponentenmodells ab. Die Genauigkeit der zugrundeliegenden Messdaten erlaubt jedoch eine Definition der unteren Grenze der Genauigkeit, eines Grenzwerts, bis zu dem Modellfehler immer akzeptabel sind.

(Jehle 2010) sowie (Bögelsack 2011) zeigen, dass selbst bei der Messung der Performanceverhaltens von Unternehmensanwendungen in einer vollständig kontrollierten Umgebung Abweichungen von über fünf Prozent auftreten, teilweise sogar deutlich höhere. Wenn die der Komponentenmodellierung zugrunde liegenden Daten bereits Messfehler von fünf Prozent beinhalten, so ist ein Modellfehler von fünf Prozent in jedem Fall akzeptabel. Enthalten die Messdaten eine Abweichung von fünf Prozent, so wird ein Modellfehler bis zu fünf Prozent ebenso viele Messfehler negieren wie verstärken. Statistisch gesehen hat der Modellfehler damit keine negative Auswirkung auf das Ergebnis, er kann damit akzeptiert werden. Die Größe von fünf Prozent dient in dieser Arbeit als Grenzwert für ein gutes Modell, in der Praxis muss je nach Anwendungsfall diese Grenze höher oder niedriger gelegt werden.

2.3.2.2 Grenzen des Modellraums

Die Aussagen eines Komponentenmodells gelten immer nur für den Bereich an Eingabedaten und in dem Kontext, in dem Messdaten vorliegen, die zur Modellierung verwendet wurden. Dieser Bereich wird in dieser Arbeit als *Modellraum* bezeichnet. Der

Modellraum ist vieldimensional, jeder relevante Eingabe- oder Kontextwert bildet in ihm eine eigene Dimension.

Außerhalb dieses Modellraums liefert ein Komponentenmodell oft auch Ergebnisse – beispielsweise dann, wenn das Komponentenmodell in Form einer mathematischen Formel vorliegt (wie beispielsweise bei der Generierung durch einen genetischen Algorithmus, siehe Abschnitt 2.4). Da die außerhalb des Modellraums berechneten Ergebnisse keine Validierung mittels umliegender Messpunkte erfahren ist ihre Erzeugung, abhängig von der Entfernung zum letzten Messpunkt, nahezu beliebig. Abbildung 2.2 illustriert dies.

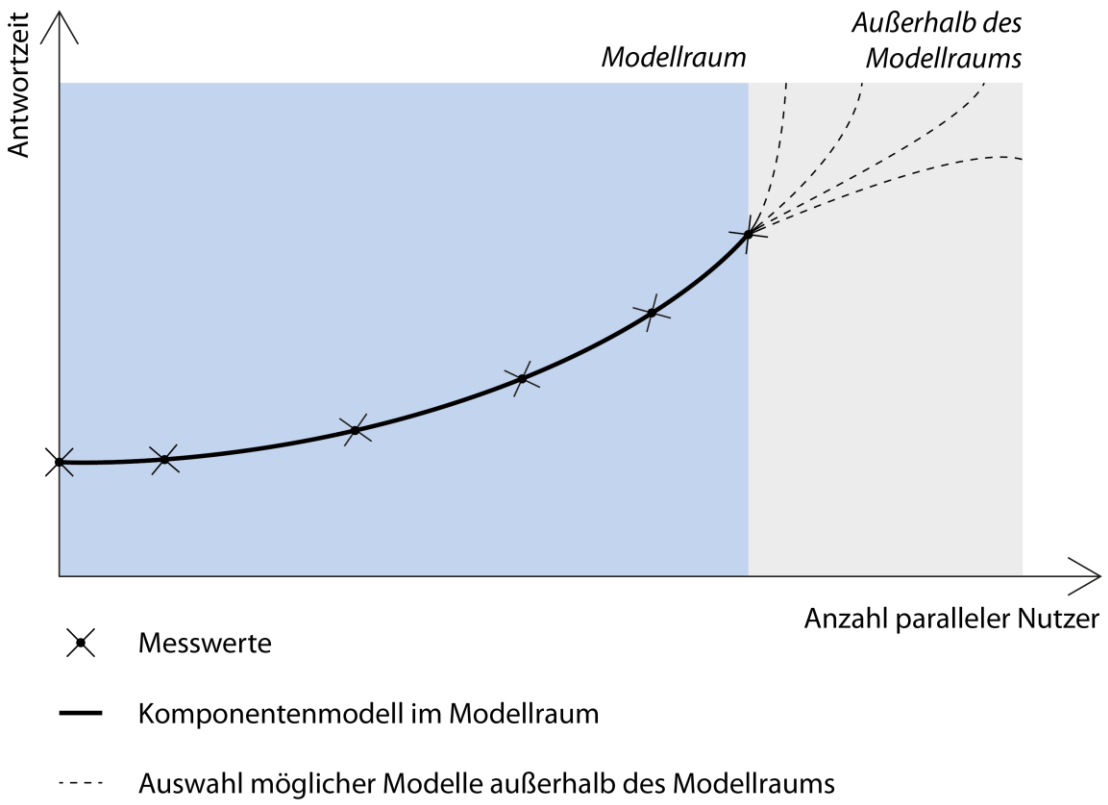


Abbildung 2.2. Komponentenmodelle in und außerhalb des Modellraums. Quelle: Eigene Darstellung.

Die Grenzen des Modellraums sind bei der Simulation demnach strikt einzuhalten. Ihr Überschreiten fügt dem Simulationsmodell einen Zufallsfaktor hinzu, welcher die Simulationsergebnisse möglicherweise unbrauchbar macht.

2.3.2.3 Erweiterung des Gültigkeitsbereichs des Modellraums

Im vorherigen Abschnitt wurde beschrieben, dass ein Überschreiten des Modellraums zwingend zu vermeiden ist, da das außerhalb des Modellraums modellierte Verhalten nahezu rein zufällig ist. Zuweilen ist es dennoch notwendig, ein Komponentenverhalten

außerhalb des messbaren Bereichs zu modellieren. Hierfür sollen an dieser Stelle ein Verfahren kurz vorgestellt werden, ohne dieses weiter auszuführen. Das Verfahren wird in dieser Arbeit nicht weiter betrachtet und findet auch keine Anwendung, seine Erwähnung dient ausschließlich der Vollständigkeit halber und als Inspiration für weitere Arbeiten.

2.3.2.3.1 Modellschablonen

Modellschablonen geben einen ungefähren Verlauf eines Komponentenmodells vor. Sie definieren Regeln, welche erfahrungsgemäß für alle Komponentenmodelle eines Typs gelten. Diese Regeln können genutzt werden, um das Antwortzeitverhaltens einer Komponente außerhalb des Modellraums vorauszusagen.

In den meisten Fällen sind diese Regeln nur für wenige Kontextfaktoren bestimmbar, selten für Eingabedaten. Diese Kontextfaktoren (wie beispielsweise die Anzahl der parallelen Zugriffe auf die Komponente) sind jedoch häufig genutzte und hoch relevante Einflussfaktoren im Komponentenmodell, und oftmals auch der in der Simulation betrachtete Performancefaktor (so dass die Fragestellung zur Simulation lautet: Wie verhält sich das System, wenn sich die Anzahl der parallelen Nutzer erhöht). Modellschablonen für ebensolche Kontextfaktoren ermöglichen die Nutzung der Komponentenmodelle auch über die Grenzen des Modellraums in den in den Modellschablonen erfassten Dimensionen heraus.

Die typische Struktur einer Modellschablone lässt sich am besten am Beispiel einer Modellschablone für das Antwortzeitverhalten und den Durchsatz einer Unternehmensanwendung zeigen. Abbildung 2.3 zeigt den Durchsatz und das Antwortzeitverhaltens einer gängigen Unternehmensanwendung abhängig von der Anzahl paralleler Anfragen.

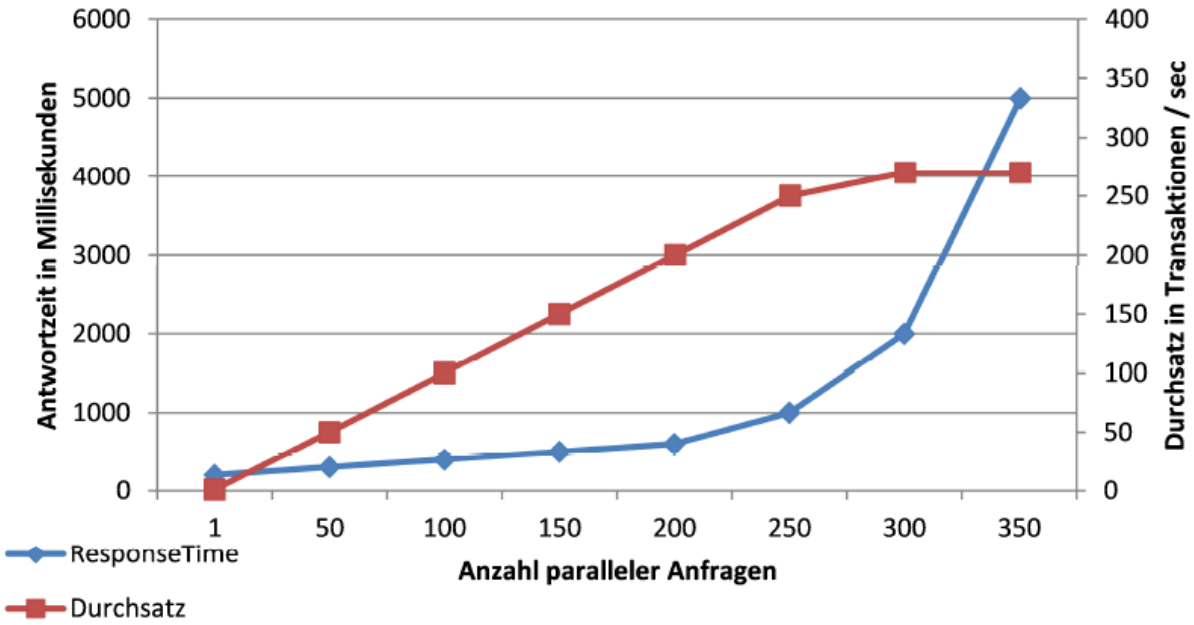


Abbildung 2.3. Durchsatz und Antwortzeitverhalten eines SAP ERP Systems abhängig von der Anzahl paralleler Anfragen. Quelle: (Gradl 2012)

Aus dieser Grafik lassen sich folgende Regeln für eine Modellschablone ableiten:

1. Die Antwortzeit steigt kontinuierlich mit zunehmender Anzahl paralleler Nutzer.

2. Der Durchsatz steigt oder bleibt konstant mit zunehmender Anzahl paralleler Nutzer.

3. Im Niedrig- und unteren Mittellastbereich steigen die Antwortzeit und der Durchsatz linear an.

4. Im Hochlastbereich konvergiert der Durchsatz gegen ein Maximum.

5. Im Hochlastbereich steigt die Antwortzeit exponentiell an.

Diese Regeln treffen erst einmal auf das betrachtete System zu. Inwieweit sie sich generalisieren lassen ist zu prüfen. Sie ermöglichen jedoch eine grundlegende strukturelle Validierung eines Komponentenmodells außerhalb des Bereichs der gemessenen Werte.

2.3.3 Zusammenfassung

Teilkapitel 2.3 führt die Komponentenmodelle ein. Komponentenmodelle beschreiben das Performance-Verhalten einer Softwarekomponente in Abhängigkeit von den Eingabedaten und dem Kontext, wie beispielsweise die Anzahl paralleler Zugriffe auf der Komponente. In

dieser Arbeit werden ausschließlich Antwortzeit-Komponentenmodelle betrachtet. Nach der Definition der Komponentenmodelle erfolgt eine Analyse ihrer Limitationen, die zeigt, dass ein Komponentenmodell nur Aussagen über den Bereich der Eingabedaten treffen kann, die durch Messwerte abgedeckt sind. Dieser Bereich wird in dieser Arbeit als Modellraum bezeichnet. Verlässt die Simulation den Modellraum, so sind die Aussagen des Komponentenmodells nicht mehr verlässlich. Zum Abschluss des Teilkapitels werden die Modellschablonen vorgestellt, ein Konzept, um den Modellraum eines Komponentenmodells zu erweitern. Die Modellschablonen werden jedoch ausschließlich als Konzept vorgestellt und in dieser Arbeit nicht weiter verfolgt.

2.4 Evolutionäre Algorithmen zur Modellapproximation

Der folgende Abschnitt erläutert die Grundlagen evolutionärer Algorithmen sowie ihre Anwendung zur Approximation mathematischer Modelle zur Abbildung des Performance-Verhaltens einer Softwarekomponente basierend auf Performance-Messwerten dieser Komponente. Im Detail wird auf genetische Algorithmen eingegangen, eine Variante der evolutionären Algorithmen, die in dieser Arbeit verwendet wird. Weiterhin erfolgt eine Untersuchung der Konfigurationsparameter des genetischen Algorithmus insbesondere auf deren Einfluss auf die Approximationsgenauigkeit. Diese beiden Faktoren sind kritisch für den Einsatz von genetischen Algorithmen zur Modellierung von Komponentenmodellen.

2.4.1 Grundlagen evolutionärer Algorithmen

Evolutionäre Algorithmen orientieren sich zur Lösung eines gegebenen Problems an dem von Charles Darwin (1859) beschriebenen Ablauf der biologischen Evolution. Ebenso wie die biologische Evolution verfolgt auch ein evolutionärer Algorithmus keinen linearen, direkten und „überlegten“ Weg zur Lösung eines Problems, sondern nutzt eine Population lernender (im Sinn von „sich anpassender“) und um gemeinsame Ressourcen konkurrierender Individuen, um mittels stochastischer Prozesse zu einer Lösungsmöglichkeit für das Problem zu gelangen. Evolutionäre Algorithmen fallen damit unter das maschinelle Lernen (engl. Machine Learning) und sind nichtdeterministisch.

Der Aufbau eines evolutionären Algorithmus erlaubt eine Lösungsfindung für ein Problem ohne vorherige Definition des Wegs zur Erlangung dieser Lösung. Hierdurch eröffnen sich gegebenenfalls neue Lösungsmöglichkeiten, die bislang für ein gegebenes Problem nicht in Betracht gezogen wurden, beispielsweise, weil sie stark von den etablierten Verfahren abweichen. Ein Beispiel für die Anwendung eines evolutionären Algorithmus zum Entwurf von Antennen, das genau diese Eigenschaft nutzt, ist in (Altshuler/Linden 1997) gegeben.

Evolutionäre Algorithmen durchsuchen den Lösungsraum eines gegebenen Problems mittels zweier Operationen: Mutation und Crossover. Der Crossover-Operator ermöglicht ein Springen durch den Lösungsraum, der Mutations-Operator hingegen eine asymptotische Annäherung an ein (möglicherweise lokales) Optimum. Abbildung 2.4 illustriert die Mutations- und Crossover-Operation als Bewegung im Lösungsraum eines Problems.

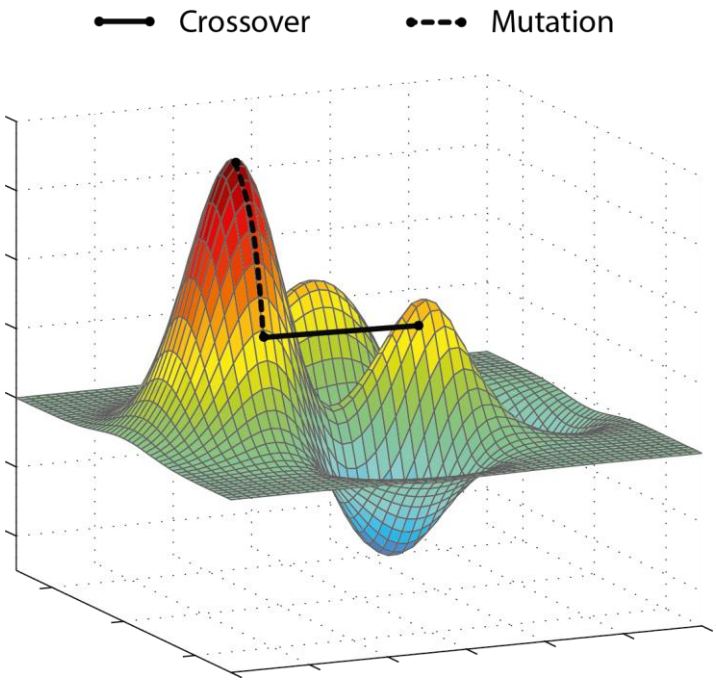


Abbildung 2.4. Darstellung der Mutations- und Crossover-Operation als Bewegung im Lösungsraum. Quelle: Eigene Darstellung.

Die von einem evolutionären Algorithmus gefundene Lösung ist dabei selten exakt, sondern eine Annäherung (engl. approximation) an ein Optimum.

2.4.1.1 Aufbau

Ein evolutionärer Algorithmus setzt sich aus drei Hauptkomponenten zusammen:

- **Initiator** – Das Startprogramm, welches den Evaluator und die Population erzeugt.
- **Individuum** – Ein Individuum repräsentiert zu einem Zeitpunkt exakt eine Lösung. Die Gesamtheit der Individuen bildet die Population.
- **Evaluator** – Der Evaluator bewertet die Modelle, wählt die Überlebenden für die nächste Generation aus und erzeugt aus diesen neue Individuen.

Abbildung 2.5 stellt die Hauptkomponenten und deren Zusammenspiel grafisch dar. In den folgenden Abschnitten werden die Hauptkomponenten sowie die Konzepte der evolutionären Algorithmen im Detail vorgestellt.

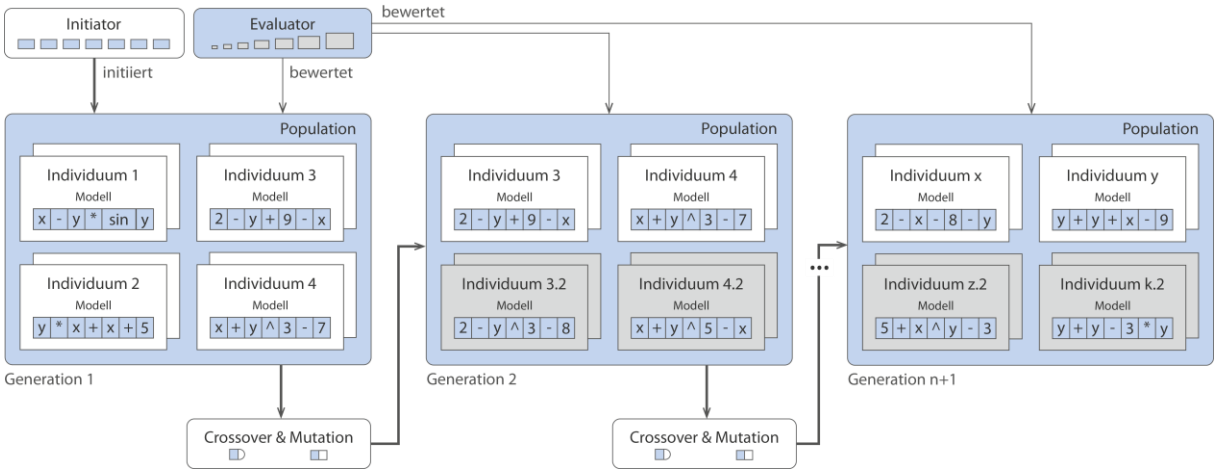


Abbildung 2.5. Hauptkomponenten eines evolutionären Algorithmus und deren Zusammenspiel. Quelle: Eigene Darstellung.

2.4.1.1.1 Individuum

Individuen sind die Kernelemente eines evolutionären Algorithmus. Überführt auf das Prinzip der natürlichen Evolution nach Darwin entspricht ein Individuum einem Lebewesen. Ein Individuum repräsentiert zu einem Zeitpunkt exakt eine Lösung. Individuen werden über die Laufzeit des evolutionären Algorithmus erzeugt, verändert oder entfernt. Zu Beginn des evolutionären Algorithmus (zur Generation 0) werden alle Individuen mit einem zufälligen Modell initialisiert.

2.4.1.1.2 Population

Die Gesamtmenge der Individuen bezeichnet man als die Population. In den gängigen Implementierungen von evolutionären Algorithmen, und auch den Implementierungen in dieser Arbeit, ist die Anzahl an Individuen in der Population fix. Jedes ausgeschiedene Individuum wird somit unverzüglich durch ein neues ersetzt.

2.4.1.1.3 Generation

Eine Generation stellt die Gesamtheit der Zustände der einzelnen Individuen einer Population zu einem bestimmten Zeitpunkt dar. Die Generation 0 besteht genau dann, wenn alle Individuen erstmalig initialisiert sind. Alle weiteren Generationen x+1 entstehen durch das Verändern und Ersetzen von Individuen der vorherigen Generation x mittels Reduktion, Elternauswahl, Crossover und Mutation.

2.4.1.1.4 Initiator

Der Initiator ist das Startprogramm, welches sowohl den Evaluator als auch die Individuen erzeugt. Anschließend besteht die Aufgabe des Initiators in der Behandlung von Ausnahmefehlern, auf den Normalablauf des evolutionären Algorithmus hat der Initiator keinen weiteren Einfluss.

2.4.1.1.5 Evaluator und Fitness-Funktion

Die Entscheidung, ob ein Individuum in die Folgegeneration übernommen wird, hängt vom Fitness-Wert des Individuums ab. Ein Individuum hat einen guten Fitness-Wert, wenn es eine (relativ zu den anderen Individuen) gute Lösung für das beschriebene Problem gefunden hat. Die Beantwortung der Frage, wann es sich um eine „gute Lösung“ handelt, ist eine der schwierigsten Aufgaben bei der Implementierung eines evolutionären Algorithmus.

Dies sei an einem Beispiel aus dem täglichen Leben verdeutlicht. Angenommen, das Ziel ist es, an einen Punkt x, beispielsweise eine bestimmte Adresse, zu gelangen. Dieser Punkt x soll möglichst mit dem Auto erreicht werden, zur Routenberechnung wird ein Navigationssystem eingesetzt. Eine intuitive Fitness-Funktion bestände aus den folgenden Bedingungen:

1. Eine Route ist umso besser, je näher sie mit dem Auto an den Punkt x heranführt.

2. Eine Route ist umso besser, je weniger Wegstrecke sie enthält.

Aus diesen beiden Bedingungen wird bereits das Gewichtungsproblem sichtbar. Welcher Abstand zu Punkt x ist akzeptabel? Welche zusätzliche Wegstrecke wird in Kauf genommen, um eine Annäherung um n Meter an den Punkt x zu erreichen?

Zudem ergibt sich eine zusätzliche, weniger intuitive Fragestellung: Ist eine Route mit kurzer Wegstrecke, die nah an den Punkt x heran führt, wirklich gut? Nicht immer – eine Route, die wenige Meter an den Punkt x heranführt, ist nur dann gut, wenn zwischen ihrem Endpunkt und dem Punkt x keine unüberwindbaren Hindernisse (bewehrte Mauern, Flüsse ohne Brücke, etc.) existieren.

An dem Beispiel wird deutlich, dass die Definition einer Fitness-Funktion, also einer Funktion, die einem Individuum einen eindeutigen Fitness-Wert zuweist, eine nicht-triviale Aufgabe darstellt. Diese wird zusätzlich erschwert durch Probleme, bei denen eine gute Teillösung nicht zwingend ein Teil einer optimalen Lösung darstellt.

Die Aufgabe des Evaluators ist die Bewertung (die Berechnung der Fitness-Funktion) der einzelnen Individuen. Der Evaluator führt diese Bewertung genau einmal für jedes Individuum jeder Generation durch. Basierend auf der Bewertung durch den Evaluator erfolgt die Entscheidung, welches Individuum in die nächste Generation übernommen wird und damit als „Elternteil“ für die ersetzten Individuen fungiert.

2.4.1.2 Ablauf

Der Ablauf eines evolutionären Algorithmus ähnelt dem Ablauf der natürlichen Evolution, jedoch stark beschleunigt. Er besteht aus drei Grundkonzepten:

- **Vererbung** – Die vom Evaluator aussortierten Individuen werden ersetzt, indem sie die Modelle überlebender Individuen, der sogenannten Eltern, übernehmen und verändern. Dies geschieht im Prozess der Vererbung.

- **Crossover** – Der Crossover-Operator führt eine Vermischung der Elternmodelle durch. Hierdurch entsteht das vorab beschriebene Springen im Suchraum. Der Crossover-Operator wird nicht immer, sondern nur mit einer vorkonfigurierten Wahrscheinlichkeit durchgeführt (siehe Abschnitt 2.4.4.1.4).

- **Mutation** – Der Mutations-Operator führt zufällige Veränderungen an den vererbten und gegebenenfalls durch Crossover vermischten Modellen durch. Dies hat einen Schritt im Suchraum zur Folge, wie im einleitenden Absatz beschrieben. Auch der Mutations-Operator wird nicht immer, sondern nur mit einer vorkonfigurierten Wahrscheinlichkeit durchgeführt (siehe Abschnitt 2.4.4.1.3).

In den folgenden Abschnitten werden diese drei Grundkonzepte im Detail ausgeführt.

2.4.1.2.1 Vererbung

Die Variation der von den Individuen getragenen Modelle erfolgt, der natürlichen Evolution folgend, von Generation zu Generation. Hierbei kommt das Konzept der Vererbung zum Einsatz. Die Vererbung bezeichnet die Übergabe der Modelle der „Eltern“-Individuen an die neu zu erstellenden Individuen der Folgegeneration. Zur Erzeugung von Variationen in den Modellen kommen zwei Operationen zum Einsatz: Mutation und Crossover. Die Auswahl der Eltern und die beiden Operationen werden im Folgenden definiert.

2.4.1.2.1.1 Eltern-Auswahl

Zur Erzeugung eines neuen Individuums werden gängig zwei Eltern-Individuen verwendet, wobei theoretisch auch drei oder mehr Eltern-Individuen möglich sind. Die Eltern-Individuen

gehören dem Teil einer Population an, der in die nächste Generation übernommen wird. Es ist jedoch nicht zwingend, dass alle übernommenen Individuen als Eltern-Individuen fungieren – gleichzeitig kann ein Individuum auch für mehrere Kind-Individuen als Eltern-Individuum zum Einsatz kommen.

Für die Auswahl der Eltern-Individuen gibt es mehrere Ansätze. All den Ansätzen liegt zugrunde, dass nur ein gewisser Anteil der Population (definiert durch die sogenannte *Parental Population Quota*) als Eltern-Individuen fungiert. Die Parental Population Quota bestimmt, wie viel Prozent der Population, in absteigendem Fitness-Wert, als Eltern-Individuen in Frage kommen. Die gängigsten Ansätze zur Eltern-Auswahl sind:

- **Gleichverteilung** – Aus den möglichen Eltern-Individuen werden zufällig zwei (unterschiedliche) Individuen ausgewählt. Diese beiden Individuen bilden die gleichberechtigten Eltern des neuen Individuums.

- **Wahrscheinlichkeit abhängig von dem Fitness-Wert** – Die Wahrscheinlichkeit, dass ein Individuum als Eltern-Individuum agiert, hängt bei diesem Ansatz von seinem Fitness-Wert ab. Individuen mit höherem Fitness-Wert kommen hierdurch mit einer höheren Wahrscheinlichkeit als Eltern-Individuum zum Einsatz als Individuen mit einem niedrigen Fitness-Wert. Dieser Ansatz erlaubt theoretisch die Verwendung von 100 Prozent der Population als mögliche Eltern-Individuen.

Die Auswahl des optimalen Ansatzes für ein Problem hängt von mehreren Faktoren ab. Zum einen beeinflusst der Ansatz das Konvergenzverhalten des evolutionären Algorithmus. Erfolgt die Eltern-Auswahl proportional zum Fitness-Wert, so führt der evolutionäre Algorithmus eine engere, zielgerichtete Suche durch. Beeinflusst dadurch, dass ein Individuum mit einem bereits guten Modell überproportional oft als Eltern-Individuum verwendet wird, erfolgt eine Optimierung dieses Modells mit deutlich höherer Wahrscheinlichkeit als die Generierung eines neuen Modells. Der Ansatz der proportionalen Wahrscheinlichkeitsverteilung führt damit zu einem beschleunigten Konvergenzverhalten, verbunden mit dem Risiko, dass der Algorithmus zu einem lokalen Optimum konvergiert. Exakt entgegengesetzt verhält es sich bei der Gleichverteilung. Hier erfolgt die Suche breiter, die Chance, ein lokales Optimum zugunsten des globalen (oder zumindest eines besseren lokalen) Optimums zu verlassen ist, abhängig von der gewählten Parental Population Quota, etwas oder deutlich besser, jedoch verbunden mit einem (zum Teil deutlich) schlechteren Konvergenzverhaltens.

Die Auswahl des Ansatzes zur Eltern-Auswahl ist damit stark abhängig vom zu lösenden Problem. Besteht der Lösungsraum aus zahlreichen lokalen Optima, so ist oft die Gleichverteilung vorzuziehen. Werden wenige lokale Optima vermutet, oder wird eine schnelle Konvergenz benötigt, so ist vermutlich der proportionale Ansatz das Mittel der Wahl. Hinzu kommt der erhöhte Aufwand für die Berechnung der Wahrscheinlichkeiten und die kompliziertere Auswahl der Eltern-Individuen beim proportionalen Ansatz – ein Faktor, der ebenfalls nicht vernachlässigt werden darf.

2.4.1.2.2 Crossover

Der Crossover-Operator ist der in der Natur am häufigsten vorkommende Operator. Auch in evolutionären Algorithmen findet er, wie in Teilkapitel 4.4 später dargelegt werden wird, häufig Anwendung. Allgemein gesagt verursacht der Crossover-Operator ein „Springen“ durch den Suchraum, und eignet sich damit insbesondere zum Auffinden von lokalen und globalen Optima. Der Crossover-Operator ist die Primäroperation evolutionärer Algorithmen (Eiben/Smith 2003b; Goldberg 1989; Koza 1992b; Lämmel/Cleve 2008). Da ein evolutionärer Algorithmus zahlreiche Formen der Modelldarstellung annehmen kann wird der Crossover-Operator in diesem Abschnitt am Beispiel der Genom-Darstellung der genetischen Algorithmen (siehe hierzu Teilkapitel 2.4.3) erläutert.

Der Crossover-Operator dient der Vermischung der Genome der Eltern-Individuen zu den Genomen der Kind-Individuen (De Jong 1988). (Goldberg 1989) und (Koza 1992b) folgend gibt es mehrere Arten des Crossover-Operators:

- **Einfache Schnittstelle** – Die beiden Genome der Eltern-Individuen werden an je einer Stelle geschnitten, die resultierenden Kind-Genome entstehen durch das Zusammenfügen von je einem Teil jedes Eltern-Genoms. Je nach Implementierung werden auf diese Weise gleich zwei Kind-Genome erzeugt.

- **Mischen** – Die Genome der Eltern-Individuen werden komplett vermischt und dann in der Mitte geteilt, um die Kind-Genome zu erzeugen. Die Crossover-Variante *Mischen* ist sehr intensiv und verursacht große Sprünge im Suchraum.

- **Partially Matched Crossover** – Das Partially Matched Crossover (PMX) ist eine komplexe Crossover-Variante, bei der ein Teil der Eltern-Genome ausgetauscht und der Rest entsprechend angepasst wird. PMX vermischt eine Konstanz in den Genoms mit einer Verschiebung in einer oder mehrere Dimensionen, abhängig vom Genom des anderen Elternteils.

- **Order Crossover** – Order Crossover (OX) ähnelt zum Teil dem PMX, ist jedoch noch komplexer im Ablauf. Beim OX werden ebenfalls Teile der Eltern-Genome ausgetauscht, jedoch erfolgt zudem eine Verschiebung innerhalb der Kind-Genome. OX erzeugt eine starke Veränderung der Genome und findet hauptsächlich bei kombinatorischen Problemen Anwendung.

Neben den vorgestellten Crossover-Varianten sind in der Literatur weitere Verfahren zu finden, beispielsweise das Cycle Crossover (Goldberg 1989). Diese sind jedoch durchweg für kombinatorische Probleme konzipiert und damit für diese Arbeit nicht von Relevanz. Bereits PMX und OX zielen auf kombinatorische Probleme und dienen als Repräsentanten dieser Klasse von Crossover-Verfahren.

In den folgenden Abschnitten werden die vorgestellten Crossover-Verfahren im Detail vorgestellt.

2.4.1.2.2.1 Crossover-Verfahren *Einfache Schnittstelle*

Das Crossover-Verfahren *Einfache Schnittstelle* bildet das einfachste Crossover-Verfahren. Wie Abbildung 2.6 zeigt entstehen durch das Crossover-Verfahren *Einfache Schnittstelle* zwei Kind-Genome, indem die Eltern-Genome an einer Stelle geschnitten werden und die vier Teile paarweise zu neuen Genomen zusammengesetzt werden.

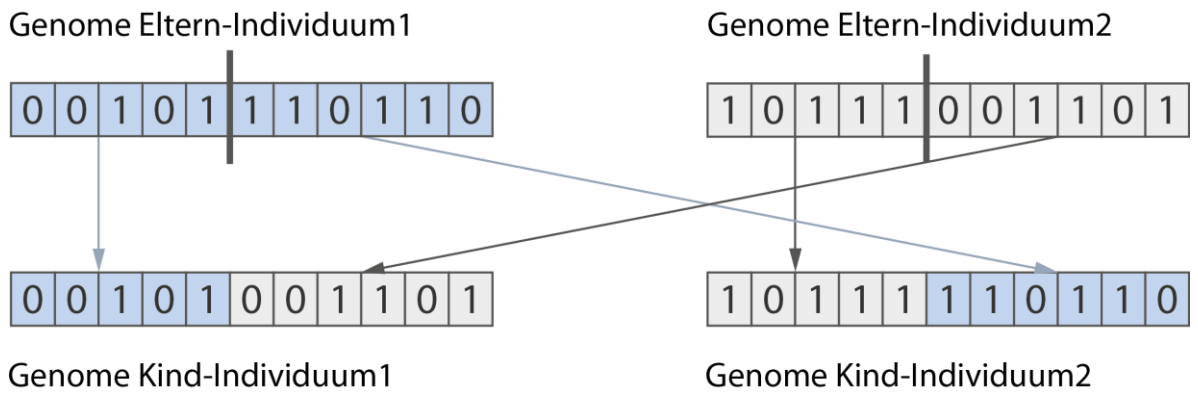


Abbildung 2.6. Crossover-Verfahren *Einfache Schnittstelle.* Quelle: Eigene Darstellung.

Das Crossover-Verfahren *Einfache Schnittstelle* existiert in mehreren Varianten, die sich in der Auswahl der Schnittstelle sowie der Reihenfolge des Zusammensetzens unterscheiden.

Die einfachste Variante ist ein Schnitt in der Mitte beider Eltern-Genome sowie einem Zusammensetzen, bei dem je der erste Teil des Eltern-Genoms zum ersten Teil des Kind-

Genoms wird. Hierdurch kommen immer beide Eltern-Genome gleichberechtigt zum Einsatz. Diese Variante lässt sich modifizieren, indem die Schnittstelle entweder rein zufällig gewählt wird (und damit die Gewichtung der Eltern-Genome), oder die Schnittstelle über eine Verteilungsfunktion oder den Fitness-Wert der Eltern-Genome bestimmt wird.

Eine weitere Variationsmöglichkeit bestimmt die Reihenfolge des Zusammensetzens. Diese kann variiert werden, indem der erste Teil eines Eltern-Genoms zufällig zum ersten oder zweiten Teil des Kind-Genoms wird. Eine solche Umsortierung hat insbesondere in der Modellbildung durch einen evolutionären Algorithmus (aufgrund der Rechtsklammerung, siehe Abschnitt 2.4.3.1) eine immense Auswirkung auf das Modell und den Fehlerwert.

Das Crossover-Verfahren *Einfache Schnittstelle* zeichnet sich insbesondere durch eine einfache und in der Ausführung ressourcenschonende Implementierung aus. Weiterhin erlaubt es eine hohe Kontrolle über die durchzuführende Veränderung der Genome. Insbesondere bei sehr langen Genomen und zufälliger Schnittstellenwahl jedoch variiert der Effekt der Crossover-Variante stark.

2.4.1.2.2.2 Crossover-Verfahren *Mischen*

Wird je Anwendung des Crossover-Operators eine möglichst große Veränderung der Kind-Genome angestrebt, so kommt das Crossover-Verfahren *Mischen* zum Einsatz. Beim Mischen werden die einzelnen Elemente der Eltern-Genome der Reihe nach vermischt. Hierdurch entsteht eine Sequenz von Genome-Elementen, bei der alle ungeraden Elemente vom Genom des Eltern-Individuums 1 und alle geraden Elemente vom Genom des Eltern-Individuums 2 stammen. Diese Sequenz wird in der Mitte getrennt, die erste Hälfte bildet das erste Kind-Genom, die zweite Hälfte das zweite Kind-Genom. Abbildung 2.7 illustriert das Vorgehen.

Durch das Mischen entsteht eine sehr starke Veränderung der Eltern-Genome bei der Vererbung hin zu den Kind-Genomen. Das Crossover-Verfahren *Mischen* ermöglicht hierdurch ein intensives Durchsuchen des Suchraums, schränkt das Konvergenzverhalten zu einem Optimum jedoch stark ein. Das Mischen kommt als Crossover-Variante deshalb insbesondere bei Problemen zum Einsatz, bei denen das Aufspüren eines oder weniger Optima im Suchraum einen hohen Suchaufwand erfordert. Bei Problemen mit vielen lokalen Optima ist das Mischen nicht effizient.

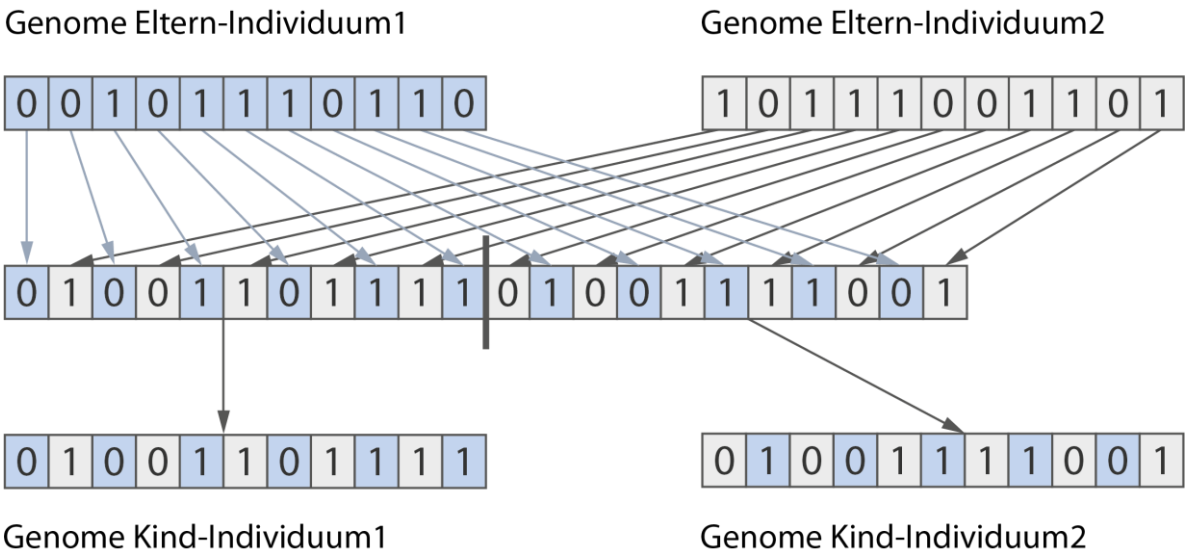


Abbildung 2.7. Crossover-Verfahren *Mischen*. Quelle: Eigene Darstellung.

2.4.1.2.2.3 Crossover-Verfahren *Partially Matched Crossover*

Das Crossover-Verfahren *Partially Matched Crossover (PMX)* gehört zu den komplexeren Crossover-Verfahren. Im ersten Schritt wird bei PMX, ähnlich dem Verfahren *Einfachen Schnittstelle*, eine Schnittstelle gewählt. Die Schnittstelle bei PMX sollte deutlich linkslastig gewählt werden, gängig ist die Wahl einer Schnittstelle nach den ersten drei oder vier Elementen. Nun folgen zwei Aktionen. Im zweiten Schritt erfolgt der Austausch der linken Hälften. Der dritte Schritt besteht aus der Ersetzung aller Elemente der rechten Hälften, die im neu dazugewonnenen linken Teil existieren, durch ihre Gegenelemente des vormaligen linken Teils. Abbildung 2.8 verdeutlicht das Vorgehen bei PMX.

PMX führt zu einer deutlichen Veränderung der Kind-Genome, die jedoch durch die Wahl der Schnittstelle dosiert werden kann. Es ist ersichtlich (und auch in der Abbildung bewusst gewählt), dass PMX ausschließlich auf komplexeren, nicht-binären Modellen durchgeführt werden kann.

Die PMX-Variante verursacht zur Laufzeit einen deutlich höheren Berechnungsaufwand als beispielsweise die Crossover-Variante *Einfache Schnittstelle*, da das gesamte Genom gelesen werden muss. Hierfür bietet PMX eine zielgerichtetere Suche mit größeren Sprungmöglichkeiten. PMX ist primär für den Einsatz bei kombinatorischen Problemen konzipiert.

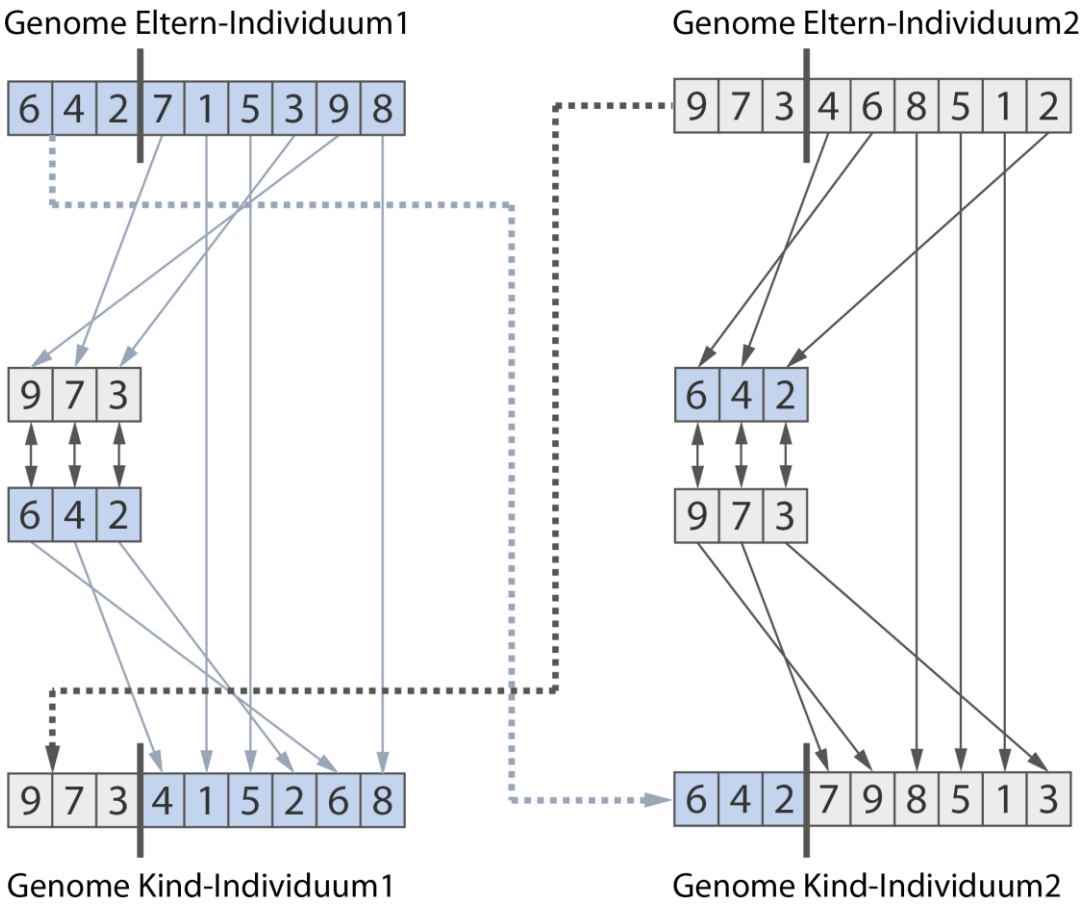


Abbildung 2.8. Crossover-Verfahren *Partially Matched Crossover*. Quelle: Eigene Darstellung.

2.4.1.2.2.4 Crossover-Verfahren *Order Crossover*

Order Crossover (OX) ist ein weiteres komplexes Crossover-Verfahren. OX orientiert sich an dem PMX-Verfahren, führt jedoch weitere, verschiebende Änderungen am Genom durch. Der erste Schritt bei OX ist die Auswahl zweier Schnittstellen, die zur Definition eines festen Blocks von Elementen, gängig drei bis fünf Elemente, führen. Diese Blöcke bilden, wie bei PMX, die Austauschelemente. Jedoch werden bei OX nicht, wie bei PMX üblich, die jeweiligen gegenüberliegenden Block-Elemente ersetzt. Vielmehr werden alle Vorkommnisse der Elemente des gegenüberliegenden Blocks aus dem Genom entfernt und durch Leerfelder ersetzt. Diese Leerfelder werden in der Mitte zusammengezogen und bilden dort einen Block. Dieser Leerblock wird durch den Block des jeweiligen anderen Eltern-Genoms ersetzt. Abbildung 2.9 illustriert das OX-Verfahren.

Es ist ersichtlich, dass sich OX ausschließlich für kombinatorische Permutationsprobleme eignet. Nur bei diesen Problemen ist sichergestellt, dass im ausgewählten Block jedes Element maximal einmal vorkommt. Und nur bei kombinatorischen Problemen ergibt sich durch das Ersetzen durch Leerelemente ein Leerblock, der exakt die Größe des Ersetzungsblocks besitzt.

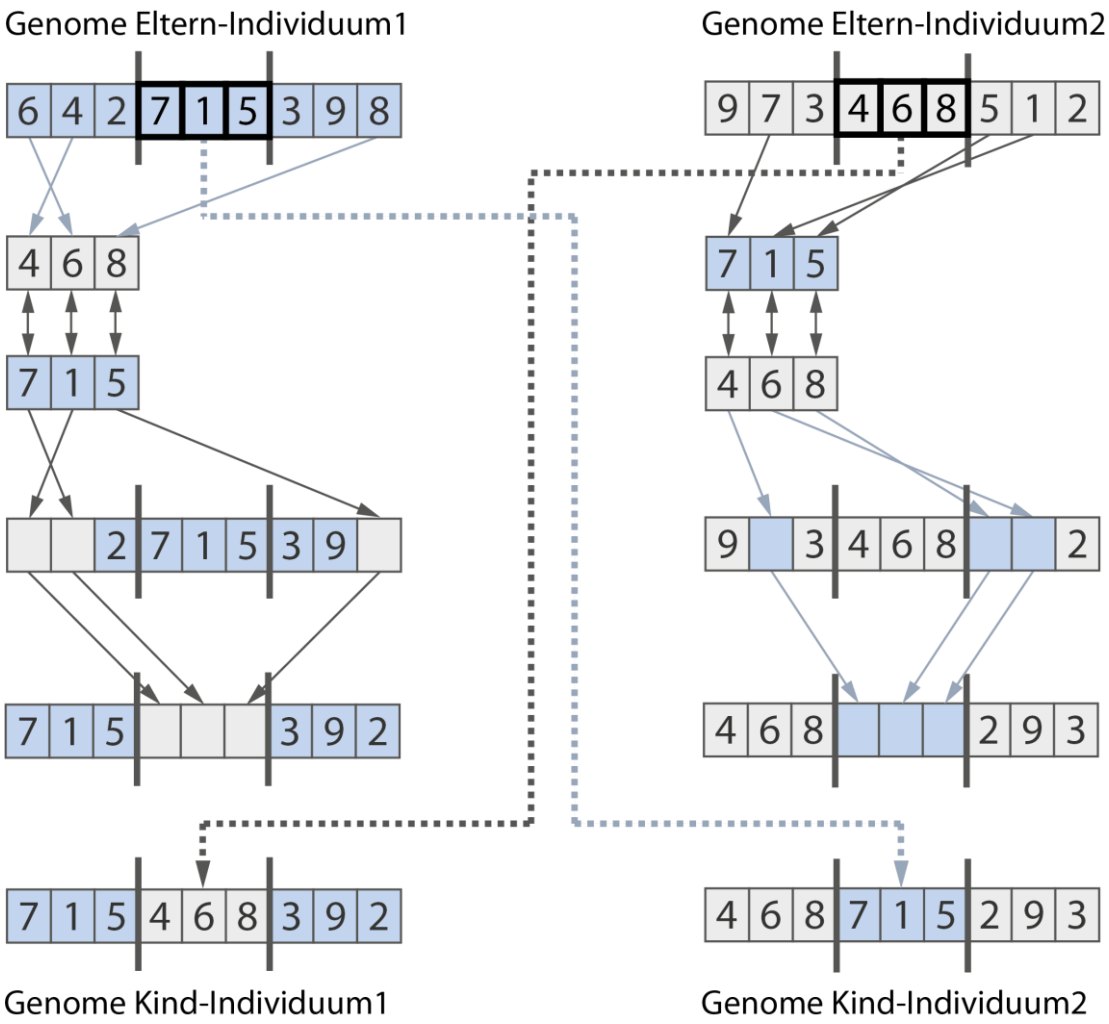


Abbildung 2.9. Crossover-Verfahren *Order Crossover*. Quelle: Eigene Darstellung.

2.4.1.2.3 Mutation

Der Mutations-Operator führt auf den Modellen der Population zufällige Veränderungen durch (Koza 1992b). Dies erfolgt durch das zufällige Verändern eines Genom-Elements. Mutation erfolgt nach der Erstellung eines neuen Kind-Individuums, im Allgemeinen nach der Durchführung des Crossover-Operators. Abbildung 2.10 stellt den Mutations-Operator grafisch dar.

Der Mutations-Operator dient der Konvergenz der Modelle der Individuen zu einem lokalen oder globalen Optimum (siehe Abbildung 2.4). In der Literatur wird der Mutations-Operator oft als Operator zweiter Ordnung angesehen (vgl. (Goldberg 1989; Koza 1992b)), da er in der Natur deutlich seltener auftritt als der Crossover-Operator. Die in Kapitel 4.4 durchgeführten Experimente belegen jedoch die enorme Wichtigkeit des Mutations-Operators für das Konvergenzverhalten des evolutionären Algorithmus für den in dieser Arbeit betrachteten Anwendungsfall.

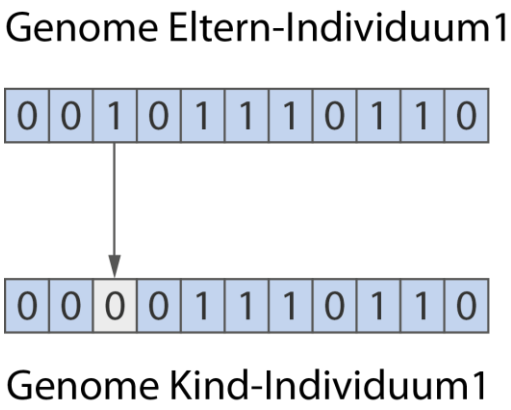


Abbildung 2.10. Grafische Darstellung des Mutations-Operators am Beispiel. Quelle: Eigene Darstellung.

2.4.1.2.4 Ende-Kriterium

Das Ende-Kriterium bestimmt, welche Bedingung erfüllt sein muss, damit der evolutionäre Algorithmus endet. (Koza 1992b) definiert zwei Ende-Kriterien:

- **Vordefinierte Anzahl an Generationen erreicht** – Dieses Ende-Kriterium kommt zum Einsatz, wenn kein problemspezifisches Erfolgsprädikat definiert werden kann, oder im Kombination mit dem im folgenden vorgestellten problemspezifischen Erfolgsprädikat, um die Laufzeit des evolutionären Algorithmus zu begrenzen. Dieses Generations-Prädikat definiert, wie viele Generationen vom evolutionären Algorithmus (maximal) durchlaufen werden. Es ist damit vollständig losgelöst von dem gestellten Problem.

- **Problemspezifisches Erfolgsprädikat erfüllt** – Gibt es eine einhundertprozentige Lösung für ein gegebenes Problem, so ist diese oft das problemspezifische Erfolgsprädikat. Bei kombinatorischen Problemen wie dem Traveling Salesman Problem (Bhatia 1994; Bryant 2000) ist dies oft der Fall. Bei Problemen, deren Lösungen nicht als solche erkannt werden, selbst wenn man sie sieht (beispielsweise Optimierungsprobleme), oder Probleme, für die es voraussichtlich keine akzeptablen exakten Ergebnisse gibt (beispielsweise der mathematischen Modellierung von verrauschten Daten), werden niedrigere Kriterien angesetzt. Das problemspezifische Erfolgsprädikat „Modellierungsfehler < 5%“ ist ein klassisches Ende-Kriterium bei der Modellierung von Performance-Messwerten (Tertilt et al. 2010).

Die Definition eines Ende-Kriteriums ist eine anspruchsvolle Aufgabe. Das Konvergenzverhalten eines evolutionären Algorithmus ist stark abhängig von der gegebenen Problemstellung sowie von der Konfiguration des Algorithmus (vgl. Kapitel 4.4). Wählt man das Ende-Kriterium zu schwach, so endet der Algorithmus mit einer suboptimalen Lösung.

Wird das Ende-Kriterium jedoch zu restriktiv gewählt, so verwendet der evolutionäre Algorithmus zu viele Ressourcen und endet gegebenenfalls nie, obwohl bereits brauchbare Ergebnisse vorliegen. Die Auswahl eines Ende-Kriteriums für den in dieser Arbeit untersuchten Anwendungsfall wird in Kapitel 4 ausführlich beschrieben.

2.4.2 Evolutionäre Algorithmen zur Modellierung

Ursprünglich fanden evolutionäre Algorithmen ihre Anwendung insbesondere bei der Lösung kombinatorischer Permutationsprobleme (siehe beispielsweise (Gwozdz/Szlachcic 2009), (Puljic/Manger 2005) und (Scheffermann et al. 2009)) sowie bei der Lösung von Optimierungsproblemen (siehe (Fonseca/Fleming 1995, 1998), (Coello et al. 2007) und (Sbalzarini et al. 2000)). Diese Problemstellungen eignen sich vorzüglich für die Bearbeitung durch einen evolutionären Algorithmus, da sie jeweils mindestens zwei der drei folgenden Eigenschaften erfüllen.

- **Beschränkte Menge an Elementwerten** – Die Anzahl der Werte, die ein Element im Modell eines Individuums annehmen kann, ist beschränkt. Die Menge der Werte, die ein Modell-Element annehmen kann, ist endlich.

- **Eine gute Lösung ist identifizierbar** – Für eine gegebene Lösung (ein Modell eines Individuums) ist entscheidbar, wie gut die Lösung ist. Insbesondere ist von Bedeutung, dass die Lösungen zueinander in Relation gesetzt werden können, dass also entschieden werden kann, ob eine Lösung besser, schlechter oder gleich gut ist wie eine andere Lösung.

- **Gute Teillösungen sind mit hoher Wahrscheinlichkeit auch Bestandteil der optimalen Lösung** – Findet ein Individuum eine gute Teillösung, so ist die Wahrscheinlichkeit hoch, dass diese Lösung ein Teil einer guten Gesamtlösung oder sogar der optimalen Lösung ist.

Erfüllt eine Problemstellung diese Eigenschaften, so ist sie von einem evolutionären Algorithmus gut zu lösen. Die Beschränkte Menge an Elementwerten verhindert, dass sich der Algorithmus in einer Dimension „verrennt“, das heißt in unsinnige Wertebereiche vorstößt. Die Identifizierbarkeit einer guten Lösung ist Grundvoraussetzung für die Auswahl überlebender Individuen für die Folgegeneration (vgl. Abschnitt 2.4.1.1.5). Und die letzte Eigenschaft ermöglicht das Konvergieren zu einer Lösung, sobald gute Teillösungen identifiziert wurden.

Kombinatorische Permutationsprobleme wie das bereits erwähnte Vehicle Routing Problem besitzen erkennbar alle drei Eigenschaften. Optimierungsprobleme besitzen theoretisch einen unendlichen Lösungsraum, da die übergebenen Parameter-Vektoren Elemente aus $\mathbb{N}$ oder $\mathbb{R}$ enthalten können, sie besitzen demnach nur die zweite und dritte Eigenschaft.

Betrachtet man nun mathematische Modellierungsprobleme, so ist die erste Eigenschaft im Allgemeinen, aus demselben Grund wie bei den Optimierungsproblemen, nicht erfüllt. Die zweite Eigenschaft besitzen Modellierungsprobleme hingegen schon, hierzu sei auf die Fehlerwertberechnung in Kapitel 4.5 verwiesen. Eigenschaft drei hingegen ist bei mathematischen Modellierungsproblemen teilweise erfüllt. Bildet ein Modell eine Menge von zu modellierenden Daten mit einem geringen Fehlerwert ab, so ist die Wahrscheinlichkeit groß, dass dieses Modell den Rahmen für die finale Lösung bildet. Im Gegensatz zu den vorab vorgestellten Problemklassen haben Lösungen in Form mathematischer Modelle jedoch die Eigenschaft, dass kleine Veränderungen grundlegende Änderungen im Modell zur Folge haben können. Man beachte die grundlegende Änderung des beispielhaften Modells $f(x) = \frac{x}{10},\ x > 10$, wenn der Divisions-Operator durch den Multiplikations-Operator ersetzt wird. (Zhong et al. 2004) sowie (Parpinelli/Lopes 2011) und andere Arbeiten in diesem Umfeld zeigen jedoch die erfolgreiche Anwendung evolutionärer Algorithmen zur Lösung mathematischer Modellierungsprobleme.

2.4.2.1 Multidimensionale Modellierung

Die betrachteten Modellierungsprobleme sind multidimensional. Auf die zu modellierenden Werte wirken also mindestens zwei Faktoren ein. Ob zwischen den Faktoren Korrelationen bestehen ist im Allgemeinen nicht bekannt. Formal gesprochen ist das Ziel der multidimensionalen Modellierung damit die Suche nach einer Funktion $f(\vec{x}) = y$, die für eine Menge an Eingabe-Faktoren (dargestellt in der Formel durch den Vektor $\vec{x}$) den zu modellierenden Wert y liefert, und dies für alle möglichen Faktor-Vektoren $\vec{x}$.

Um dieses Ziel zu erreichen modellieren die Individuen des evolutionären Algorithmus Funktionen $f'(\vec{x}) = y'$, mit dem Ziel $\forall \vec{x}: y = y'$.

2.4.2.2 Fehlergrenzwert

In der Praxis ist eine exakte Modellierung komplexer mehrdimensionaler Datensätze kaum zu erreichen. Bedingt ist dies zum einen durch Fehler in den Datensätzen, zum anderen durch weitere, unbekannte Faktoren, die $f(\vec{x})$ beeinflussen und nicht in den Datensätzen abgebildet sind (beispielsweise, weil sie bei der Messung nicht berücksichtigt wurden). Ein evolutionärer Algorithmus als Approximationsansatz wird deshalb in nahezu allen Fällen ein

$f'(\vec{x}) = y'$ modellieren, für das gilt: $\forall\vec{x}: f(\vec{x}) = f'(\vec{x}) + \varepsilon_{\vec{x}}$. Die Modellwerte weichen von den zugrundeliegenden Werten ab.

Betrachtet man die Einsatzgebiete der durch den evolutionären Algorithmus erzeugten Modelle, so ist diese Abweichung ε in den meisten Fällen nicht problematisch, solange sie unter einem Grenzwert, dem sogenannten *Fehlergrenzwert*, liegt.

Der Fehlergrenzwert gibt an, ab welcher Abweichung des Modells von den zugrundeliegenden Daten das Modell seinen Nutzen verliert. Damit ist der Fehlergrenzwert stark abhängig vom Anwendungsfall und muss für diesen bestimmt werden.

Im Fall der vorliegenden Arbeit erfolgt eine Modellerstellung auf Performance-Messdaten von Unternehmensanwendungen. Der Fehlergrenzwert, der in dieser Arbeit zum Einsatz kommt, entspricht dem erwarteten Messfehler bei der Erhebung der Performance-Messwerte. Liegt der Modellierungsfehler unter dem Messfehler, so ist er für die Exaktheit des Modells nicht relevant – im Schnitt wird der Modellfehler den Messfehler genauso oft verringern wie erhöhen. Welcher konkrete Messfehler als Fehlergrenzwert zugrunde liegt wird in den jeweiligen Abschnitten dieser Arbeit am konkreten Fall ausgeführt.

2.4.3 Genetische Algorithmen – Eine Spezialform

Genetische Algorithmen nach (Eiben/Smith 2003b) sind evolutionäre Algorithmen mit einer speziellen Repräsentation der durch die Individuen generierten Modelle. In einem genetischen Algorithmus erfolgt die Repräsentation des Modells analog zum natürlichen Genom in Form einer linearen Folge von Informationselementen, wobei die Länge der Folge im Vornhinein definiert ist. Die genetischen Algorithmen bilden damit eine Untergruppe der evolutionären Algorithmen, die sich insbesondere durch eine schnelle Evaluation des Fehlerwerts eines Individuums auszeichnet. Die Komplexität der in dieser Arbeit durchgeführten multidimensionalen Modellierung erfordert den Einsatz von evolutionären Algorithmen mit großen Populationen, so dass sich diese Eigenschaft genetischer Algorithmen stark vorteilhaft auf die Anzahl der Generationen pro Minute auswirkt.

2.4.3.1 Aufbau eines Genoms

Genetische Algorithmen unterscheiden sich von anderen evolutionären Algorithmen insbesondere durch die Repräsentation des (Lösungs-)Modells eines Individuums. Die Hauptaufgabe beim Einsatz eines genetischen Algorithmus zur Lösung eines Problems liegt in der Erstellung einer effizienten Kodierung der potentiellen Lösung eines Individuums in

einer Genom-Struktur, welche die Berechnung eines Fitness-Wertes für jedes Individuum ermöglicht (Koza 1992b).

In der vorliegenden Arbeit besteht das gesuchte Lösungsmodell aus einer mathematischen Formel, welche die der Modellierung zugrunde liegenden Messdaten möglichst genau abbildet. Zur Darstellung eines solchen Modells in einer linearen Genom-Struktur interpretieren wir alle Genom-Elemente mit ungeradem Index (beginnend bei 1) entweder als Variable oder als Fließkommazahl, alle Genom-Elemente mit geradem Index als mathematische Operation. Auf diese Weise entstehen Genome mit immer ungerader Länge.

Im Folgenden wird der Aufbau des Genoms anhand eines Beispiels erläutert. Abbildung 2.11 stellt graphisch das exemplarische Modell $a + \frac{x}{b^{y - \sin(c*z)}}$ dar, mit *a*, *b*, *c* als Fließkommazahlen und *x, y, z* als Variablen.

a	+	*x*	/	*b*	^	*y*	- *sin*	*c*	*	*z*

Abbildung 2.11. Genom-Darstellung eines exemplarischen Modells. Quelle: (Tertilt et al. 2012)

Das Genom wird von rechts nach links interpretiert, wobei eine entsprechende Klammerung vorausgesetzt wird. Das in Abbildung 2.11 dargestellte Genom wird demnach interpretiert als f(x, y, z) = (a + (x / (b ^ (y –sin (c * z))))).

Die Auswahl der dem genetischen Algorithmus zu Verfügung gestellten Operationen wirkt sich stark auf die generierten Modelle aus. In der in dieser Arbeit verwendeten Implementierung *Mendel* (benannt nach dem Forscher Georg Johann Mendel) sind die in Tabelle 1 beschriebenen Operatoren implementiert.

Mit diesem Operationssatz sind die grundlegenden mathematischen Operationen definiert, die zum Aufbau einer mathematischen Formel benötigt werden. Zudem erlauben die Sinus-Operatoren die Modellierung von Schwingungen. Beim Entwurf von *Mendel* wurde das Ziel verfolgt, mit möglichst wenigen Operatoren auszukommen, um die resultierenden Modelle leicht verständlich zu halten. Die Fähigkeit, komplexe Messdatensätze zu modellieren, ist mit diesem Satz von Operatoren gegeben, wie in (Tertilt/Krcmar 2012) gezeigt.

Tabelle 1. Operationen des genetischen Algorithmus.

Operator	**Beschreibung**
a + b	Fließkomma-Addition *a + b*
a - b	Fließkomma-Subtraktion *a - b*
*a * b*	Fließkomma-Multiplikation *a * b*
a / b	Fließkomma-Division *a / b*
a ^ b	Fließkomma-Potenz. Interpretiert als *abs(a) ^ b*, da ein negatives a zu einem komplexen Ergebnis führen kann, wenn *b* < 1 ist
a + sin(b)	Fließkomma-Addition von *a* und dem Sinus von *b*
a – sin(b)	Fließkomma-Subtraktion von *a* und dem Sinus von *b*
*a * sin(b)*	Fließkomma-Multiplikation von *a* und dem Sinus von *b*
a / sin(b)	Fließkomma-Division von *a* und dem Sinus von *b*
a ^ sin(b)	Fließkomma-Potenz von *a* und dem Sinus von *b*, ebenfalls interpretiert als *abs(a) ^ sin(b)*

Die erweiterten Sinus-Operatoren sind notwendig, da der Sinus-Operator an sich unär ist. Der vorab beschriebene Aufbau des Genoms benötigt allerdings ausschließlich binäre Operatoren. Durch die Erweiterung des Sinus-Operators um jeweils die elementaren mathematischen Operatoren zu mehreren erweiterten Sinus-Operatoren ergibt sich ein Set von ausschließlich binären Operatoren.

2.4.3.2 Spezielle Eigenschaften genetischer Algorithmen

Genetische Algorithmen bringen den Vorteil einer einfachen Modellstruktur mit sich. Die Genom-Darstellung als lineare Folge von Elementen ist einfach zu erzeugen und zu parsen, im Vergleich beispielsweise zu einem evolutionären Algorithmus mit Baum- oder Netzstruktur. Ebenso sind die Operationen *Mutation* und *Crossover* leicht zu implementieren. Der Vorteil eines genetischen Algorithmus liegt damit in seiner Einfachheit, Effizienz und Verständlichkeit.

Dem gegenüber stehen zwei Nachteile. Zum einen eigenen sich genetische Algorithmen nur zur Lösung von Problemstellungen, deren Lösungen sich in einer lineare Genom-Struktur effizient darstellen lassen. Weiterhin lassen sich in einem genetischen Algorithmus nur schwer Hierarchieebenen abbilden, wie dies beispielsweise in einer Baumstruktur der Fall ist. Hierarchieebenen erlauben es, die Mutations- und Crossover-Operatoren mit unterschiedlicher Wahrscheinlichkeit einzusetzen, beispielsweise intensiver in niedrigen Hierarchieebenen.

Bei der Verwendung genetischer Algorithmen zur Generierung von Komponentenmodellen fallen die beiden Nachteile nicht ins Gewicht. Die Darstellung der Modelle in Form einer Genom-Struktur wurde bereits im vorherigen Abschnitt vorgestellt, eine hierarchische Betrachtung ist durch die Problemstellung nicht gegeben und gefordert. Die Vorteile

genetischer Algorithmen wirken sich auf der komplexen Problemstellung jedoch deutlich positiv aus. Aus diesem Grund eignen sich die genetischen Algorithmen gut für diese Aufgabe.

2.4.4 Konfiguration des genetischen Algorithmus

Die Konfiguration des genetischen Algorithmus wirkt sich stark auf seine Performance und Effizienz beim Lösen eines gegebenen Problems aus (Zitzler/Thiele 1999). Die optimale Konfiguration ist dabei stark abhängig von der Struktur des zu lösenden Problems, so dass keine generellen Aussagen über die Güte einer Konfiguration gemacht werden können. Eine optimale Konfiguration ist damit für jede Problemklasse separat zu identifizieren.

2.4.4.1 Konfigurationsparameter

Bei der Konfiguration eines genetischen Algorithmus gibt es nach (Goldberg 1989) fünf primäre Konfigurationsparameter zu beachten: die Populationsgröße, die Genome-Länge, die Mutationswahrscheinlichkeit, die Crossoverwahrscheinlichkeit und der Prozentsatz der überlebenden und als Eltern fungierenden Individuen einer Population, folgend Parental Population Quota genannt. Im Folgenden sind diese Konfigurationsparameter definiert. Es ist zu beachten, dass die Konfigurationsparameter nicht getrennt voneinander betrachtet werden können, sondern zueinander in Beziehung stehen. So ist beispielsweise die Genome-Länge invers proportional zur Populationsgröße zu setzen, um akzeptable Durchlaufzeiten für eine Generation zu erhalten.

2.4.4.1.1 Populationsgröße

Die Populationsgröße des genetischen Algorithmus bestimmt, wie viele Individuen zur Generierung des Approximationsmodells eingesetzt werden. Eine große Population birgt den Vorteil vieler Evolutionspfade, welche wiederum zu einer hohen Verbesserungswahrscheinlichkeit innerhalb einer Generation führen. Gleichzeitig jedoch benötigt jedes Individuum CPU- und Speicherressourcen für die Verwaltung und Evaluierung des generierten Modells. Diese Ressourcen stehen nur beschränkt zu Verfügung, so dass eine große Population zu langen Zeitabständen zwischen den einzelnen Generationen führt. Dies wiederum erhöht die Konvergenzzeit zu einem Optimum.

2.4.4.1.2 Genome-Länge

Über die Genome-Länge wird bestimmt, aus wie vielen Elementen ein Genom, also die Modellkodierung eines Individuums, besteht (siehe Abschnitt 2.4.3.1). Die Genome-Länge beschränkt damit die Komplexität des mathematischen Modells, welches zur Approximation der Messdaten heran gezogen werden kann. Ein langes Genom ermöglicht dem Algorithmus die Generierung komplexer Modelle und erhöht damit die Wahrscheinlichkeit eines geringen

Fehlerwertes. Gleichzeitig beschränken jedoch drei Faktoren die praktisch anwendbare Genome-Länge:

- Die für die Evaluierung des Fehlerwertes eines Modells benötigten CPU- und Speicherressourcen sind abhängig von der Komplexität des Modells und damit von der Genome-Länge. Lange Genome erhöhen damit insbesondere die für die Evaluierung einer Generation benötigen CPU-Zyklen und reduzieren damit die Anzahl an Generationen innerhalb eines Zeitabschnitts. Dies wiederum wirkt sich negativ auf die Konvergenzzeit des Algorithmus aus.

- Das Anwendungsgebiet des Modells, beispielsweise das Simulations-Framework, in welchem das Modell verwendet werden soll, schränkt möglicherweise die Länge des Modells ein.

- Der Effekt von Mutationsoperationen nimmt durchschnittlich mit zunehmender Genome-Länge ab. Der Grund hierfür liegt in der Eigenschaft der Mutationsoperation, ein Element des Genoms zu verändern. Besteht das Genom aus vielen Elementen, so sinkt die Wahrscheinlichkeit, dass die Veränderung eines Elements zu einer großen Veränderung des Fehlerwertes führt.

2.4.4.1.3 Mutationswahrscheinlichkeit

Die Mutationswahrscheinlichkeit bestimmt, mit welcher Wahrscheinlichkeit ein Element des Genoms zufällig verändert wird. Mutation bewirkt die Konvergenz zu einem (möglicherweise lokalen) Optimum. Eine geringe Mutationswahrscheinlichkeit bewirkt eine schnelle Durchdringung der Population mit einem Modell geringen Fehlerwerts, erhöht jedoch die Konvergenzzeit. Eine hohe Mutationswahrscheinlichkeit wiederum reduziert die Konvergenzzeit, reduziert jedoch die Durchdringung der Population mit einem möglicherweise guten Modell.

2.4.4.1.4 Crossoverwahrscheinlichkeit

Die Crossover-Operation ermöglicht den Individuen ein „Springen“ durch den Suchraum. Ohne Crossover besteht nahezu keine Chance für das Verlassen eines lokalen Optimums, nachdem dieses einmal gefunden wurde. Eine niedrige Crossoverwahrscheinlichkeit fördert damit die Konvergenz zu (möglicherweisen lokalen) Optima, während eine hohe Crossoverwahrscheinlichkeit diese vermindert, jedoch die Chance auf das Auffinden eines besseren lokalen oder sogar des globalen Optimums erhöht. Eine hohe

Crossoverwahrscheinlichkeit erhöht damit die Konvergenzzeit, liefert im Schnitt aber bessere Approximationen.

2.4.4.1.5 Parental Population Quota

Der Konfigurationsparameter Parental Population Quota gibt an, wie viel Prozent einer Population einen Generationswechsel überleben und als Eltern-Individuen für die in der folgenden Generation neu erstellten Individuen fungieren. In nahezu allen Implementierungen bezeichnet die Parental Population Quota die Quote der besten Individuen. Der Konfigurationsparameter wirkt sich auf die Diversifizierung der Modelle einer Generation aus: wird er hoch gewählt, so haben auch Individuen mit geringerem Fitness-Wert eine Chance, in die folgende Generation über zu gehen. Dies ist bei einem geringen Wert nicht der Fall. Gleichzeitig verursacht ein geringerer Wert ein deutlich besseres Konvergenzverhalten, da auf den wenigen sehr guten Modellen eine breitere Suche erfolgt, da mehr Individuen auf diesen Modellen operieren.

2.4.5 Zusammenfassung

Die vorherigen Abschnitte beschreiben die Grundlagen evolutionärer Algorithmen sowie ihre Anwendbarkeit auf das spezielle Problem der mathematischen Modellierung von Datensätzen. Im Detail werden die grundlegenden Elemente eines evolutionären Algorithmus und seine Operationen dargelegt. Anschließend wird die in dieser Arbeit verwendete Spezialform der evolutionären Algorithmen, die genetischen Algorithmen, vorgestellt. Folgend werden die Konfigurationsparameter eines genetischen Algorithmus erläutert, die in der folgenden Arbeit, insbesondere in den Kapiteln 4.4 und 5.3, eine bedeutende Rolle spielen.

Das Teilkapitel erläutert, weshalb der Einsatz evolutionärer Algorithmen, insbesondere die Unterart der genetischer Algorithmen, gut für die Modellierung multidimensionaler, in ihrer Struktur und ihren Abhängigkeiten vorab unbekannter Datensätze geeignet sind. Es werden die Vor- und Nachteile des Einsatzes evolutionärer Algorithmen aufzeigt. Weiterhin wird dargelegt, weshalb eine detaillierte Beschäftigung mit der Konfiguration des Algorithmus notwendig ist.

2.5 Unternehmensanwendungen

Der Titel der vorliegenden Abhandlung gibt bereits preis, dass Unternehmensanwendungen den Anwendungskontext dieser Arbeit bilden. Unternehmensanwendungen haben zahlreiche Eigenschaften, die sie von anderen Anwendungssystemen wie beispielsweise High-Performance-Computing-Systemen (HPC) unterscheiden. Dieses Teilkapitel führt in diese

Eigenschaften ein und definiert damit die Grundlagen für das Verständnis insbesondere des Anwendungsfalls, der in Kapitel 5 beschrieben wird.

Unternehmensanwendungen (englisch: Enterprise Applications) sind Softwaresysteme, die einen oder mehrere Geschäftsprozesse eines Unternehmens unterstützen. Klassische Unternehmensanwendungen sind Customer Relationship Management (CRM)-Systeme, Abrechnungsanwendungen, Versandplanungssysteme und Schadensbearbeitungssysteme von Versicherungen (Krafzig et al. 2005b). Im Vergleich zu anderen Softwaresystemen wie Betriebssystemen, Desktopanwendungen, eingebetteten Systemen, wissenschaftlicher Software oder Videospielen weisen Unternehmensanwendungen einige Besonderheiten auf. Im folgenden Teilkapitel werden diese Besonderheiten, Fowler (2003b) folgend, ausgeführt. Anschließend erfolgt eine Betrachtung der für Unternehmensanwendungen relevanten Performance-Kennzahlen. Die dort getroffenen Definitionen dienen als Basis für alle in dieser Arbeit durchgeführten Performance-Analysen.

2.5.1 Eigenschaften von Unternehmensanwendungen

Aufgrund ihrer Struktur, aber auch aufgrund der Art und Weise ihres Einsatzes unterscheiden sich Unternehmensanwendungen grundlegend von anderen Softwaresystemen. Die besonderen Eigenschaften von Unternehmensanwendungen wirken sich stark auf deren Performance-Verhalten aus und müssen bei sämtlichen Performance-Betrachtungen berücksichtigt werden. Im Folgenden werden die grundlegenden Charakteristika nach (Fowler 2003b) dargelegt.

2.5.1.1 Persistente Datenhaltung

Unternehmensanwendungen arbeiten generell auf Datensätzen, die aus betrieblichen oder rechtlichen Gründen über einen Neustart des Systems hinaus erhalten bleiben müssen, oftmals sogar über Jahre hinweg. Dieser kontinuierlich wachsende Datenstamm ist regelmäßig strukturellen Veränderungen durch Anpassungen an rechtliche oder betriebliche Rahmenbedingungen ausgesetzt. Auch über Änderungen am Betriebssystem oder der Unternehmensanwendung selber hinweg müssen diese Daten erhalten bleiben. Sie müssen meist aus Altsystemen migriert werden wenn eine Unternehmensanwendung eingeführt wird und an das Nachfolgesystem übergeben, wenn eine Unternehmensanwendung abgelöst wird.

2.5.1.2 Große Datenmengen

Unternehmensanwendungen decken Geschäftsprozesse breitflächig ab und verwalten auch die hierfür benötigten Daten. Diese Datensätze umfassen Lagerbestände, Personaldaten, Auftragseingänge, Zeitpläne und vieles mehr und haben nicht selten ein Volumen von vielen

hundert Gigabyte oder gar Terabyte. Diese Datenbestände sind in modernen Unternehmensanwendungen zumeist in relationalen Datenbanken abgelegt, doch befinden sich auch zahlreiche ältere Systeme im Einsatz, die ihre Daten auf dem Dateisystem, zum Beispiel in indizierten Dateisystemen wie VSAM oder ISAM, speichern. Die Verwaltung großer und kontinuierlich wachsender Datenbestände ist eine der Hauptaufgaben heutiger Unternehmensanwendungen.

2.5.1.3 Konkurrierender Datenzugriff

Der Zugriff auf Unternehmensanwendungen erfolgt konkurrierend, das heißt, mehrere Nutzer können gleichzeitig auf dieselben Daten zugreifen. Innerhalb eines Unternehmens sind dies im Normalfall bis zu einigen Hundert oder Tausend, doch bei Anwendungen mit Internetzugriff kann die Anzahl der parallelen Zugriffe um Größenordnungen höher liegen. Die Verwaltung der konkurrierenden Zugriffe und die Sicherstellung der Datenkonsistenz insbesondere bei Schreibzugriffen ist Aufgabe der Unternehmensanwendung.

2.5.1.4 Zahlreiche Benutzerschnittstellen

Aufgrund der Anzahl und Komplexität der abgebildeten Geschäftsprozesse und der Vielzahl der Anwender mit unterschiedlichsten Kenntnisständen besitzen Unternehmensanwendungen oft zahlreiche Benutzerschnittstellen mit hunderten von Oberflächen. Über diese Oberflächen werden die Daten der Anwendung den Nutzern auf verschiedenste Weise dargestellt und ermöglichen, je nach Anwendungsfall, unterschiedliche Interaktionen. Zusätzlich zu den Anwendungsoberflächen greifen Batch-Prozesse auf die Operationen und Daten der Unternehmensanwendung zu.

2.5.1.5 Integration

Unternehmensanwendungen sind oft in ein Netzwerk von mehreren Unternehmensanwendungen eingebettet. Diese Anwendungen sind zumeist zu unterschiedlichen Zeitpunkten und in unterschiedlichen Technologien entwickelt und über verschiedenste Interaktionskanäle wie Shared Databases (Hohpe/Woolf 2003b), CORBA (Dustdar et al. 2003) oder einen Messaging Bus (vgl. Abschnitt 2.6.2.4) integriert. Insbesondere im Zuge von Unternehmenszusammenführungen finden oft komplexe Systemintegrationen statt.

2.5.1.6 Konzeptuelle Unstimmigkeiten

Durch die Verteilung der Anwender über verschiedene Unternehmensbereiche und die Integration verschiedener Softwaresysteme kann es zu unterschiedlichen semantischen Interpretationen gleicher oder ähnlicher Datensätze kommen. Diese konzeptuellen Unstimmigkeiten entstehen, wenn ein Term in einer Anwendergruppe anders definiert wird

als in einer anderen. So kann, nach (Fowler 2003b), ein Kunde in einer Anwendergruppe als natürliche oder juristische Person definiert sein, mit der aktuell eine Geschäftsbeziehung besteht, in einer anderen Anwendergruppe gehören jedoch auch diejenigen Personen zu den Kunden, mit denen in der Vergangenheit Geschäftsbeziehungen bestanden. In großen Unternehmensanwendungen ist von einer Vielzahl dieser konzeptuellen Unstimmigkeiten auszugehen, insbesondere da sich die semantischen Interpretationen von Datenobjekten, verursacht durch geänderte Geschäftsprozesse und Rahmenbedingungen, in einem kontinuierlichen Wandel befinden.

2.5.1.7 Komplexe Geschäfts-„Unlogik“

Unternehmensanwendungen unterstützen Geschäftsprozesse. Dennoch ist insbesondere in etablierten Unternehmen mit einer Vielzahl an Ausnahmen von diesen fest definierten Geschäftsprozessen zu rechnen. Zum einen sind diese auf historische Änderungen zurück zu führen, zum anderen auch auf Verhandlungsergebnisse oder sonstige geschäftliche Zwänge, die insbesondere bei Kooperationen mit großen Unternehmen mit Marktmacht auftreten. Anders als beispielsweise bei wissenschaftlichen Anwendungen ist bei Unternehmensanwendungen die Umsetzung von tausenden dieser Ausnahmeregelungen der Normalfall.

2.5.2 Performance-Kennzahlen

Die Performance eines Softwaresystems manifestiert sich in unterschiedlichen Performance-Kennzahlen. Die Bedeutung einer Kennzahl variiert je nach Systemtyp und Anwendungsfall. So ist für ein Bremssteuerungssystem eines Automobils die Antwortzeit von entscheidender Bedeutung, während für ein Batchverarbeitungssystem im Allgemeinen der Durchsatz die entscheidende Kennzahl ist. Im Folgenden werden die für Unternehmensanwendungen relevanten Performance-Kennzahlen und ihre Bedeutung im Kontext dieser Arbeit definiert. Der in dieser Arbeit verwendete Anwendungsfall (definiert in Kapitel 5) basiert auf einer Service-orientierten Architektur, welche wiederum von einer Messaging-basierten Kommunikation geprägt ist (vgl. Teilkapitel 2.6), so dass die folgenden Definitionen die Eigenschaften dieses Kommunikationsstils berücksichtigen. Die Auflistung folgt der Empfehlung Fowlers (2003b) und dient als Grundlage der in den folgenden Kapiteln durchgeführten Performance-Betrachtungen.

2.5.2.1 Antwortzeit

Die Antwortzeit einer Operation ist die verstrichene Zeit zwischen dem Absenden des ersten Bits einer Anfrage bis zum Empfang des letzten Bits der Antwort. Dieser Definition folgend gehören zu einer Antwortzeit die Laufzeit der Anfrage (beispielsweise über ein Netzwerk) zum Server, gegebenenfalls das Routing und Demarshalling beim Einsatz von Messaging

(Krafzig et al. 2005d), die Verarbeitungsdauer der Anfrage auf dem Server, die Zeit zum Generieren der Antwort sowie zum Versenden der Antwort an den Client (und gegebenenfalls auch hier wieder das Routing und Marshalling der Message). Nicht zur Antwortzeit gehören hingegen die Zeit zum Generieren der Anfrage und des Marshallings der Message auf dem Client sowie die Zeit zum Verarbeiten der Antwort.

2.5.2.2 Reaktionszeit

Als Reaktionszeit einer Operation bezeichnet man die verstrichene Zeit zwischen dem Absenden des ersten Bits einer Anfrage bis zum Empfang der ersten Rückmeldung vom Server. Dies ist, im Gegensatz zur Antwortzeit, zumeist nicht das Ergebnis der Operation, sondern eine Empfangsbestätigung für die Anfrage, ein sogenanntes Acknowledge (ACK). Das Acknowledge dient im Allgemeinen dazu, den Nutzer über das erfolgreiche Versenden der Anfrage zu informieren. Abbildung 2.12 zeigt schematisch den Unterschied zwischen der Antwortzeit und der Reaktionszeit.

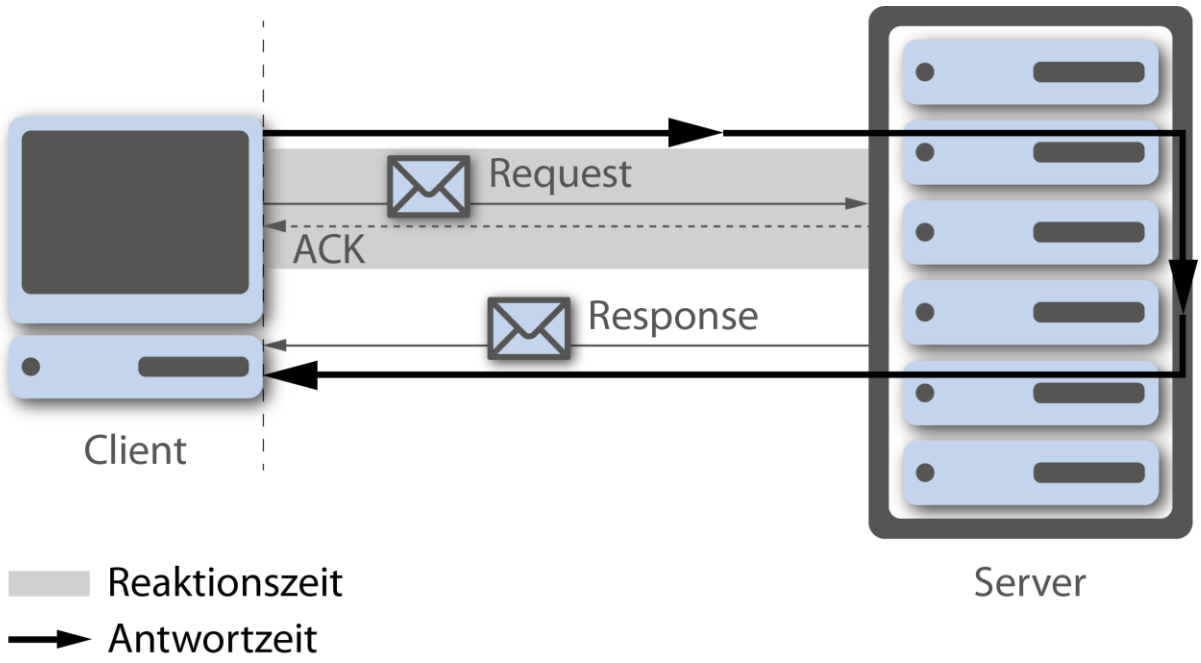


Abbildung 2.12. Schematische Darstellung der Reaktions- und Antwortzeit. Quelle: Eigene Darstellung.

2.5.2.3 Latenz

Die Latenz ist die minimale Antwortzeit eines Servers zu einem bestimmten Client. Sie wird erreicht, wenn der Server keinerlei Arbeit zu verrichten hat. Die Latenz bestimmt sich damit hauptsächlich durch die Netzlaufzeiten von Anfrage und Antwort, darin enthalten sind Routing, Authentifizierung und Verschlüsselung. Die Latenz hat großen Einfluss auf die Systemarchitektur – je höher die vermutete Latenz, desto weniger Service-Aufrufe sollten

notwendig sein um einen Prozess durchzuführen. Die Granularität der Serviceoperationen hängt damit maßgeblich von der erwarteten Latenz ab.

2.5.2.4 Durchsatz

Als Durchsatz bezeichnet man die Anzahl an Arbeitsschritten, die innerhalb eines Zeitschritts durchgeführt werden können. Beim Kopieren von Dateien ist eine typische Einheit für den Durchsatz *Bytes pro Sekunde*, bei Unternehmensanwendungen findet oft das Maß *Transaktionen pro Sekunde (tps)* Anwendung. Da Transaktionen unterschiedliche Laufzeiten haben können ist hier jedoch eine Auswahl vergleichbarer Transaktionen zu treffen, um Durchsatz in tps über Anwendungsgrenzen hinweg vergleichen zu können.

2.5.2.5 Last

Die Last ist eine Aussage über die Belastung einer Softwareanwendung. Ein gängiges Maß für die Last auf einer Anwendung ist die Anzahl der parallel auf die Anwendung zugreifenden Nutzer. Die Last ist keine Performance-Kennzahl an sich, sondern dient im Allgemeinen dazu, andere Performance-Kennzahlen in einen Kontext zu stellen. So ist beispielsweise die Antwortzeit einer Operation unter definierten Lastpunkten eine übliche Aussage über die Skalierbarkeit einer Anwendung.

2.5.2.6 Lastsensitivität

Über die Lastsensitivität wird definiert, wie stark sich das Antwortzeit- oder Durchsatzverhalten einer Anwendung verändert, wenn die Last auf der Anwendung zunimmt. Bei Anwendungen mit hoher Lastsensitivität verschlechtern sich Antwortzeit und Durchsatz schneller mit zunehmender Last als bei Anwendungen geringer Lastsensitivität. Die Lastsensitivität ist damit insbesondere bei Skalierbarkeitsbetrachtungen von hoher Bedeutung.

2.5.2.7 Effizienz

Die Effizienz einer Anwendung ergibt sich aus der erreichten Performance, dividiert durch die verwendeten Ressourcen. So ist eine Anwendung, die auf 4 CPU 40 tps erreicht, effizienter als eine Anwendung, die auf 8 CPU 70 tps erreicht.

2.5.2.8 Kapazität

Der maximale Durchsatz oder die maximale Last, die eine Anwendung verträgt oder erreichen kann, definiert ihre Kapazität. Je nach Anwendungsfall ist diese Kapazitätsgrenze erreicht sobald die Anwendung weitere Anfragen ablehnt oder der Grenzwert einer Performance-Kennzahl wie beispielsweise der Antwortzeit oder der Ressourcenauslastung überschritten wird.

2.5.2.9 Skalierbarkeit

Die Skalierbarkeit einer Anwendung ist ein Maß für die Kapazitätserweiterung, die durch das Hinzufügen weiterer Ressourcen erreicht werden kann. Bei der Skalierbarkeit sind zwei Varianten zu unterscheiden, die vertikale Skalierbarkeit und die horizontale Skalierbarkeit. Die vertikale Skalierbarkeit gibt die Kapazitätserweiterung an, wenn ein bestehender Anwendungsserver mit mehr Leistung versehen wird. Die horizontale Skalierung wiederum definiert sich aus der Kapazitätserweiterung beim Hinzufügen weiterer Server. Als optimale Skalierung wird die lineare Skalierung angesehen, bei der die Verdopplung der Ressourcen eine Verdopplung der Kapazität zur Folge hat. Eine lineare Skalierung ist in der Realität zumeist nicht zu erzielen. Häufig beschränken Schwachstellen wie gemeinsam genutzte Datenquellen, der ESB, oder andere limitierende Komponenten die Skalierbarkeit einer Anwendung.

2.5.3 Zusammenfassung

Teilkapitel 2.5 gibt eine Übersicht über die Eigenschaften von Unternehmensanwendungen. Unternehmensanwendungen bilden das in dieser Arbeit betrachtete Anwendungsgebiet, auch der in Kapitel 5 betrachtete Anwendungsfall ist einer Unternehmensanwendung entnommen. Bei der Darstellung der grundlegenden Eigenschaften von Unternehmensanwendungen erfolgt insbesondere eine Betrachtung der Eigenschaften, die sich auf die Performance einer Unternehmensanwendung auswirken.

Im Anschluss an die Darstellung der Eigenschaften erfolgt eine Definition der im Rahmen der Performance-Analyse einer Unternehmensanwendung relevanten Performance-Kennzahlen. Performance-Kennzahlen dienen zur Beschreibung des Performance-Verhaltens einer Anwendung. Der Abschnitt Performance-Kennzahlen gibt eine Übersicht möglicher Performance-Kennzahlen sowie deren Definition, und zeigt zudem die Zusammenhänge zwischen den Kennzahlen auf. Die Ausführungen in diesem Teilkapitel sind notwendige Voraussetzungen für ein Verständnis des in dieser Arbeit betrachteten Anwendungsfalls.

2.6 Aufbau einer Service-orientierten Architektur

Die experimentelle Überprüfung der in Teilkapitel 1.2 getroffenen Annahmen erfolgt anhand eines Feldexperiments an einer realen Unternehmensanwendung. Im Rahmen des Projekts ROBASO werden bei der Bundesagentur für Arbeit (im Folgenden BA genannt) über die Jahre gewachsene Legacy-Systeme über einen Enterprise Service Bus (ESB) zu einer Service-orientierten Architektur (SOA) integriert. Durch die Integration zu einer SOA verändert sich nicht nur das Aufrufverhalten auf den integrierten Anwendungen, auch kommen weitere Komponenten wie der ESB und die Facades hinzu, welche sich auf das

Performanceverhalten der Anwendung auswirken. Im Folgenden werden die grundlegenden Eigenschaften und Komponenten einer SOA beschrieben. Details zum ROBASO-Untersuchungsobjekt sind in Teilkapitel 5.1 zu finden.

2.6.1 Überblick über eine SOA

Eine SOA stellt eine Orchestrierung von neu geschaffenen oder bereits bestehenden Services zu einer lose gekoppelten Anwendung dar. Hauptaufgabe der SOA ist dabei die Bereitstellung von Lösungen zum Auffinden der Services sowie zur „Verhandlung“ des Kommunikationsvertrags (englisch: communications contract) zwischen dem Servicenutzer und dem Service (Hohpe/Woolf 2003d). Auch wenn Projekte existieren, in denen alle Services neu implementiert werden, so sind doch die sogenannten Enterprise Application Integration (EAI) Projekte dominant. In EAI-Projekte werden bestehende, fachlich zusammengehörende aber technisch getrennte Anwendungen im Rahmen einer SOA integriert (Hohpe/Woolf 2003d). Hierzu werden die nach Außen sichtbaren Operationen der zu integrierenden Anwendungen in Services aufgeteilt und über sogenannte Facades (Fowler 2003a) in der SOA verfügbar gemacht. Über den ESB können diese Services angesprochen werden, ohne auf die Implementierungsdetails der Anwendung eingehen zu müssen. In den folgenden Abschnitten werden die wichtigsten Konzepte einer SOA im Detail erläutert.

2.6.2 SOA-Konzepte

Nach Krafzig et al. (2005e) ist eine SOA eine Softwarearchitektur, die auf den Kernkonzepten Anwendungsoberfläche, Services, Service Repository und Service Bus basiert. Für das Verständnis einer SOA ist es elementar, diese grundlegenden Konzepte zu erläutern. Selbstverständlich umfasst diese Auflistung bei Weitem nicht alle Konzepte einer SOA, und selbst die betrachteten werden nur oberflächlich vorgestellt. Zum Verständnis dieser Arbeit und der Besonderheiten des verwendeten Anwendungsfalls ist diese Übersicht jedoch ausreichend. Für die weitergehende Lektüre zu EAI und SOA sind die bereits vorgestellten Arbeiten von Hohpe/Woolf (2003d) und Fowler (2003a) zu empfehlen.

2.6.2.1 Anwendungsoberfläche

Die Anwendungsoberfläche bildet die Schnittstelle zum Nutzer der SOA-Anwendung. Bei der Anwendungsoberfläche kann es sich um eine serverseitig implementierte Anwendung wie beispielsweise eine Web-Oberfläche oder um eine clientseitige Implementierung wie beispielsweise eine Windows-Anwendung handeln. Über die Anwendungsoberfläche werden Anfragen an die Services in Form von Messages an den ESB gestellt sowie Rückantworten, ebenfalls in Form von Messages, für den Nutzer verwendbar, beispielsweise in Form von Text, Grafiken, Sound oder Video, ausgegeben. Die Anwendungsoberfläche enthält oft

Prozessinformationen wie beispielsweise Prozessabläufe, welche zum Führen des Nutzers durch die Oberfläche verwendet werden. Auch Batch-Programme können nach Krafzig et al. (2005a) die Anwendungsoberfläche einer SOA bilden. In diesem Fall erfolgt nicht zwingend eine Ausgabe für den Nutzer.

2.6.2.2 Service

Ein Service ist ein extern zugreifbares, in sich abgeschlossenes Programmmodul (Krafzig et al. 2005c). Er besteht aus einem Vertrag, einem oder mehreren Interfaces, und einer Implementierung (Krafzig et al. 2005e). Im Rahmen einer SOA ist ein Service als ein „Business Service“ wie beispielsweise ein Flugbuchungsservice zu sehen, weniger als ein technischer Service wie der Auf- und Abbau einer Datenbankverbindung. Ein Service ist damit eine Gruppierung von fachlich zusammen gehörenden Operationen, die über ein einheitliches Interface anderen Anwendungen angeboten werden. Das einheitliche Interface, wie im Falle von Web Services beispielsweise die WSDL (Christensen et al. 2001), versteckt dabei Implementierungsdetails. Der Vertrag gibt an, zu welchen Konditionen auf den Service zugegriffen werden kann, und welche nicht-funktionalen Anforderungen vom Service erfüllt werden.

2.6.2.3 Service Repository

Ein Service Repository stellt die Hilfsmittel bereit, Services aufzufinden und alle Informationen zu erlangen um den Service zu nutzen, insbesondere wenn diese Services außerhalb des funktionalen und zeitlichen Rahmen des Projekts aufgefunden werden müssen, in dem sie erstellt wurden (Krafzig et al. 2005a). Das Repository ermöglicht damit die vollständige Loslösung von allen Bindungen an externe Services zu Design- und Implementierungszeit und das Auffinden benötigter Services zur Laufzeit. Die Standardimplementierung eines Service Repositories ist UDDI (OASIS 2012).

2.6.2.4 Service Bus

Der Service oder Messaging Bus ist das verbindende Element einer SOA. Über den Bus läuft sämtliche Kommunikation zu, zwischen und von den Services. Er ist eine Kombination aus einem kanonischen Datenmodell, einem vereinheitlichten Befehlssatz und einer Messaging-Infrastruktur, die es unterschiedlichen Systemen erlaubt, über eine Menge an gemeinsam genutzten Interfaces miteinander zu kommunizieren (Hohpe/Woolf 2003c). Neben der reinen Nachrichtenübertragung und dem damit verbundenen Routing bietet der Service Bus Mehrwertdienste wie Authentifizierung von Clients, Verschlüsselung, Message Translation und Fehlererkennung. Im Kontext von Unternehmensanwendungen wird oft das Präfix „Enterprise“ voran gestellt, wodurch die auch in dieser Arbeit verwendete Bezeichnung „Enterprise Service Bus“ entsteht.

2.6.2.5 Facade

Eine Facade (deutsch: Fassade) bildet eine einheitliche Schnittstelle zu den Operationen eines Subsystems. Die Facade definiert ein Interface auf einer höheren Ebene, das die Benutzung des Subsystems vereinfacht (Gamma 1995b). Im Kontext von SOA und EAI dient die Facade dazu, eine existierende Anwendung zu „SOA-fizieren“, sie also an den ESB anzubinden und ihre Operationen über den ESB als Services anzubieten. Die Facade übernimmt dabei die Übersetzung der proprietären Schnittstellen der Anwendung zum Messaging des ESB und umgekehrt. Die Aufteilung der Anwendungsoperationen auf fachlich getrennte Services erfolgt damit ebenfalls über die Facade. Sie enthält selber keine Geschäftslogik, sondern dient rein der Übersetzung und Anbindung. Im Rahmen von SOA findet oft auch der Channel Adapter (Hohpe/Woolf 2003a) als Spezialisierung der Facade Anwendung.

2.6.3 Vor- und Nachteile einer SOA

SOA als Architekturparadigma für Unternehmensanwendungen gewann in den vergangenen Jahren zunehmend an Bedeutung (Gartner Inc. 2011). Grund für diesen Bedeutungszuwachs ist vor allem die durch SOA ermöglichte Flexibilität, existierende Anwendungen mit relativ geringem Änderungsaufwand in die Architektur zu integrieren sowie Services und Prozessabläufe zur Laufzeit oder mit geringen Ausfallzeiten an veränderte Rahmenbedingungen (zum Beispiel Änderungen am Geschäftsprozess oder Gesetzesänderungen) anzupassen. Das SOA-Paradigma erlaubt zudem eine Vereinheitlichung der Softwarearchitektur eines Unternehmens, ohne die Notwendigkeit der Ersetzung oder des grundlegenden Architektur-Refactorings der bestehenden Systeme. Durch den Einsatz von Services können Funktionalitäten ohne die Notwendigkeit von spezifischen Anpassungen unternehmensweit angeboten werden.

Diesen Vorteilen gegenüber steht ein administrativer Mehraufwand durch die im Zuge der Service-Einführung zusätzlich eingefügten Service- und Middlewareschichten. Weiterhin führt die Kommunikation über den Service Bus und die Service-Fassaden zu einer Laufzeitverlängerung. Die Flexibilität und Strukturierung kommt auf Kosten der Laufzeit und der Komplexität.

2.6.4 Zusammenfassung

Das Konzept der Service-orientierten Architektur findet zunehmend Eingang in die Welt der Unternehmensanwendungen. Im Rahmen von Neuimplementierungen, insbesondere jedoch bei der Integration existierender Anwendungen, bietet das SOA-Paradigma zahlreiche Vorteile, beispielsweise durch eine erhöhte Flexibilität und die Reduktion des Integrations-

und Entwicklungsaufwands. Teilkapitel 2.6 führt in das Konzept der Service-orientierten Architekturen ein und stellt die grundlegenden Konzepte vor. Anschließend werden die Vor- und Nachteile des SOA-Paradigmas abgewogen, mit dem Ergebnis, dass in den meisten Fällen der Gewinn an Flexibilität den zusätzlichen Aufwand durch die Einführung von Facades und weiterer Middleware-Schichten übersteigt. Das Verständnis dieser grundlegenden Konzepte der Service-orientierten Architektur ist Voraussetzung für die Betrachtung des Anwendungsfalls in Kapitel 5.

2.7 Zusammenfassung

Dieses zweite Kapitel bietet eine Einführung in all die Grundlagen, die zum Verständnis dieser Arbeit benötigt werden. Im ersten Schritt erfolgt die Definition der abstrakten Problemstellung. Das Teilkapitel beschreibt die Differenzen zwischen dem Last- und Performance-Test auf der einen Seite und der Performance-Vorhersage auf der anderen. Es wird dargelegt, dass die Komplexität der betrachteten Unternehmensanwendungen und die begrenzten für die Performance-Analyse zu Verfügung stehenden Ressourcen die Messmöglichkeiten für die Performance stark einschränken. Dieser Beschränkung wird durch den Einsatz von Simulation entgegen gewirkt. Die existierenden Simulationsansätze benötigen jedoch sehr detaillierte Ressourcennutzungsinformationen für die einzelnen Softwarekomponenten. In der Praxis sind diese Informationen schwer bis gar nicht zu erlangen. Im ersten Teilkapitel wird dargelegt, dass für die Analyse der Performance von Unternehmensanwendungen eine Simulation auf Antwortzeit-Basis möglich sein muss.

Performance-Simulationsansätze spielen in dieser Arbeit eine wichtige Rolle. Zur Sicherstellung eines einheitlichen Verständnisses gibt das zweite Teilkapitel eine Übersicht über die Performance-Analyse durch Simulation im Allgemeinen. Hierzu werden Argumente für die Nutzung der Simulation aufgeführt – die vorab angesprochenen beschränkten Ressourcen bilden hiervon nur einen Punkt. Anschließend werden verschiedene grundlegende Simulationsansätze vorgestellt und begründet, weshalb in dieser Arbeit nur die Discrete-Event-Simulation zum Einsatz kommt. Eine Erörterung des Ablaufs der Simulation zeigt, dass es die Simulation im „weiten“ und im „engen“ Sinn gibt. Der weite Sinn umfasst die gesamte Simulationspipeline von der Modellerstellung über die Berechnung bis hin zur Validierung und Auswertung der Ergebnisse. Der enge Sinn hingegen umfasst ausschließlich den Berechnungsschritt. In der vorliegenden Arbeit geht die Bedeutung zumeist aus dem Kontext hervor, wobei die Simulation im engen Sinn als Simulationslauf bezeichnet wird. Im Anschluss erfolgt eine Betrachtung möglicher Simulationsziele, wobei in dieser Arbeit vor allem auf das Ziel „Architektur- und Testunterstützung zur Entwicklungszeit“ abgezielt wird. Die erreichbaren Ziele sind stark von der Modelltiefe abhängig. Diese wird im

darauffolgenden Teilkapitel erörtert mit dem Ergebnis, dass die Modelltiefe von zwei Faktoren abhängt: von dem geplanten Simulationsziel, aber insbesondere von der Verfügbarkeit der benötigten Informationen. So sollte kein Simulationsziel gewählt werden, für das nicht alle benötigten Anwendungsinformationen zu Verfügung stehen. Die Flexibilität der Simulationsansätze liefert auch in einem solchen Fall ein Ergebnis – die Verlässlichkeit des Ergebnisses ist jedoch zweifelhaft. Hieraus erfolgt ein natürlicher Übergang zu den Voraussetzungen und Limitationen der Simulation, die dieses Teilkapitel abschließen.

Im Anschluss an die Grundlagen der Performance-Simulation erfolgt die Einführung in das Thema *Komponentenmodelle*. Komponentenmodelle bilden das Performance-Verhalten einer Softwarekomponente ab. In dieser Arbeit werden ausschließlich Antwortzeit-Komponentenmodelle betrachtet. Das Teilkapitel geht auf die Aussagekraft eines Komponentenmodells ein, insbesondere jedoch auch auf seine Limitationen. So ist ein Komponentenmodell nur in dem Bereich zuverlässig, in dem ihm Messwerte zugrunde liegen. Wird ein Komponentenmodell für Eingabewerte berechnet, denen keine Messwerte zugrunde liegen, so gefährdet dies die Zuverlässigkeit der Simulationsergebnisse. Mit dem Konzept der Modellschablonen wird ein Konzept vorgestellt, um den Gültigkeitsbereich eines Komponentenmodells zu erweitern. Dieses Konzept wird in dieser Arbeit jedoch nicht weiter betrachtet.

Im Rahmen dieser Arbeit werden evolutionäre Algorithmen zur Modellapproximation mit dem Ziel der Generierung von Komponentenmodellen eingesetzt. Das grundlegende Konzept, der Aufbau und Ablauf eines evolutionären Algorithmus sowie seine Anwendbarkeit zur Modellierung wird in Teilkapitel 2.4 vorgestellt. Eine Ausprägung der evolutionären Algorithmen, die genetischen Algorithmen, eignet sich aufgrund ihrer Modellrepräsentation gut für die Modellierung von Komponentenmodellen. Die Grundkonzepte genetischer Algorithmen werden vorgestellt, mit einem Fokus auf die spezielle Modellrepräsentation. Dabei erfolgt eine umfangreiche Betrachtung der Konfigurationsparameter der genetischen Algorithmen, da diese im späteren Verlauf dieser Arbeit eine wichtige Rolle spielen werden.

Die beiden Teilkapitel zu Unternehmensanwendungen und Service-orientierten Architekturen erklären die Grundlagen zum Verständnis des in Kapitel 5 beschriebenen Anwendungsfalls. Anhand ihrer Eigenschaften wird aufgezeigt, weshalb sich die Performance-Analyse von Unternehmensanwendungen grundlegend von der Performance-Analyse wissenschaftlicher Anwendungen unterscheidet, für die bereits zahlreiche Ansätze existieren. Hierbei spielt insbesondere die enge Verzahnung mit den Geschäftsprozessen und dem Aufbau des Unternehmens eine große Rolle. Verschiedene Kennzahlen für die Performance einer

Unternehmensanwendung werden vorgestellt, anschließend wird erläutert, weshalb die Antwortzeit in dieser Betrachtung die relevante Kennzahl ist.

Die kritische Änderung, die aktuell am Anwendungsfall durchgeführt wird, ist die Integration zahlreicher Einzelanwendung in eine Service-orientierte Architektur. Diese Integration hat einen immensen Einfluss auf die Performance dieser Einzelanwendungen sowie der integrierten Lösung. Um diesen Einfluss verstehen zu können ist eine Betrachtung des SOA-Konzepts sowie der relevanten Einzelkonzepte notwendig. Dies geschieht im abschließenden Teilkapitel. Die einzelnen SOA-Konzepte werden vorgestellt und ihre Einbindung in das Gesamtkonzept erläutert. Anschließend erfolgt eine Abwägung der Vor- und Nachteile einer SOA. Diese Abwägung erlaubt einen Einblick in die grundlegende Veränderung, welche die Integration in eine SOA mit sich bringt.

3 Verwandte Forschungsarbeiten

Die in diesem Kapitel durchgeführte Literaturanalyse dient der Einordnung der vorliegenden Arbeit in die Forschungslandschaft sowie der Definition und Abgrenzung der Forschungslücke, welche durch diese Arbeit geschlossen wird. Hierzu wird zunächst in Teilkapitel 3.1 das Vorgehen zur Erhebung und Analyse der Literatur dargelegt. Anschließend erfolgt in je einem Teilkapitel die Analyse der Literatur zu den Kernthemen der vorliegenden Arbeit. Die Aufteilung folgt dabei grob der Strukturierung von Nudd et al. (2000): „Konventionelle Performance-Evaluationstechniken sind aufgeteilt in drei Hauptklassen: Performance-Messung, Performance-Modellierung, und Simulation". Die Performance-Messung ist nicht Bestandteil dieser Arbeit, sie wurde bereits, neben anderen, von Jehle (2010) und Bögelsack (2011) intensiv bearbeitet, so dass sie im Rahmen dieser Literaturanalyse nicht betrachtet wird. Hinzugenommen werden die relevanten Literaturquellen zur Performance-Modellierung mittels evolutionärer Algorithmen.

Teilkapitel 3.2 bietet eine Übersicht über die in der Literatur beschriebenen Verfahren zur Performance-Modellierung von Softwaresystemen. Hierbei werden explizit Verfahren betrachtet, die das Performance-Verhalten von Softwaresystemen im Ganzen modellieren. Eine Betrachtung der Komponentenmodellierung erfolgt in einem späteren Teilkapitel. Teilkapitel 3.3 legt die existierende Literatur zu Performance-Simulationsansätzen dar. Die Simulationsansätze für High-Performance-Computing werden hier ebenso betrachtet wie die Simulationsansätze für Unternehmensanwendungen. Letztere bilden die Grundlage für die Auswahl eines Simulationsansatzes für das in dieser Arbeit beschriebene kontrollierte Softwareexperiment und damit für die Beantwortung der ersten Forschungsfrage. In Teilkapitel 3.4 wird eine Übersicht über die in der Literatur beschriebenen Verfahren zur Modellbildung auf Basis von Komponenten-Messdaten eines Softwaresystems vorgestellt. In diesem Teilkapitel erfolgt eine Analyse der bestehenden Ansätze zur Abbildung des Komponentenverhaltens. Weiterhin wird analysiert, ob und in welchem Umfang Literatur zum Einsatz evolutionärer Algorithmen zur Modellierung des Performance-Verhaltens von Softwaresystemen und ihren Komponenten existiert. Abschließend folgt in Teilkapitel 3.5 eine Zusammenfassung mit der Darstellung aller zuvor erfassten Ergebnisse der Arbeiten im Überblick.

3.1 Vorgehen

Ziel dieses Kapitels ist die Darlegung des wissenschaftlichen Kontextes dieser Arbeit. Die hierzu durchgeführte Literaturanalyse hat den Anspruch, nachvollziehbar und wiederholbar

zu sein. Um die Nachvollziehbarkeit und Wiederholbarkeit zu gewährleisten wird im Folgenden das Vorgehen zu dieser Literaturanalyse erläutert.

3.1.1 Überblick

Das Vorgehen zur Analyse der Literatur folgt den Vorgaben von Webster und Watson (2002) sowie den Richtlinien von Torraco (2005) und Kitchenham und Charters (2007). Fettke (2006) folgend ergibt sich die in Tabelle 2 beschriebene Einordnung der Literaturanalyse.

Tabelle 2. Einordnung der Literaturanalyse nach Fettke (2006).

Charakteristik		**Kategorie**
Typ		Natürlichsprachlich
Fokus		Forschungsergebnis
Ziel	Formulierung	Nicht expliziert
	Inhalt	Kritik und zentrale Themen
Perspektive		neutral
Literatur	Auswahl	Expliziert
	Umfang	Repräsentativ
Struktur		Thematisch
Zielgruppe		Spezialisierte Forscher
Zukünftige Forschung		Nicht expliziert

Die Literaturanalyse zielt auf eine Darlegung des aktuellen Forschungsstands in den genannten Themengebieten ab. Eine Bewertung der einzelnen Arbeiten erfolgt nicht, jedoch ein Vergleich der Anwendbarkeit der dokumentierten Ergebnisse auf das Anwendungsfeld dieser Arbeit.

3.1.2 Datenquellen

Neben der üblichen Bibliothekssuche bietet sich in der schnelllebigen Welt der Informationssysteme das Internet als Informationsquelle an. Aufgrund der rasanten Entwicklungsgeschwindigkeit erfolgen zahlreiche Veröffentlichungen im Bereich der Performance von Unternehmensanwendungen auf Konferenzen und in elektronischen Journalen. Zum Auffinden dieser sowie von Beiträgen in klassischen Journalen und Büchern eignen sich vor allem die online verfügbaren wissenschaftlichen Datenbanken und Suchmaschinen.

Die wissenschaftlichen Datenbanken ermöglichen einen effizienten und umfassenden Zugriff auf Konferenzbeiträge, Journals und Bücher. Die in den Datenbanken gefundenen Informationen werden ergänzt durch Universitätswebseiten, White Papers und Reports, die durch Suchmaschinen aufgefunden werden können. Auf diese Weise erfolgt eine Analyse der wissenschaftlichen Basis, ergänzt um den aktuellen Stand der Wissenschaft und Technik.

Von den online verfügbaren Datenbanken sind, Hossain et al. (2009) folgend, die folgenden zur Literaturrecherche herangezogen worden:

- AIS eLibrary (http://aisel.aisnet.org/)
- ACM Digital Library (http://portal.acm.org/portal.cfm)
- Elsevier Science Direct (http://www.sciencedirect.com/)
- IEEE Explorer (http://ieeexplore.ieee.org/)
- SpringerLink (http://springerlink.com/computer-science/)

Zusätzlich zu der Suche in den wissenschaftlichen Datenbanken erfolgt eine Suche mittels zweier Suchmaschinen, welche als Quellen neben den wissenschaftlichen Artikeln auch nichtwissenschaftliche Beiträge (technische Reports, White Paper etc.) umfassen:

- Google Scholar (http://www.google.de/scholar)
- CiteseerX (http://citeseerx.ist.psu.edu/)

Bei der Suche mithilfe des Internets ergibt sich aufgrund von Querverweisen in den Datenbanken und der Verfügbarkeit verschiedener Versionen von Veröffentlichungen eine zu bereinigende Ergebnismenge. So enthält die ACM Digital Library einige (jedoch nicht alle) Artikel der IEEE-Datenbank. Die in der ACM Digital Library enthaltenen IEEE-Publikationen sind vollständig in der Menge der Suchergebnisse aus dem IEEE Explorer enthalten. Zudem liefern Google Scholar- und CiteseerX-Suchen zuweilen Vorab-Versionen von Veröffentlichungen, die von Autorenwebseiten stammen. Diese Veröffentlichungen sind, in Form der offiziellen Veröffentlichung, ebenfalls in den Ergebnismengen der Datenbanken enthalten. Im Zuge der Recherche werden solche Duplikate herausgefiltert.

3.1.3 Suchstrategie

Die Literatursuche erfolgt anhand von Suchbegriffen, die in drei Klassen eingeteilt werden. Zu jeder Klasse erfolgt ein Suchdurchlauf in allen vorab aufgeführten Datenquellen. Die Ergebnisse der Suche werden je Durchlauf gesammelt und bewertet, Duplikate werden

entfernt. Anschließend werden, soweit vorhanden, die Referenzen der gefundenen Artikel analysiert und als zusätzliche Datenquelle betrachtet. Durch dieses Verfahren entsteht eine Art Schneeballprinzip – die Referenzen von als wertvoll erachteten Artikeln dienen als Grundlage für die weitere Suche.

Die Klassifizierung der Suchbegriffe erfolgt anhand der einleitend dargestellten drei Bereiche der Literaturanalyse: *Performance-Modellierung*, *Performance-Simulation* und *Evolutionäre Algorithmen zur mathematischen Modellierung*. Hierdurch ergeben sich die folgenden drei Klassen mit Suchbegriffen:

- Klasse 1: Software + Performance + Model; Software + Performance + Modeling
- Klasse 2: Software + Performance + Simulation; Information + System + Performance + Simulation; EIS + Performance + Simulation
- Klasse 3: Evolutionary + Algorithm + Mathematical + Modeling; Genetic + Algorithm + Mathematical + Modeling; Evolutionary + Algorithm + Software + Performance; Genetic + Algorithm + Software + Performance

Die Suche erfolgt mittels AND-Verknüpfung der angegebenen Suchbegriffe. Der Suchbereich wurde auf Titel, Abstract und Keywords der Publikationen beschränkt, soweit dies die Suchmaske erlaubt. Hierdurch wird eine deutliche Verbesserung der Ergebnis-Relevanz erreicht. Eine Volltextsuche erzeugt im Vergleich deutlich mehr Ergebnisse, die nicht oder nur peripher mit den genannten Themen in Verbindung stehen, ohne zusätzliche relevante Ergebnisse zu liefern.

3.1.4 Selektion

Die im vorherigen Abschnitt beschriebene Literatursuche resultiert in einer Ergebnismenge von über fünfhundert Arbeiten. Die Auswahl der relevanten Veröffentlichungen aus der Ergebnismenge geschieht nach einem Phasenmodell, das an das Vorgehen von (Hossain et al. 2009) angelehnt und ähnlich von (Bögelsack 2011) durchgeführt wurde:

- Phase 1: Löschen aller Duplikate und nicht wissenschaftlicher Beiträge
- Phase 2: Selektion der Beiträge mit Bezug zur Performance von Softwaresystemen und evolutionären Algorithmen

- Phase 3: Löschen der Beiträge mit Performance zu eingebetteten Systemen

In Phase 1 werden aus der Ergebnismenge alle doppelten Vorkommnisse herausgefiltert. Hierbei werden auch in mehreren Versionen vorkommende Publikationen identifiziert. Die in den Datenbanken vorliegenden Versionen werden dabei den Versionen aus Suchmaschinen-Ergebnissen vorgezogen. Die Filterung ist vor dem Hintergrund der genutzten Suchmaschinen wichtig, da diese auch die Beiträge der separat durchsuchten Datenbanken indizieren. Durch das Löschen der Duplikate und der nicht wissenschaftlichen Literatur reduziert sich die Ergebnismenge auf 223 Publikationen.

In Phase 2 erfolgt eine Selektion anhand des Inhalts der Publikationen. Neben der technischen Performance von Softwaresystemen existieren zahlreiche Publikationen, welche auf die betriebswirtschaftliche Performance abzielen. Diese sind nicht Bestandteil dieser Untersuchung und werden nicht berücksichtigt. Durch das Selektieren reduziert sich die Zahl der Quellen auf 196.

Phase 3 umfasst die Löschung der Veröffentlichungen, die sich zwar mit der Performance von Softwaresystemen beschäftigen, dies aber speziell mit Fokus auf eingebettete Systeme oder Hardware- und Firmware-Komponenten tun. Zahlreiche Arbeiten im Bereich der Software-Performance befassen sich mit der Performance-Analyse von eingebetteten Systemen (englisch: embedded systems) oder der Auswirkung spezieller Hardware-Komponenten (wie beispielsweise spezieller Prozessortypen) auf die Performance einer Programmiersprache oder Anwendung. Da sich die vorliegende Arbeit mit Unternehmensanwendungen beschäftigt sind diese Publikationen nicht relevant und werden aussortiert. Weiterhin werden alle Arbeiten aussortiert, die sich primär mit der Messung des Performance-Verhaltens beschäftigen. Die Messung wird in Arbeiten wie (Jehle 2010) oder (Bögelsack 2011) intensiv behandelt, diese Arbeiten bieten auch umfangreiche Übersichten über die relevante Literatur. Für die in der vorliegenden Arbeit betrachteten Themenfelder ist die korrekte Messung zwar als Grundlage relevant, sie wird jedoch als gegeben angenommen und nicht weiter verfolgt.

Durch die Reduzierung in der letzten Phase ergibt sich die Anzahl von 39 zu analysierenden Veröffentlichungen.

3.1.5 Analyse

Die Analyse der 39 identifizierten Quellen erfolgt, in Anlehnung an (Bögelsack 2011), in drei Kategorien:

- Kategorie 1: Performance-Modellierung für Softwaresysteme
- Kategorie 2: Performance-Simulation
- Kategorie 3: Modellbildung von Komponentenverhalten auf Messdaten

Die vorgestellten Kategorien decken das gesamte Anwendungsfeld dieser Arbeit ab. Das Ziel der Kategorisierung ist die Darstellung der relevanten Arbeiten in diesen Feldern und damit die Einbettung dieser Arbeit in den Forschungskontext. Aus der Analyse der Veröffentlichungen wird ersichtlich, dass keiner der existierenden Performance-Simulations- und Modellierungs-Ansätze die Brücke zwischen den Messwerten und der Analyse schlägt. Die Literaturanalyse zeigt damit die von dieser Arbeit bearbeitete Forschungslücke auf. Weiterhin stellt die Analyse der Literatur in Kategorie 2 die existierenden Ansätze zur Performance-Simulation dar und ermöglicht damit die Beantwortung der zweiten Forschungsfrage.

3.2 Performance-Modellierung für Softwaresysteme

Die Performance-Modellierung eines Softwaresystems dient der Vorhersage des Performance-Verhaltens des Systems mittels mathematischer Berechnungen. Anders als bei der Simulation werden bei der Performance-Modellierung nicht einzelne (Zeit-)Schritte berechnet. Vielmehr wird das Modell als Ganzes gelöst.

Für die Performance-Modellierung existieren verschiedene Definitionen und Auffassungen. Diese werden im Folgenden vorgestellt. Anschließend erfolgt eine Übersicht über die in der Literatur auffindbaren Ansätze zur Performance-Modellierungen sowie der existierenden Frameworks und Implementierungen. In diesem Teilkapitel erfolgt ausschließlich die Betrachtung der Performance-Modellierung eines Softwaresystems im Ganzen. Die Betrachtung der Modellierung von Softwarekomponenten für die Simulation erfolgt in Teilkapitel 3.4.

3.2.1 Definitionen der Performance-Modellierung

Performance-Modellierung für Softwaresysteme wird in der Literatur auf verschiedene Weise definiert. Eine sehr abstrakte und umfassende Definition wird von Rolia und Sevcik (1995) gegeben. Die Autoren definieren Performance-Modellierung als “[…] eine Methode um herauszufinden, welche der zahlreichen Eigenschaften eines Systems die Performance beschränken, indem sie in einem Vorhersagemodell abgebildet werden”. Hollingsworth et al.

(2005) hingegen definieren ein Performance-Modell als “[…] einen berechenbaren Ausdruck, der erklärt und quantifiziert, warum eine Anwendung auf einer Maschine performt wie sie es tut”.

Das Ziel der Performance-Modellierung definieren Bailey und Snavely (2005) als “[…] das Gewinnen von Verständnis von der Performance eines Computersystems unter verschiedenen Anwendungen, unter Verwendung von Messungen und Analyse, und anschließend die Zusammenfassung dieser Eigenschaften in einer kompakten Formel“. Kerbyson und Jones (2005) erklären, dass “Performance-Modelle zum Beispiel genutzt werden können, um die Performance von Systemen zu untersuchen, ohne die Notwendigkeit von ausführlichen empirischen Untersuchungen, zur Unterstützung der Identifikation von Performance-Schwachstellen, und zur Bereitstellung von Informationen zum Tuning einer Anwendung für ein spezielles System”. Die Autoren betonen zudem die Vorteile der Performance-Modellierung im Vergleich zum Benchmarking, zum Beispiel die Verwendbarkeit günstigerer small-scale-Systeme anstelle von large-scale-Systemen für die Betrachtung, und die Anwendbarkeit von Modellen für Situationen, die nicht messbar sind, zum Beispiel zum Erörtern von Designalternativen in zukünftigen Maschinen oder zur Effektanalyse von Codeänderungen vor deren Umsetzung.

3.2.2 Ansätze zur Performance-Modellierung

Die *Clisspe*-Sprache (*Client Server Software Performance Engineering*), entwickelt von Menascé (1997), ermöglicht Entwicklern neuer Anwendungen die Untersuchung der Performance ihrer Anwendungen im Entwicklungsstadium. Die Sprache erlaubt die Spezifikation von Objekten wie beispielsweise Clients, Server, Datenbanktabellen, Netzwerke und Transaktionen, und kann in ein analytisches Queuing Network Modell eines Client-Server-Systems kompiliert werden. In (Menascé/Gomaa 1998) erweitern die Autoren ihren Ansatz und präsentieren dessen Anwendung in einem Kapazitätsplanungsszenario. Obwohl sich Layered Queuing Networks, eine Erweiterung der Queuing Networks, in der Praxis gegenüber traditionellen Queuing Networks durchgesetzt haben sind die vorgestellten Publikationen von Menascé als Kernliteratur für das Verständnis der Historie der Performance-Modellierung relevant.

(Liu et al. 2004) stellen einen auf der Queuing-Theorie (Gorton et al. 2003; Jacobson/Lazowska 1982) basierenden Ansatz zur Performance-Vorhersage von Middleware-basierten Anwendungssystemen während der Entwurfszeit vor. Der vorgestellte Ansatz nutzt analytische Performance-Modelle der Anwendung, der Middleware sowie der zugrunde liegenden Hardware, welche kombiniert und analytisch gesolved werden. Die

Anwendung des Ansatzes wird an einer Fallstudie, der Vorhersage der Stock-Online-Anwendung (Gomaa/Menascé 2000), exemplarisch vorgestellt. Als Solving-Methode für die Performance-Modelle kommt der *Approximate Mean Value Analysis*-Algorithmus (Liu et al. 2003) zum Einsatz.

In der Publikation „Performance Modeling for Service Oriented Architectures" präsentieren Brebner et al. (2008) kritische Anforderungen an ein Performance-Modellierungswerkzeug. Die Autoren kommen zu dem Schluss, dass die existierenden Anwendungen, ob Open Source oder kommerziell, diese Anforderungen nicht erfüllen, und stellen ihrerseits eine Performance-Modellierungswerkzeug vor, das die Anforderungen vollständig abdeckt. Das von Brebner et al. vorgestellte Werkzeug erlaubt die Performance-Analyse von SOA-Systemen basierend auf Architekturartefakten wie beispielsweise UML-Diagrammen, Servicelandkarten, Hardwarediagrammen und anderen. In weiteren Arbeiten (Brebner 2009; Brebner et al. 2009) erweitern die Autoren ihren Performance-Modellierungsansatz, welcher zudem den Namen *Service-Oriented Performance Modelling* (SOPM) erhält, zu einem vollwertigen Werkzeug für die Performance-Modellierung von heterogenen large-scale SOA-Systemen. Die Anwendbarkeit ihres Ansatzes verifizieren Brebner et al. an real-world SOA-Anwendungen wie der Mule Enterprise Service Bus (ESB) Loan Broker Application (MuleSoft 2012) und vier weiteren Fallstudien.

3.2.3 Frameworks

Rolia et al. (2009) setzen Layered Queuing Models (LQM, (Franks et al. 2009; Rolia/Sevcik 1995)) für die Modellierung des Performance-Verhaltens von SAP ERP-Systemen ein. Die Autoren heben in ihrer Veröffentlichung die Eignung von LQM für die Performance-Modellierung hervor und benennen die Vorteile von LQM gegenüber den elementaren product-form Queuing Network Models (QNMs, (Kienzle/Sevcik 1979)). Weiterhin belegen Rolia et al. ihre Aussagen durch eine Fallstudie an einer *Sales and Distribution* Anwendung.

Der Ansatz der Modellierung des Performance-Verhaltens von ERP-Systemen mittels Layered Queuing Networks (LQN) wird ebenfalls von Gradl et al. (2009) verfolgt. Gradl et al. bieten in ihrer Publikation eine Kurzübersicht über die Architektur eines ERP Systems und geben eine Anleitung zur Entwicklung eines passenden LQN für die betrachteten Systeme. An einem exemplarischen ERP-System demonstrieren die Autoren die Anwendung des von (Woodside 2002) und (Franks et al. 2005) entwickelten Layered Queueing Network Solver (LQNS). Die Veröffentlichung von Gradl et al. stellt einen guten Einstiegspunkt in die Verwendung von LQN für die Performance-Analyse komplexer Softwaresysteme dar. Der Ansatz wird in (Gradl 2012) ausführlich behandelt und erweitert. In dieser zweiten Publikation

wird zudem der Aufbau des betrachteten ERP-Systems verstärkt betrachtet und sein Einfluss auf das Performance-Verhalten erläutert.

Nudd et al. (2000) stellen das Framework PACE (Performance Analysis and Characterization Environment) vor, ein Performance-Evaluations – und Voraussageframework für parallele und verteilte Systeme, die in C/C++ oder FORTRAN geschrieben sind. Die in PACE implementierte Methode deckt den Softwaredesign- und Implementierungzyklus ab und "nutzt die besten Eigenschaften sowohl der Messung als auch des analytischen Modellierens und ermöglicht so die Durchführung qualitativ hochwertiger Analysen für die Performance-Evaluation und -Voraussage", wie die Autoren betonen. Der Artikel bietet eine Übersicht über die Kernidee und Methodik von PACE, die tatsächliche Implementierung des Frameworks und des Performance-Vorhersageprozesses, inklusive der genutzten Methoden für die Analyse und Vorhersage. An mehreren Beispielen illustrieren die Autoren die Anwendung von PACE. PACE erfährt eine kontinuierliche Weiterentwicklung und wird von (Jarvis et al. 2006) aktualisiert beschrieben.

Im Gegensatz zur Antwortzeit- und Durchsatzanalyse der vorab vorgestellten Performance-Analyseframeworks konzentriert sich der SCALability Analyzer (SCALA) von Sun et al. (2002) auf die Skalierung von Softwaresystemen, insbesondere von HPC-Anwendungen. Basierend auf der Integration von symbolischen Kostenmodellen und dynamischen, zur Laufzeit erhobenen Metriken, berechnet SCALA die Skalierung von hochperformanten, parallelisierten und verteilten Anwendungen. Die Publikation von Sun et al. bietet eine umfassende Übersicht über die Einflussfaktoren auf die Skalierbarkeit von komplexen, verteilten Softwaresystemen, konzentriert auf primär wissenschaftliche Anwendungen.

3.2.4 Zusammenfassung

Betrachtet man die vorab vorgestellten Ansätze und Frameworks zur Performance-Modellierung von Softwaresystemen, so wird ersichtlich, dass sich diese hauptsächlich mit der Performance von HPC-Anwendungen befassen. Einzig die Layered Queuing Networks bilden hier eine Ausnahme. Der Grund für diese Beschränkung liegt in der Komplexität der betrachteten Systeme. Während HPC-Anwendungen vergleichsweise kleine Programme darstellen, die hoch parallel auf zahlreichen Prozessoren ausgeführt werden, sind Unternehmensanwendungen deutlich komplexer in ihrem Aufbau. Aus der gefunden Literatur lässt sich schließen, dass die Performance-Modellierung für Unternehmensanwendungen zwar möglich, aber nur in wenigen Anwendungsfällen geeignet ist. Die analytische Untersuchung des Performance-Verhaltens von Unternehmensanwendungen wird deshalb in dieser Arbeit nicht weiter verfolgt.

3.3 Performance-Simulation

Die Simulation bildet neben dem analytischen Solving von Performance-Modellen den zweiten großen Ansatz zur Analyse des Performance-Verhaltens eines Softwaresystems. Softwaresysteme, insbesondere Unternehmensanwendungen, bilden komplexe Systeme mit zahlreichen Beziehungen zwischen den einzelnen Elementen und ebenso zahlreichen, das Performance-Verhalten beeinflussenden, Faktoren. Damit gleichen sie von der Struktur her anderen Systemen, wie beispielsweise Verkehrs- und Stromnetzen, deren Verhalten ebenfalls durch Simulation vorhergesagt werden kann (vgl. z.B. (Boel/Mihaylova 2006)). Die Simulation bietet sich damit für die Performance-Analyse eines Softwaresystems an.

Dieses Teilkapitel stellt einige der grundlegenden Arbeiten zur Simulation komplexer Systeme im Allgemeinen und der Performance von Softwaresystemen im Speziellen vor. Nach einer Betrachtung grundlegender Arbeiten liegt der Fokus vor allem auf der Analyse von Publikationen, die Frameworks für die Simulation komplexer Softwaresysteme vorstellen. Diese Betrachtung dient der Beantwortung der ersten Forschungsfrage.

3.3.1 Grundlagen

(Bossel 2004) gibt einen umfangreichen Überblick über die Erstellung von Simulationsmodellen sowie die Simulationsdurchführung von komplexen Systemen. Bossel betrachtet dabei komplexe Systeme im Allgemeinen, ohne sich auf eine bestimmte Menge (wie z.B. Computersysteme) einzuschränken. Hierdurch entsteht eine Übersicht über die Möglichkeiten, Voraussetzungen und erzielbaren Ergebnisse durch Simulation im Generellen. Der Autor geht dabei auf den Aufbau komplexer Systeme ein und beschreibt, wodurch sich ein Zustand auszeichnet (bezeichnet als *statische Sicht*). Anschließend erfolgt die Betrachtung des Systemverhaltens in Form der dynamischen Sicht und eine Übersicht über mögliche Wege, diese Systeme zu analysieren. Abgeschlossen wird die Arbeit durch eine Betrachtung verschiedenster komplexer Systeme, von Elementarsystemen wie des Ansteckungsverhaltens in einer Population bis hin zu globalen Entwicklungen wie der Schuldenkrise.

Die Grundlagen der Simulation zur Analyse von Softwaresystemen werden ausführlich im in Teilkapitel 2.2 bereits erwähnten Buch von (Bungartz et al. 2009) erläutert. Der Autor führt eine umfassende Analyse des Aufbaus, Ablaufs und der zu erzielenden Ergebnisse einer Simulation durch und führt ihre Einschränkungen und Herausforderungen auf. Die ebenfalls bereits erwähnte Unterscheidung in die Simulation „im weiteren Sinne“ sowie die Simulation im „engeren Sinne“ erlaubt die Betrachtung der Simulation sowohl als Prozess als auch als technisch-mathematischen Vorgang.

Bögelsack et al. (2008) beschreiben Regeln für die Entwicklung eines Simulationsmodells für komplexe ERP-Systeme. Die Autoren konzentrieren sich in der Publikation auf die Entwicklung einer adäquaten Struktur zur Repräsentation der Architektur von SOA-basierten ERP-Unternehmensanwendungen. Hierzu brechen die Autoren das ERP-System in verschiedene Schichten auf und beschreiben, welche Faktoren die Performance des Systems beeinflussen. Die Publikation bietet eine solide theoretische Basis für die Entwicklung eines Performance-Simulationsframeworks.

3.3.2 Frameworks

Eine Discrete-Event-Simulation mit mathematischen Modellen zur Abbildung des Komponentenverhaltens werden in (Pllana et al. 2008) vorgestellt. Mit einem Fokus auf Großrechnersysteme und HPC zeigen die Autoren auf, wie hybride Ansätze aus Performance-Modellierung und Performance-Vorhersage, die Kombination von mathematischer Modellierung und Simulation, durchgeführt werden können. Pllana et al. implementieren ihren Ansatz in dem Performance-Vorhersageframework *Performance Prophet* und evaluieren ihn in einer Fallstudie.

Bourgeois und Spies (1999, 2000) stellen die ChronosMix-Umgebung vor, ein Werkzeug zur Voraussage der Ausführungszeit eines verteilten Algorithmus auf Systemen mit paralleler oder verteilter Architektur. Das Ziel von ChronosMix ist “die Bereitstellung einer so vollständigen Performance-Voraussageumgebung wie nur möglich”. Um dieses Ziel zu erreichen nutzt ChronosMix einen Simulationsansatz, der auf einer parallelen Architektur und einem Programm-Modell basiert. Bourgeois und Spies präsentieren in ihrer Publikation die Anwendung von ChronosMix in der Vorhersage des NAS Integer Sorting Benchmarks. ChronosMix fokussiert, ähnlich dem vorab vorgestellten Performance Prophet, auf wissenschaftliche Anwendungen.

Der COMponent-based Parallel System Simulator (COMPASS), entwickelt von Bagrodia et al. (1999), ist ein Discrete Event Simulator für das Performance-Verhalten von High-Performance-Computing-Systemen (HPC). COMPASS zielt auf die Performance-Vorhersage von großen, parallelen Programmen ab, basierend auf der Betrachtung von CPU- und I/O-Nutzung. Die Autoren demonstrieren die Anwendbarkeit von COMPASS an Realweltanwendungen und synthetischen Benchmarks. COMPASS last sich aufgrund seiner Fokussierung auf HPC-Systeme nicht ohne Anpassungen auf Unternehmensanwendungen anwenden, die Publikation gibt jedoch einen guten Überblick über die Architektur von Performance-Simulationsansätzen und Performance-Benchmarks.

Ein generischer Ansatz zur Vorhersage der Performance von HPC-Systemen wird von Carrington et al. (2002; 2006) vorgestellt. Benchmarks wie der LINPACK-Benchmark (Dongarra 1988) werden zur Erstellung von Anwendungs- und Maschinenprofilen genutzt, welche wiederum die Basis für die Simulation des Performance-Verhaltens des zu testenden Systems bilden. Carrington et al. zeigen Faktoren auf, die das Performance-Verhalten von HPC-Systemen maßgeblich beeinflussen, und zeigen die Effektivität ihres Ansatzes an verschiedenen Fallstudien. Die Publikationen von Carrington et al. sind sehr detailliert und richten sich an Leser mit umfangreicher HPC-Erfahrung.

Hollingsworth et al. (2005) kombinieren das “automatisierte Wesen der direkt ausführbaren Simulation mit dem Vorhersagevermögen der Performance-Modellierung” in ihrem Framework EMPS (Environment for Memory Performance Studies). Das Framework wurde mit dem Ziel entworfen, verschiedene Werkzeuge zum Sammeln von Performance-Informationen mit Simulationswerkzeugen zu integrieren, um das Performance-Verhalten von parallelen Programmen auf aktuellen und zukünftigen High End Computing (HEC)-Systemen vorherzusagen. EMPS basiert auf verschiedenen existierenden Techniken wie Dyninst (Buck/Hollingsworth 2000) und Sigma (DeRose et al. 2002) zur Akquise der Speicherinformationen, und MetaSim (Carrington et al. 2002) zur Modellextraktion aus den Speicherinformationen.

Der im vorherigen Abschnitt bereits vorgestellte Layered Queuing-Ansatz von (Woodside 2002) und (Franks et al. 2005) bietet neben dem analytischen Solver LQNS auch den Simulator LQSIM. Mittels dieses Simulators können die vorab bereits vorgestellten Layered Queuing-Modelle von Softwaresystemen mittels Simulation zur Performance-Analyse genutzt werden. Dies ist insbesondere dann interessant, wenn die Komplexität der betrachteten Anwendung oder des Nutzerverhaltens ein Solving unmöglich macht. Dies ist bei den meisten Unternehmensanwendungen der Fall.

In (Gradl 2012) baut Gradl den im vorherigen Abschnitt vorgestellten Ansatz der Modellierung und Simulation eines ERP-Systems mittels LQN aus. Der Autor fokussiert sich in seiner Arbeit auf den SAP Web Application Server ABAP und nutzt ein auf der Mess- und Warteschlangentheorie basierendes LQN-Modell sowie eine aus Mess- und Analysewerten entwickelte Parametrisierung zur Vorhersage des Antwortzeitverhaltens des betrachteten Systems unter einer steigenden Anzahl paralleler Benutzer. Im Gegensatz zu den im vorherigen Abschnitt vorgestellten Arbeiten des Autors kommt in (Gradl 2012) eine Discrete-Event-Simulation zum Einsatz. Gradl zeigt, dass sich mittels des vorgestellten Ansatzes

belastbare Vorhersagen für den Niedrig- und Mittellastbereich erzielen lassen. Im Hochlastbereich hingegen wirken externe, schwer modellierbare Faktoren auf die Antwortzeit, wodurch eine belastbare Vorhersage stark erschwert wird.

Einen ähnlichen Ansatz verfolgt (Mayer 2013). Im Gegensatz zu Gradl fokussiert sich der Autor jedoch auf ein Java-Portalsystem, explizit das SAP-Netweaver-Portal-System. Auch Mayer bildet das betrachtete System in Form eines parametrisierten LQN-Modells ab und erweitert dieses um die speziellen Architekturmerkmale des SAP-Netweaver-Portal-Systems. Die erzielten Simulationsergebnisse bestätigen die von Gradl getroffene Aussage, dass sich mittels Discrete-Event-Simulation das Verhalten komplexer ERP-Systeme im Niedrig- und Mittellastbereich exakt abbilden lässt, im Hochlastbereich jedoch nicht modellierbare Einflüsse zu stark zunehmen, um belastbare Ergebnisse zu erzielen. Weiterhin zeigt die Arbeit die Portierbarkeit des Ansatzes von ABAP-basierten Systemen auf ein Java-System.

(Reussner et al. 2007) sowie Becker et al. (1996; 2001; 2004) führen in ihren Veröffentlichungen das Palladio Component Model (PCM) ein, ein Performance-Simulationsansatz und -Framework, welches den komponentenbasierten Software-Engineering-Entwicklungsansatz (Component-based software engineering, CBSE) unterstützt. PCM ist ein Meta-Modell zur Beschreibung von komponentenbasierten Softwarearchitekturen mit einem Fokus auf der Vorhersage von Quality-of-Service (QoS) Eigenschaften, insbesondere der Performance und der Zuverlässigkeit. Die Performance-Simulation führt PCM mittels der Simulations-Engine SimuCom durch, einer Eigenentwicklung, die auf dem Java-Simulationsframework SSJ (L'Ecuyer et al. 2002) aufbaut. Die Performance-Analyse mittels PCM kann zudem mit zahlreichen anderen Solvern durchgeführt werden (vgl. (Software Design and Quality Group 2012)). Zur Evaluation des PCM-Konzepts führten Becker et al. Fallstudien unter anderem auf dem “Web Audio Store” durch, wobei sie eine auf dem Eclipse-Framework (The Eclipse Foundation 2011) basierte Implementierung des Ansatzes verwenden.

Eine umfangreiche Abhandlung über das Durchführen von Performance-Vorhersagen von Message-orientierten Middlewaresystemen (MOM) mittels Simulation ist in (Happe et al. 2010) beschrieben. Martens et al. (2001) sowie Kapová und Becker (2009) beschreiben Wege wie die Anwendung evolutionärer Algorithmen zur Verfeinerung der Performancemodelle. Alle drei Arbeiten sind im Kontext von PCM angesiedelt.

3.3.3 Zusammenfassung

Für die Performance von Softwaresystemen existieren einige Simulationsansätze. Die meisten dieser Ansätze konzentrieren sich auf die Performance-Betrachtung von HPC-Systemen, nur wenige sind für die Anwendung auf Unternehmensanwendungen ausgelegt.

Bei einer weiterführenden Betrachtung der vorgestellten Simulations-Frameworks erweisen sich viele von ihnen als „one-time-shot“. So tauchen beispielsweise COMPASS und EMPS in keinen weiteren Publikationen auf. Auch lässt sich keine Website aufspüren, die aktuelle Informationen und Versionsstände zu diesen Frameworks anbietet. Zum Performance Prophet existiert eine Website. Die letzten Einträge auf dieser stammen jedoch aus dem Jahr 2005.

Für diese Arbeit sind nur die Simulationsansätze für Unternehmensanwendungen von Interesse. Die beiden im vorherigen Abschnitt vorgestellten Ansätze sind beide aktuell, werden weiter entwickelt und haben in Projekten ihre Anwendbarkeit bewiesen. Für diese Arbeit fällt die Entscheidung auf PCM, aus dem Grund, dass für PCM Zugang zum Quellcode möglich ist. Für das in dieser Arbeit durchgeführte kontrollierte Softwareexperiment ist eine Anpassung am Simulations-Framework (die Integration der Antwortzeitmodelle) notwendig. In einem geschlossenen System wie LQSIM ist dies nicht möglich.

3.4 Modellbildung von Komponentenverhalten auf Messdaten

In diesem Teilkapitel sind zwei Themenbereiche zusammengefasst: die Modellbildung von Komponentenverhalten basierend auf Messdaten und die Modellbildung mittels eines evolutionären Algorithmus. Zu beiden Themenbereichen existiert nur wenig Literatur, und semantisch gehören sie im Rahmen dieser Arbeit zusammen, so dass in diesem Teilkapitel eine vereinte Betrachtung erfolgt. Im ersten Abschnitt erfolgt die Analyse bestehender Literatur zur Modellbildung von Komponentenverhalten, gefolgt von der Literaturanalyse zu evolutionären Algorithmen zur Modellierung.

3.4.1 Modellierung von Komponentenverhalten

Der bereits vorgestellte EMPS-Ansatz von Hollingsworth et al. (2005) verwendet den MetaSim Convolver (Snavely et al. 2002) zur Erzeugung von Performance-Modellen für HEC-Anwendungskomponenten. Der MetaSim Convolver extrahiert das Speichernutzungs-verhalten der Komponenten der betrachteten HEC-Anwendung aus Trace-Dateien und kombiniert sie mit Maschinenprofilen der Maschinen, auf welchen die Anwendung ausgeführt

werden soll. Mittels statistischer Simulation ermöglicht der EMPS-Ansatz hierdurch eine Performance-Vorhersage für die betrachtete Anwendung.

Der vorab bereits vorgestellte Ansatz *Performance Prophet* von (Pllana et al. 2008) bildet das Performance-Verhalten von Komponenten der betrachteten HPC-Anwendungen in Form mathematischer Modelle ab. Die Komponentenmodellerstellung erfolgt mithilfe mathematischer Regression auf Messdaten. Die Autoren betonen die Losgelöstheit der mathematischen Modelle von allen strukturellen Informationen wie der Reihenfolge der Befehlsausführung oder des Kontrollflusses, weshalb eine Kombination mit einem Simulationsansatz notwendig sei. Bei Pllana et al. bilden die Komponentenmodelle einzig das Performance-Verhalten der einzelnen Komponenten ab, die Betrachtung des Kontrollflusses erfolgt mithilfe der vorab vorgestellten Discrete-Event-Simulation. (Schwetman 1978) bezeichnet dies als hybrides Simulationsmodell. Am Beispiel einer wissenschaftlichen Anwendung mit 15.000 Programmzeilen zeigen die Autoren die Effizienz dieser Aufteilung.

Kraft et al. (2009) modellieren die Ressourcennutzung von Services auf Basis von Antwortzeitmessungen. Der Verzicht auf Serverinstrumentierung und Sampling ermöglicht eine einfache Anwendung der Methodik selbst wenn der Systemzugriff eingeschränkt ist. Die Autoren fokussieren sich auf die Analyse von ERP-Systemen und erklären im Detail die Anwendung linearer Regression und die *maximum likelihood estimation* (Bolch et al. 2006). In der Publikation wird erörtert, dass die Ressourcennutzung eines Services ein wichtiger Eingabewert für die Performance-Simulation ist. In ihrer Arbeit geben Kraft et al. eine akkurate Beschreibung der Anwendung der vorgestellten Methoden und validieren ihren Ansatz an einem Realwelt-ERP-System.

3.4.2 Evolutionäre Algorithmen zur Modellbildung

Ein sehr technischer und auf das High Performance Computing (HPC) orientierter Ansatz der Performance-Modellierung wird von Tikir et al. (2007) vorgestellt. Die Autoren präsentieren in ihrer Publikation einen Ansatz zur Performance-Vorhersage basierend auf einem genetischen Algorithmus. In diesem genetischen Algorithmus wird die Performance-Vorhersage als multidimensionales, globales Suchproblem abgebildet. Tikir et al. validieren ihren Ansatz an verschiedenen HPC-Anwendungen und kommen zu dem Ergebnis, dass eine Performance-Vorhersage auf diesem Weg möglich ist.

Trotz umfangreicher Recherche bildet die Publikation von Tikir et al. neben den eigenen Arbeiten (Tertilt et al. 2012; Tertilt/Krcmar 2012; Tertilt et al. 2010) die einzige Publikation im Feld der Komponentenmodellierung mittels evolutionärer Algorithmen.

3.4.3 Zusammenfassung

Während die Modellierung und Simulation des Performance-Verhaltens von Anwendungssystemen im Ganzen in zahlreichen Publikationen thematisiert wird findet die Modellbildung der in der Simulation verwendeten Komponenten in der Literatur kaum Beachtung. Selbst die Dokumentationen der Simulations-Frameworks liefern zu diesem Thema nur unzureichend Informationen – insbesondere, wenn die Modellierung nicht auf Ressourcennutzungsinformationen, sondern auf dem Antwortzeitverhalten erfolgt. Die wenigen hierzu gefundenen Publikationen konzentrieren sich, ähnlich wie bei den Modellierungs- und Simulationsframeworks, hauptsächlich auf den Bereich wissenschaftlicher Anwendungen.

Die Suche nach Literatur zur Verwendung evolutionärer Ansätze zur Modellierung des Performance-Verhaltens letztendlich resultiert in genau einem Ergebnis. Die Arbeit von Tikir et al. zeigt die Anwendbarkeit genetischer Algorithmen zur Performance-Modellierung für den HPC-Bereich. Eine ähnliche Arbeit für den Bereich der Unternehmensanwendungen existiert nicht. Diese Lücke wird durch die vorliegende Arbeit geschlossen.

3.5 Zusammenfassung

Kapitel 3 stellt den in der Literatur beschriebenen aktuellen Forschungskontext dieser Arbeit dar. Hierzu werden die relevanten Arbeiten zu den drei *Bereichen Performance-Modellierung für Softwaresysteme*, *Performance-Simulation* sowie *Modellbildung von Komponentenverhalten auf Messdaten* vorgestellt. Aus den vorgestellten Arbeiten können folgende Schlüsse gezogen werden:

- Die Performance-Analyse von Softwaresystemen erfährt seit Jahren eine kontinuierliche wissenschaftliche Betrachtung von einer kleinen aber etablierten Community.

- Der Fokus der bisherigen Arbeit konzentriert sich primär auf wissenschaftliche Anwendungen aus dem Bereich des HPC und HEC.

- In Bezug auf die Modellierung von Komponentenverhalten auf Messdaten existieren nur wenige wissenschaftliche Publikationen. Insbesondere der Einsatz evolutionärer

Algorithmen findet nur in einer Veröffentlichung Betrachtung, und dort auch nur zur Abbildung der Performance des Gesamtsystems.

- Von den wenigen beschriebenen Performance-Analyse-Frameworks für Unternehmensanwendungen befinden sich kaum welche in einem kontinuierlichen Weiterentwicklungsprozess und im Einsatz.

Für die Beantwortung der ersten Forschungsfrage ergibt sich insbesondere durch den letzten Punkt die Frage, ob es überhaupt einen Simulationsansatz gibt, der sich für die Performance-Analyse von Unternehmensanwendungen auf Basis von Antwortzeit-Komponentenmodellen eignet. Eine intensive Untersuchung ergab die Eignung zweier Ansätze: LQSIM von (Woodside 2002) und (Franks et al. 2005) sowie PCM von (Reussner et al. 2007) und Becker et al. (1996; 2001; 2004). PCM besitzt eine deutlich aktivere Community und befindet sich in einem Prozess kontinuierlicher Weiterentwicklung. Zudem besteht für PCM der für die Umsetzung der Einbindung von Antwortzeit-Komponentenmodellen notwendige Zugriff auf den Programmcode. Hierdurch ergibt sich die folgende Beantwortung der ersten Forschungsfrage:

Forschungsfrage 1: Welcher in der Literatur beschriebene Simulationsansatz für das Performance-Verhalten von Unternehmensanwendungen eignet sich für die Performance-Analyse auf Basis des Antwortzeitverhaltens der Anwendungskomponenten?

Antwort 1: Der Performance-Simulationsansatz *Palladio Component Model* (PCM) von (Reussner et al. 2007) und (Becker et al., 2009) eignet sich für die Performance-Analyse von Unternehmensanwendungen auf Basis des Antwortzeitverhaltens der Anwendungskomponenten. Der Ansatz ist umfangreich in der Literatur beschrieben und wird kontinuierlich durch eine aktive und etablierte Community weiterentwickelt. PCM besitzt bereits eine erste Schnittstelle zum Einbinden von Antwortzeitkurven und liegt in Form des Quellcodes vor, wodurch die Entwicklung einer Schnittstelle zum Einbinden von Antwortzeit-Komponentenmodellen ermöglicht wird.

Der PCM-Ansatz dient damit als Basis für das in dieser Arbeit beschriebene kontrollierte Softwareexperiment.

Durch die Analyse der vorgestellten Literatur wird die von dieser Arbeit geschlossene Forschungslücke deutlich. Während die betrachteten Performance-Simulationsansätze für Unternehmensanwendungen im Detail erklären, wie das Simulationsmodell des untersuchten

Systems im Ganzen erstellt werden muss findet die Abbildung der Komponentenmodelle primär auf Basis des Ressourcenverbrauchs statt. In der Praxis ist dieser Ressourcenverbrauch jedoch oft nicht gegeben oder zu erlangen. Die Komponentenmodellbildung auf Basis des Antwortzeitverhaltens einer Software-Komponente findet kaum bis keine Betrachtung. Insbesondere bei Systemen, die nur als Black Box vorliegen ist dies jedoch in vielen Fällen das einzige Performance-Maß, das erhoben werden kann.

Der Einsatz evolutionärer Algorithmen zur Modellierung des Komponentenverhaltens ist in der Literatur ebenfalls bislang nicht dokumentiert. Die Arbeit von Tikir et al. (2007) zeigt jedoch die Anwendbarkeit eines solchen Ansatzes im Kontext der Performance-Modellierung. Hierdurch bestärkt sich Annahme 2, dass mittels evolutionärer Algorithmen Antwortzeit-Komponentenmodelle generiert werden können.

4 Integration und Konfiguration des evolutionären Algorithmus

Die im vorherigen Kapitel durchgeführte Literaturanalyse offenbart gerade bei der Performance-Analyse von Unternehmensanwendungen eine deutliche Lücke zwischen der durchgeführten Forschung und der praktischen Anwendung der Performance-Analyse im realen Kontext. Zur Untersuchung der Effektivität der vorgestellten Performance-Analyseframeworks werden im Allgemeinen kleine Beispielanwendungen oder Laborumgebungen bemüht. Die Anwendung der Performance-Analyse im realen Unternehmenskontext (wie im folgenden Kapitel 5.1 vorgestellt) zeigt jedoch, dass in einem solchen realen Kontext zahlreiche der für die Performance-Simulationsmodelle benötigten Informationen (insbesondere der Ressourcenverbrauch einer Komponente) nur teilweise oder gar nicht zu erlangen sind. Oftmals, insbesondere bei verwendeten externen Komponenten oder Services, ist einzig eine Antwortzeitmessung an der Schnittstelle durchführbar.

Das in Kapitel 2.4 vorgestellte Konzept der evolutionären Algorithmen ermöglicht eine Modellgenerierung auf Antwortzeitdaten. Mithilfe eines evolutionären Algorithmus werden aus gemessenen Mengen an Antwortzeitinformationen mathematische Modelle extrahiert, die das Antwortzeitverhalten von Komponenten oder Services abbilden (siehe Kapitel 2.3 zu den Komponentenmodellen). Im vorliegenden Kapitel wird beschrieben, wie der in dieser Arbeit verwendete evolutionäre Algorithmus *Mendel* aufgebaut und implementiert ist. Anschließend wird dargelegt, wie die durch den evolutionären Algorithmus generierten Modelle in die verwendete Performance-Simulationsumgebung PCM integriert werden. Hierzu erfolgt eine Beschreibung der Umsetzung der Formula-Implementierung der Performance Curve Integration. Zudem erfolgt eine Analyse der optimalen Konfiguration des evolutionären Algorithmus für den Anwendungsfall der Antwortzeitmessdaten. Die Konfiguration des Algorithmus ist immens wichtig für eine effiziente Modellbildung, Fehler in der Algorithmenkonfiguration resultieren in fehlerhaften Modellen oder keiner Konvergenz.

4.1 Annahmen

Die im vorliegenden Kapitel durchgeführten Schritte entsprechen der Experimentvorbereitung des kontrollierten Softwareexperiments. Durch die Implementierung des evolutionären Algorithmus, die Integration der Algorithmus-Ergebnisse in die Simulationsumgebung sowie die Ermittlung einer optimalen Konfiguration für den evolutionären Algorithmus erfolgt die Definition des stabilen Experimentkontextes.

Die Definition des stabilen Experimentkontextes entspricht dabei auch der Beantwortung der zweiten Forschungsfrage. Diese lässt sich aufteilen in drei Annahmen.

Annahme 2.1: Es lässt sich ein evolutionärer Algorithmus implementieren, der aus gemessenen Antwortzeitdaten von Softwarekomponenten mathematische Modelle generiert, welche die gemessenen Antwortzeitdaten beschreiben.

Die Anwendung des evolutionären Algorithmus für die Modellierung des Antwortzeitverhaltens von Softwarekomponenten auf Basis von Messdaten ist das Kernthema dieser Arbeit. Die Annahme, dass ein solcher evolutionärer Algorithmus nicht nur theoretisch, sondern auch praktisch umsetzbar ist, ist Grundvoraussetzung für den Erfolg dieser Arbeit. Die Validierung der Annahme erfolgt durch die Umsetzung des evolutionären Algorithmus in Form eines ausführbaren Programms.

Annahme 2.2: Die durch den evolutionären Algorithmus generierten Modelle lassen sich in eine existierende Simulationsumgebung für die Performance von Unternehmensanwendungen einbinden.

Ist Annahme 2.1 bestätigt, so bringen die generierten Modelle jedoch für die Überprüfung der in Kapitel 1.2 getroffenen Grundannahmen wenig, wenn sie nicht in die verwendete Performance-Simulationsumgebung eingebunden werden können. Annahme 2.2 ist eine Grundvoraussetzung für die im nächsten Kapitel 5 durchgeführte Experimentausführung zur Beantwortung der dritten Forschungsfrage. Die Validierung der zweiten Annahme erfolgt, ebenso wie die Überprüfung der ersten Annahme, durch Umsetzung, in Form einer Schnittstellenimplementierung in das ausgewählte Performance-Simulationsframework.

Annahme 2.3: Für den evolutionären Algorithmus existiert eine optimale Konfiguration, und diese kann ermittelt werden.

Die Effizienz eines evolutionären Algorithmus zur Modellierung von Datensätzen ist stark abhängig von seiner Konfiguration. Eine optimale Konfiguration wiederum ist abhängig von der Struktur der zugrundeliegenden Daten. Die Annahme 2.3 impliziert, dass für das Problem der Modellierung von Antwortzeitdaten von Softwarekomponenten eine optimale Konfiguration des evolutionären Algorithmus existiert.

Um diese Annahme zu überprüfen erfolgt ein Vorgehen in zwei Schritten. Im ersten Schritt erfolgen umfangreiche Messungen des Modellierungs- und Konvergenzverhaltens des

evolutionären Algorithmus auf Performance-Messdaten unter kontrolliert veränderten Konfigurationen. Im zweiten Schritt erfolgt die Ermittlung der optimalen Konfiguration mittels eines evolutionären Algorithmus auf Meta-Ebene, der wiederum den evolutionären Algorithmus zur Modellierung als Individuen verwendet. Die Überprüfung der dritten Annahme erfolgt sowohl auf den gemessenen statistischen Daten als auch auf den Ergebnissen des evolutionären Meta-Algorithmus.

4.2 Implementierung des evolutionären Algorithmus

Die Implementierung des verwendeten Prototyps *Mendel* erfolgt in Java. Bei der Umsetzung des Prototyps wird bewusst auf die Verwendung von externen Bibliotheken verzichtet, um volle Einsicht in den Ablauf des Programms und die Möglichkeit zur Anpassung zu erhalten. Die Implementierung von *Mendel* zielt auf maximale Performance, für das Experiment überflüssige und die Messung beeinflussende Elemente wie eine grafische Oberfläche werden bewusst nicht implementiert.

4.2.1 Aufbau

Der Prototyp *Mendel* besteht im Kern aus fünf Klassen. Der Aufbau des Prototyps orientiert sich strikt an den Vorgaben der objektorientierten Programmierung, so dass jede Aufgabe von einer separaten Klasse implementiert wird. Durch die Klassenstruktur sind zudem Ansatzpunkte für die Erweiterung des Prototyps, beispielsweise um eine Implementierung von Genomen flexibler Länge, gegeben. Abbildung 4.1 zeigt das vereinfachte Klassendiagramm des *Mendel*-Core. In der Abbildung wurden alle Klassen weg gelassen, die für das Verständnis des Aufbaus von *Mendel* ohne Belang sind. Dies sind insbesondere Hilfsklassen, die wiederkehrende Funktionalität (wie beispielsweise das Logging) implementieren.

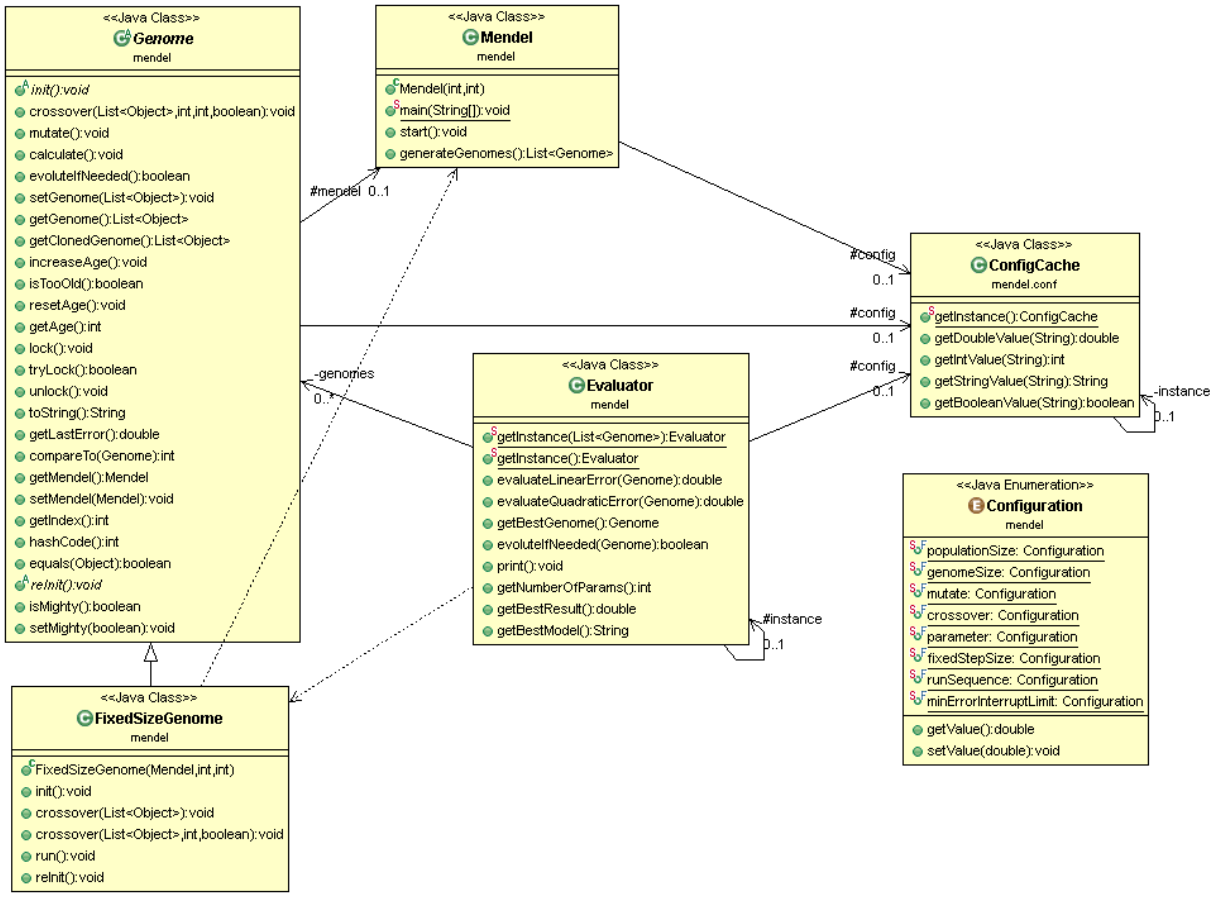


Abbildung 4.1. Klassendiagramm der *Mendel*-Core-Klassen (vereinfacht). Quelle: Eigene Darstellung

4.2.1.1 Klasse *Mendel*

Die Klasse *Mendel* bildet das Startprogramm. Sie initialisiert die Population an Genomen sowie die Evaluator-Instanz. Zur Steigerung der Performance des *Mendel*-Prototyps läuft jedes Genom in einem eigenen Thread. Die Klasse *Mendel* erzeugt und verwaltet diese Threads. Nach dem Ende aller Threads (nach dem Erreichen des Ende-Kriteriums) sorgt die Klasse *Mendel* für ein korrektes Beenden des evolutionären Algorithmus.

4.2.2 Klasse *Genome*

Jedes Objekt der Klasse *Genome* stellt ein Individuum der Population dar. Die Menge aller Objekte der Klasse *Genome* bildet damit die Population des evolutionären Algorithmus (vgl. Abschnitt 2.4.1.1.2). Die Klasse *Genome* implementiert das Runnable-Interface und kann somit als eigenständiger Thread ausgeführt werden. Im Prototyp *Mendel* implementiert die Klasse Genome die Mutations- und Crossover-Operationen. Weiterhin führt jedes *Genome* ein Age-Attribut. Je nach Konfiguration des evolutionären Algorithmus wird ein *Genome* automatisch re-initialisiert, wenn es eine vorgegebene Menge an Generationen überlebt hat. Auf diese Weise wird die Modell-Diversifikation erhöht. Die Klasse *Genome* ist abstrakt und

wird durch eine konkrete Ausprägung implementiert. Die Spezialisierung der konkreten Ausprägung liegt insbesondere in der Abbildung des Modells.

4.2.3 Klasse *FixedSizeGenome*

Die Klasse *FixedSizeGenome* bildet die im Prototyp *Mendel* verwendete konkrete Ausprägung der abstrakten Klasse *Genome*. Die Modellrepräsentation in der Klasse *FixedSitzeGenome* erfolgt in Form einer Liste von Genome-Elementen mit fester Länge. Das durch die Klasse abgebildete Genom wird während der gesamten Laufzeit des evolutionären Algorithmus weder verkürzt noch verlängert. Diese Art der Genome-Repräsentation hat den Vorteil, dass der Crossover-Operator sehr leicht zu implementieren ist (vgl. Abschnitt 2.4.1).

4.2.4 Klasse *ConfigCache*

Ein evolutionärer Algorithmus bringt zahlreiche Konfigurationsparameter mit sich, die sich stark auf das Konvergenzverhalten und die Lösungsfindungseigenschaft des Algorithmus auswirkt (vgl. Teilkapitel 4.4). Die Konfiguration des Prototyps *Mendel* erfolgt über eine Konfigurationsdatei, in der alle benötigten Parameter (wie beispielsweise die Populationsgröße, die Genom-Länge und andere) vor dem Programmstart definiert werden. Da die Konfigurationsparameter in jedem Genom und in jeder Generation gelesen werden, sich zur Laufzeit des Algorithmus jedoch nicht ändern, ist ein Speichern der Parameterwerte in jedem Genom oder gar ein wiederholtes Auslesen der Konfigurationsdatei nicht effizient. Aus diesem Grund bildet die Klasse *ConfigCache* eine zentrale Speicherstelle für die Konfigurationsparameter des evolutionären Algorithmus. Die Klasse ist als Singleton implementiert, es existiert damit exakt eine Instanz. Diese Instanz wird von jedem Genom sowie dem Evaluator verwendet, um Konfigurationsparameter auszulesen. Die Konfigurationsdatei wird auf diese Weise genau einmal gelesen.

4.2.5 Enumeration *Configuration*

Die Enumeration *Configuration* definiert die Konifgurationsparameter, die vom ConfigCache bereitgestellt werden. Die Einträge der Enumeration dienen als Schlüssel um aus dem *ConfigCache* Konfigurationswerte auszulesen.

4.2.6 Evaluator

Der Evaluator dient der Bewertung der Fitness eines Genoms. Die grundlegende Funktionalität des Evaluators ist in Abschnitt 2.4.1.1.5 beschrieben. Abbildung 4.2 stellt den Evaluator sowie die verwendeten Klassen grafisch dar.

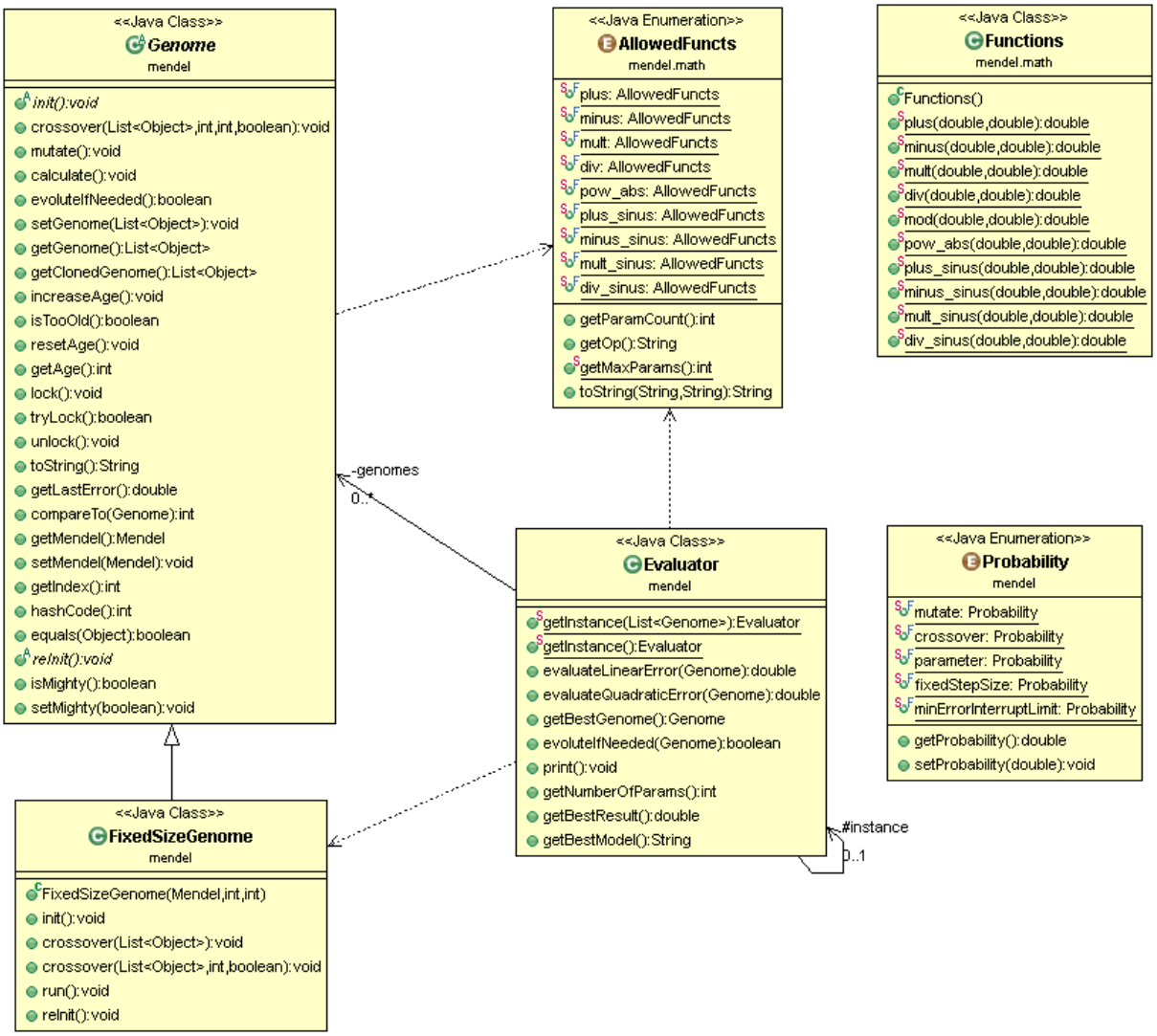


Abbildung 4.2. Klassendiagramm des *Mendel*-Evaluators und der verwendeten Klassen (vereinfacht). Quelle: Eigene Darstellung

4.2.6.1 Klasse *Evaluator*

Die Klasse *Evaluator* führt die Bewertung der Genome einer Generation durch. Sind alle Genome einer neuen Generation erstellt, so berechnet der *Evaluator* für jedes Genom der Population den Fitness-Wert. Anschließend sortiert der *Evaluator* die Liste der Genome anhand des Fitness-Wertes und entscheidet, welche Genome in die nächste Generation übernommen werden. Für alle Genome, die nicht übernommen werden, führt der *Evaluator* den Evolutionsschritt durch: er sucht zwei Eltern, rekombiniert deren Genome mittels Crossover, und führt gegebenenfalls eine Mutation durch. Auf diese Weise erzeugt der *Evaluator* neue Genome für die nächste Generation.

Die Klasse *Evaluator* ist ebenfalls als Singleton implementiert. Jedes Genom operiert somit mit demselben *Evaluator*-Objekt. Dies ermöglicht dem *Evaluator*, eine Bewertung aller

Genome einer Population durchzuführen, ohne dass separat eine Liste der Genome gespeichert werden muss.

4.2.6.2 Enumeration *Probability*

Die Enumeration *Probability* enthält Wahrscheinlichkeitswerte für Ereignisse (wie beispielsweise die Durchführung des Mutations-Operators). Diese Wahrscheinlichkeitswerte nutzt der Evaluator um zu ermitteln, ob ein bestimmtes Ereignis eintritt.

4.2.6.3 Enumeration *AllowedFuncts*

Ziel des Prototyps *Mendel* ist die Erstellung eines mathematischen Modells zur Beschreibung des Performance-Verhaltens einer Softwarekomponente. Die Enumeration *AllowedFuncts* definiert, welche Operatoren in einem solchen mathematischen Modell genutzt werden dürfen. Der Satz der in *Mendel* definierten mathematischen Operationen ist in Abschnitt 2.4.3.1 beschrieben. Die Enumeration *AllowedFuncts* enthält sowohl die Bezeichnung der Operationen, als auch die textuelle Darstellung für die Ausgabe, jedoch nicht die Implementierung der Operationen. Die Enumeration *AllowedFuncts* dient ausschließlich der Referenzierung auf eine der Operationen, beispielsweise in den Modellen – für die Implementierung wird die nachfolgend beschriebene Klasse *Functions* herangezogen.

4.2.6.4 Klasse *Functions*

Die Klasse *Functions* enthält die Implementierungen der in der Enumeration *AllowedFuncts* definierten mathematischen Operatoren. Die zu einem mathematischen Operator passende Methode wird vom Evaluator zur Laufzeit geladen, so dass die Operationsimplementierungen ohne großen Aufwand ausgetauscht oder erweitert werden können.

4.2.7 Zusammenfassung

Der Prototyp *Mendel* ist die Implementierung eines evolutionären Algorithmus zur mathematischen Modellierung des Performance-Verhaltens von Softwarekomponenten großer Unternehmensanwendungen. Die Umsetzung von *Mendel* erfolgt in Java, mit dem Fokus auf der Leistungsfähigkeit des Algorithmus. *Mendel* besitzt eine Multithreading-Architektur, die eine Verteilung des Algorithmus auf nahezu beliebig viele Prozessoren erlaubt. Diese Architektur trägt maßgeblich zur Leistungsfähigkeit des Algorithmus zur Modellierung auch großer Datenmengen bei. Zudem ist der Prototyp vollständig eigenimplementiert, wodurch eine vollständige Einsicht in den Programmablauf gegeben ist.

4.3 Integration des evolutionären Algorithmus in die Simulation

Zur Analyse des Performanceverhaltens von Unternehmensanwendungen findet in dieser Arbeit, wie in Teilkapitel 2.2 beschrieben, die Simulation Anwendung. Als Simulationsansatz wurde das Palladio Component Model (PCM) von Becker et al. (2009) ausgewählt. PCM eignet sich aufgrund der Abbildungsmöglichkeiten komponentenbasierter Systeme optimal für den in dieser Arbeit betrachteten Anwendungsfall (siehe Teilkapitel 5.1). Weiterhin befindet sich PCM in einem kontinuierlichen Weiterentwicklungsprozess durch eine etablierte Community, so dass die Nachhaltigkeit der vorgestellten Lösung gegeben ist.

Zur Integration der durch den evolutionären Algorithmus generierten Modelle in PCM wird der Performance Curve Integration-Ansatz von Wert (2011) weiter entwickelt. Im Folgenden wird vorgestellt, wie die Modelle der evolutionären Algorithmen in PCM integriert werden.

4.3.1 Integration von Performancekennlinien

Die Integration von Performancekennlinien (engl. Performance Curve Integration: PCI) von Wert (2011) ermöglicht die Einbindung von externen Modellen in die PCM-Simulation. Wert bindet die Performance Curves als externes System in das PCM-Simulationsmodell ein und nutzt Quality of Service Annotationen (*QoSAnnotations*), um die Verknüpfung zwischen einer Komponente und der zur Komponenten gehörenden Performancekennlinie abzubilden. Durch die PCI wird erstmalig die Möglichkeit gegeben, Antwortzeitmodelle in die PCM-Simulation einzubinden, die über einfache Verteilungsfunktionen hinausgehen. Im Folgenden wird erörtert, inwieweit die PCI für diese Arbeit nutzbar ist, und welche Anpassungen notwendig sind, um die Beantwortung der Forschungsfragen 2 und 3 zu unterstützen.

4.3.1.1 Einbindung der Performancekennlinien in PCM

Die Performancekennlinien werden in PCM als *PerformanceCurveQoSAnnotation* eingebunden, eine Spezialisierung der *QoSAnnotations*. *QoSAnnotations* dienen in PCM als Verweis auf die Spezifikation von nichtfunktionalen Anforderungen eines externen Systems, wie beispielsweise dessen Verfügbarkeit oder Antwortzeitverhalten. Sie ist verbunden mit exakt einer *RequiredRole* und einem *Interface* (siehe (Becker et al. 2009) für das PCM Metamodell) und verweist damit auf die Eigenschaften exakt eines verwendeten externen Systems. Die *Signature* und ihre Verbindung über die *SignatureReference* zu exakt einer *Specification* (im Fall der Performancekennlinien die *PerformanceCurveSpecification*) definiert die tatsächlichen nichtfunktionalen Eigenschaften des externen Systems. Abbildung 4.3 stellt die Abhängigkeiten in Form eines PCM-Metamodell-Ausschnitts grafisch dar.

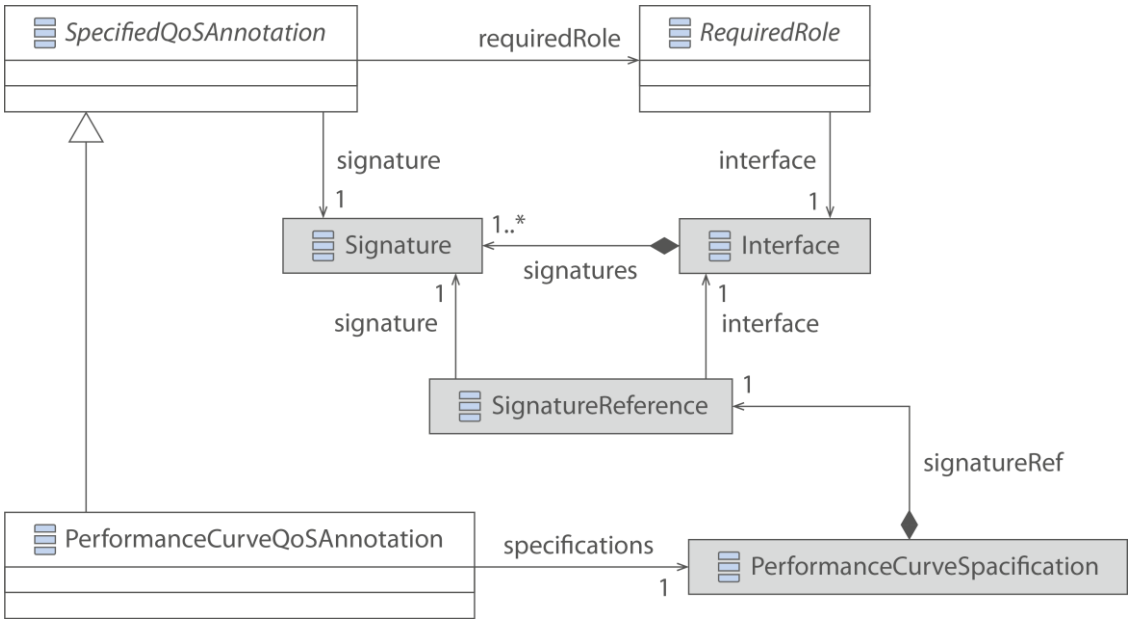


Abbildung 4.3: Performancekennlinien als externes System. Quelle: Eigene Darstellung, nach (Wert 2011).

Die *PerformanceCurveQoSAnnotation* dient damit zur Einbindung exakt einer Performancekennlinie. Die Performancekennlinie selber ist über die *PerformanceCurveSpecification* definiert, welche wiederum eine konkrete *PerformanceCurve* referenziert. *PerformanceCurve* ist ein abstrakter Datentyp, implementiert von Wert (2011) zum aktuellen Zeitpunkt im Subtyp *DataTable* (Abbildung 4.4).

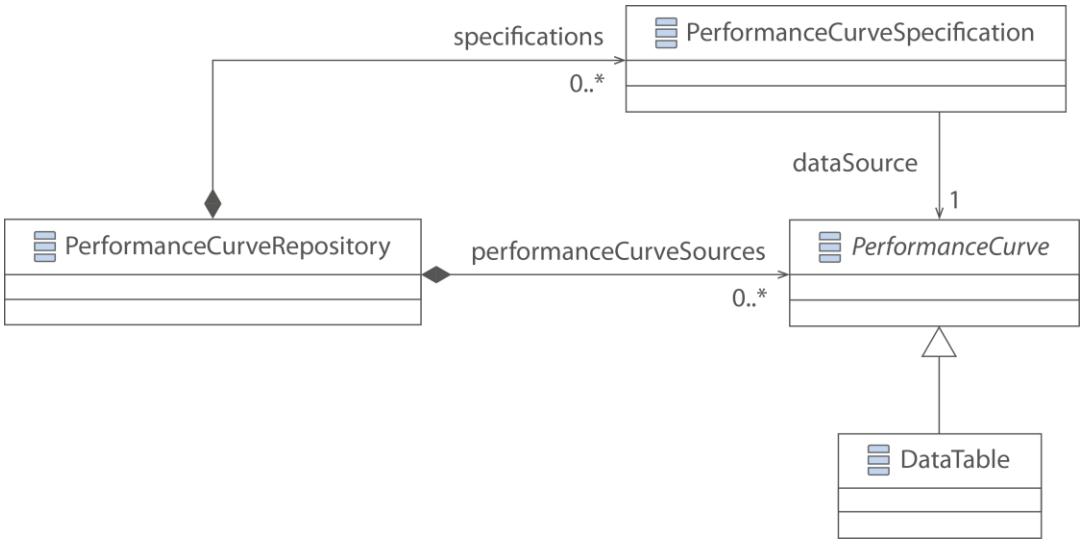


Abbildung 4.4: Metamodell der Performancekennlinien, Implementierungsstand zu Beginn dieser Arbeit. Quelle: Eigene Darstellung, nach (Wert 2011).

Das Performancekennlinien-Metamodell sieht vor, dass eine Performancekennlinie für verschiedene Interfaces verwendet werden kann. Performancekennlinien sind statusfrei, so dass eine Mehrfachverwendung ohne weiteres möglich ist. Um unnötige Doppelallokationen der teilweise recht umfangreichen Performancekennlinien zu vermeiden werden einmal allokierte Performancekennlinien im *PerformanceCurveRepository* abgelegt. Aus dem Repository heraus können sie für die Antwortzeitberechnung anderer Komponenten heran gezogen werden.

4.3.1.2 *DataTable*-Implementierung von *PerformanceCurve*

Bei der *DataTable*-Implementierung von *PerformanceCurve* liegt die Performancekennlinie in Form einer CSV-Datei vor. Für jeden vorhandenen Messwert enthält die CSV-Datei einen Eintrag in Form einer Zeile, die sich aus den, mit Semikolon getrennten, Eingabewerten sowie der resultierenden Antwortzeit zusammensetzt.

Aus den in der CSV enthaltenen Messwert-Tupeln werden während des Simulationslaufs die Einträge ausgewählt, die entweder exakt auf die in der Simulation vorliegenden Input-Faktoren passen oder diese direkt umgeben. Existieren keine exakt passenden Einträge, so erfolgt eine lineare Interpolation zur Berechnung der tatsächlich zu erwartenden Antwortzeit. Die vorliegende *DataTable*-Implementierung verlangt hierzu ein gleichmäßiges Gitter an Messwerten mit äquidistanten Schritten auf allen Input-Dimensionen.

4.3.1.3 Nachteile der *DataTable*-Implementierung

Für den in dieser Arbeit betrachteten Anwendungsfall bringt die *DataTable*-Implementierung einige Nachteile mit sich. Diese manifestieren sich insbesondere in der Notwendigkeit äquidistanter Schritte auf allen Eingabe-Dimensionen der Messwerte sowie in der Tatsache, dass für jeden simulierten Operationsaufruf die gesamte DataTable, also alle in der CSV-Datei enthaltenen Messwerte, durchsucht werden müssen.

Die Projekterfahrung zeigt, dass es für produktiv eingesetzte Unternehmensanwendungen wie dem in dieser Arbeit betrachteten Anwendungsfall schwer oder sogar unmöglich ist, die geforderten äquidistanten Messwerte zu erlangen. Das Antwortzeitverhalten einer Komponente ist stark abhängig von der zugrunde liegenden Hardware, so dass es bei der Performance-Analyse auf Antwortzeitbasis zwingend notwendig ist, die Messungen auf der produktiven Hardwareumgebung oder zumindest einer produktivnahen Umgebung durchzuführen. Wird die Anwendung, wie im unter 5.1 beschriebenen Anwendungsfall, als Metrocluster über zwei Rechenzentren verteilt betrieben, so steht eine Kopie dieser Umgebung zu Messzwecken meist nicht zu Verfügung.

Im vorliegenden Anwendungsfall erfolgt die Messwerterhebung über Logfile-Analysen sowie justierte Last- und Performance-Messwerte (siehe Teilkapitel 5.2. für Details zur Messung). Die hierdurch erhobenen Messwerte ergeben sich zu einem großen Teil aus dem Zugriffsverhalten realer Anwender auf das Produktivsystem. Eine Äquidistanz in den Input-Dimensionen der Messwerte ist damit im Allgemeinen nicht gegeben.

Weiterhin stellt die Suchstrategie der *DataTable*-Implementierung ein beträchtliches Performance-Problem für die Simulation selber dar. Für jede Anfrage an die Performancekennlinie erfolgt ein linearer Suchlauf durch alle Einträge. Dies führt insbesondere bei Komponenten mit vielen kurzläufigen Operationsaufrufen zu erheblichen Verzögerungen in der Simulation. In Teilkapitel 4.3.4 wird diese Laufzeitproblematik in der Simulation verdeutlicht.

Zur Lösung der in diesem Abschnitt vorgestellten Problematik der *DataTable*-Implementierung für den betrachteten Anwendungsfall erfolgt im Rahmen dieser Arbeit die *Formula*-Implementierung von *PerformanceCurve*.

4.3.2 *Formula*-Implementierung von *PerformanceCurve*

Die *Formula*-Implementierung von *PerformanceCurve* erlaubt eine mathematische Formel als Definition des Antwortzeitverhaltens einer Operation. Im Gegensatz zur *DataTable*-Implementierung operiert die *Formula*-Implementierung damit nicht auf einem Datensatz unterschiedlicher Größe, sondern auf exakt einer mathematischen Formel pro Operation. Diese Formel besitzt alle das Antwortzeitverhalten der Operation beeinflussenden Variablen als Eingabeparameter und liefert als Ergebnis das zu erwartende Antwortzeitverhalten, welches in der Simulation als *Delay* implementiert wird. Die mathematische Formel bedarf zu ihrer Erstellung kein gleichmäßiges Gitter an Messwerten, wie in Teilkapitel 4.2 gezeigt wird. Abbildung 4.5 zeigt das um die *Formula*-Implementierung erweiterte Metamodell der PCI.

Die Umsetzung der *Formula*-Implementierung erfolgt über die drei Klassen *FormulaPCSource*, *FormulaPCSourceConfiguration* und *FormulaPCBuilder* und fügt sich nahtlos in die bestehende PCI-Paketstruktur ein. Abbildung 4.6 zeigt das Klassendiagramm der Formula-Klassen sowie ihre Einbindung in die bestehende PCI-Struktur.

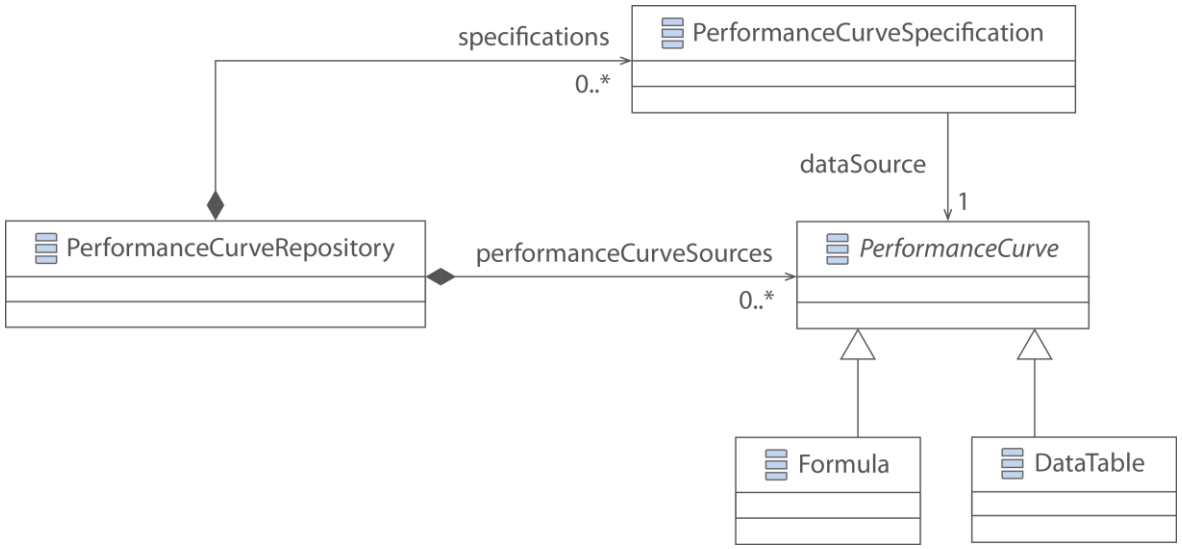


Abbildung 4.5: Metamodell der Performancekennlinien mit *Formula*-Implementierung. Quelle: Eigene Darstellung, nach (Wert 2011).

4.3.2.1 FormulaPCSource

Die Klasse *FormulaPCSource* stellt die eigentliche Implementierung der Performancekennlinie als Formel dar. *FormulaPCSource* erbt von der abstrakten Klasse *PCSource* und implementiert insbesondere die Methode *getResponseTime*. *getResponseTime* nimmt als Parameter die aktuellen Simulationsparameter entgegen und liefert als Ergebnis die zu erwartende Antwortzeit. In *FormulaPCSource* findet damit insbesondere die Auswertung der Formel unter den gegebenen Variablen statt.

4.3.2.2 FormulaPCSourceConfiguration

Als Subklasse von *PCSourceConfiguration* enthält *FormulaPCSourceConfiguration* alle Konfigurationsparameter, die für die Erstellung einer *FormulaPCSource* benötigt werden. *FormulaPCSourceConfiguration* wird von der Klasse *FormulaPCBuilder* genutzt, um eine neue *FormulaPCSource* zu erzeugen. In *FormulaPCSourceConfiguration* wird insbesondere die zu verwendende mathematische Formel als *String* gespeichert.

4.3.2.3 FormulaPCBuilder

FormulaPCBuilder ist eine Subklasse von *AbstractPCBuilder* und implementiert das Factory-Pattern (Gamma 1995a) zur Erzeugung von *FormulaPCSource*-Instanzen. *FormulaPCBuilder* nutzt für die Erstellung einer *FormulaPCSource*-Instanz eine *FormulaPCSourceConfiguration*, in welcher die für die *FormulaPCSource*-Instanz notwendigen Informationen wie beispielsweise die zu verwendende Formel hinterlegt sind. Nach der Erstellung registriert der *FormulaPCBuilder* die *FormulaPCSource*-Instanz in der *PerformanceCurveRegistry* und instanziiert einen *PerformanceCurveInterpreter*, welcher die

Einbindung der *FormulaPCSource* in die Simulation in Form eines Delays vornimmt (siehe (Wert 2011)).

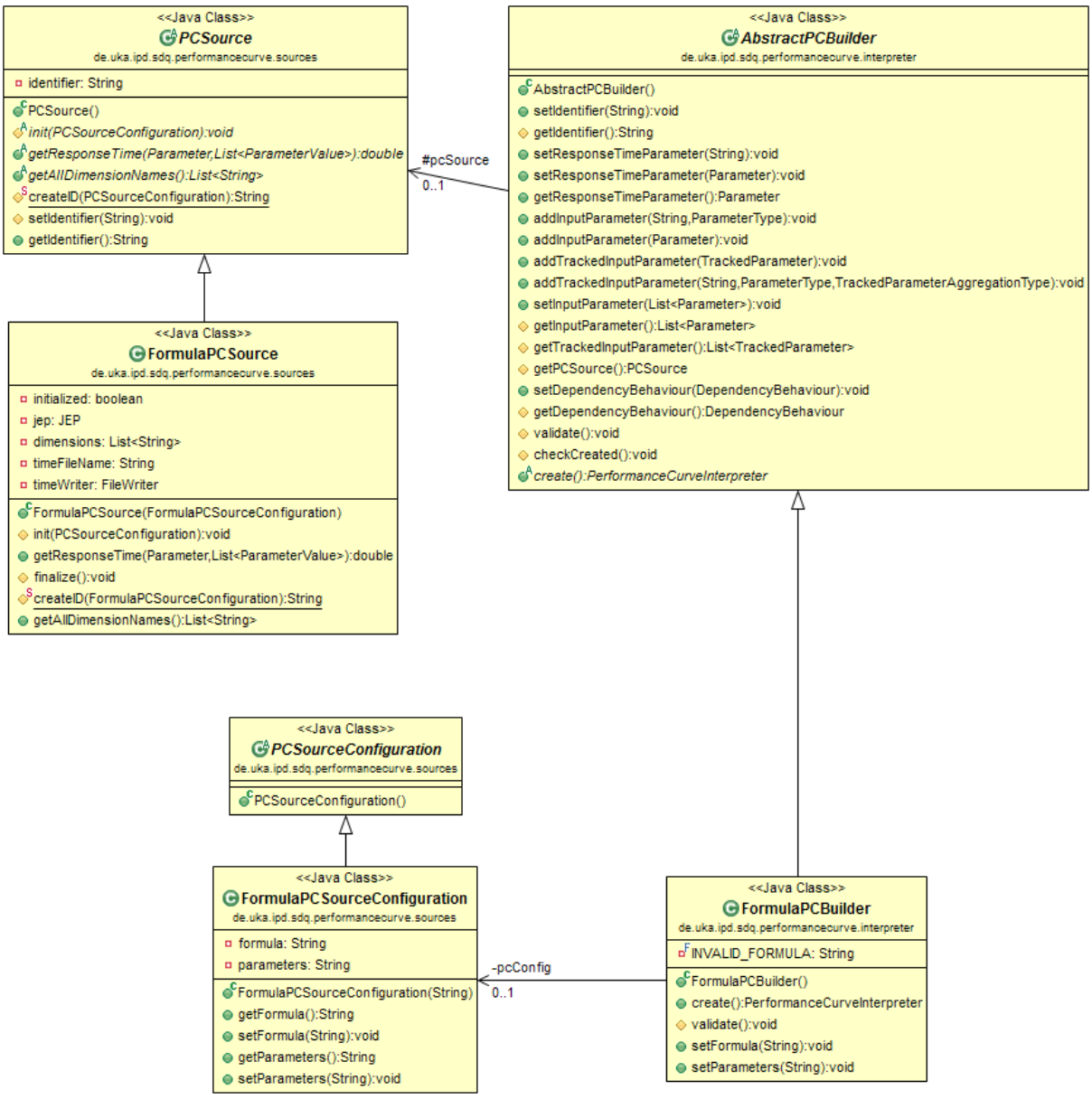


Abbildung 4.6. Klassendiagramm der PCI Formula-Implementierung. Quelle: Eigene Darstellung.

4.3.2.4 Struktur und Auswertung der mathematischen Formel

Über den Modelleditor von PCM hat der Anwender die Möglichkeit, die in der Performancekennlinie zu verwendende Formel im Textformat einzugeben (siehe Abbildung 4.7). Dies geschieht über die Eigenschaft *Formula*. Über die Eigenschaft *Parameters* werden die in der Formel verwendeten Parameter (oder auch Variablen) bekannt gemacht. Dies geschieht zu Zwecken der Validierung und ist durch die abstrakte Klasse *PCSource* vorgeschrieben. Die interne Variable *QUEUELENGTH* enthält die Anzahl der aktuell

durchgeführten parallelen Aufrufe auf der spezifizierten Komponente und muss nicht in *Parameters* deklariert werden.

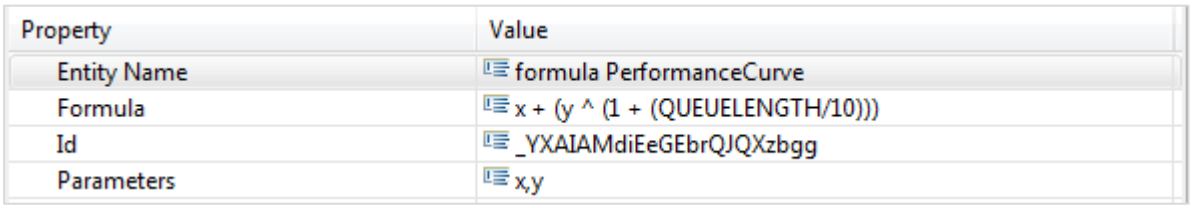


Abbildung 4.7. Eigenschaften der Formula-Implementierung der PCI im PCM-Modelleditor. Quelle: Eigene Darstellung.

Bei jedem simulierten Aufruf der durch die Performancekennlinie repräsentierten Operation wird die Formel ausgewertet. Die Auswertung der textuellen Repräsentation der Formel erfolgt durch die Bibliothek *JEP Java – Math Expression Parser* (SourceForge 2012). JEP unterstützt alle gängigen mathematischen Ausdrücke und speichert die Formel nach dem Parsen in einer optimierten internen Struktur, welche wiederholte Auswertungen unter veränderten Eingabeparametern stark beschleunigt.

Die folgenden Abschnitte zeigen die Berechnungszeiten von JEP für gängige mathematische Funktionen. Das Set an Beispielfunktionen ist entnommen aus (Karaboga/Akay 2006) und repräsentativ für den Anwendungsfall der Performancekennlinienmodellierung. Das Mapping auf die Performancekennlinienmodellierung erfolgt indem die Performancefaktoren als Eingabewerte (wie beispielsweise die Anzahl paralleler Nutzer, die Größe der Übergabeparameter in Byte etc.) als Vektor interpretiert werden.

Zur Untersuchung der Berechnungszeit der mathematischen Funktionen mittels des JEP werden Vektoren mit 500, 1.000, 3.000, 5.000 und 10.000 Dimensionen berechnet. Die einzelnen Dimensionen werden zufällig mit Werten aus dem Intervall [-100; 100] belegt. Für jede der Dimensionsgrößen werden 10.000 Durchläufe gemessen, wobei bei jeder Berechnung die Belegungen durch Zufallswerte neu erzeugt werden.

(Karaboga/Akay 2006) folgend werden die nachfolgenden Funktionen betrachtet.

4.3.2.4.1 Sphere

Die einfachste Test-Funktion ist die Sphere-Funktion, bekannt auch als De Jong-Funktion 1 (De Jong/Spears 1989). Sie ist kontinuierlich, konvex und unimodal: die Funktion hat genau ein globales Optimum und keine lokalen Optima. Formel 1 definiert die Sphere-Funktion.

Formel 1. Die Sphere-Funktion.

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2 \qquad \forall x \in R^n$$

Abbildung 4.8 stellt die Sphere-Funktion $z = f(\vec{x})$ im Intervall [-100, 100] für den zweidimensionalen Eingabevektor $\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix}$ grafisch dar.

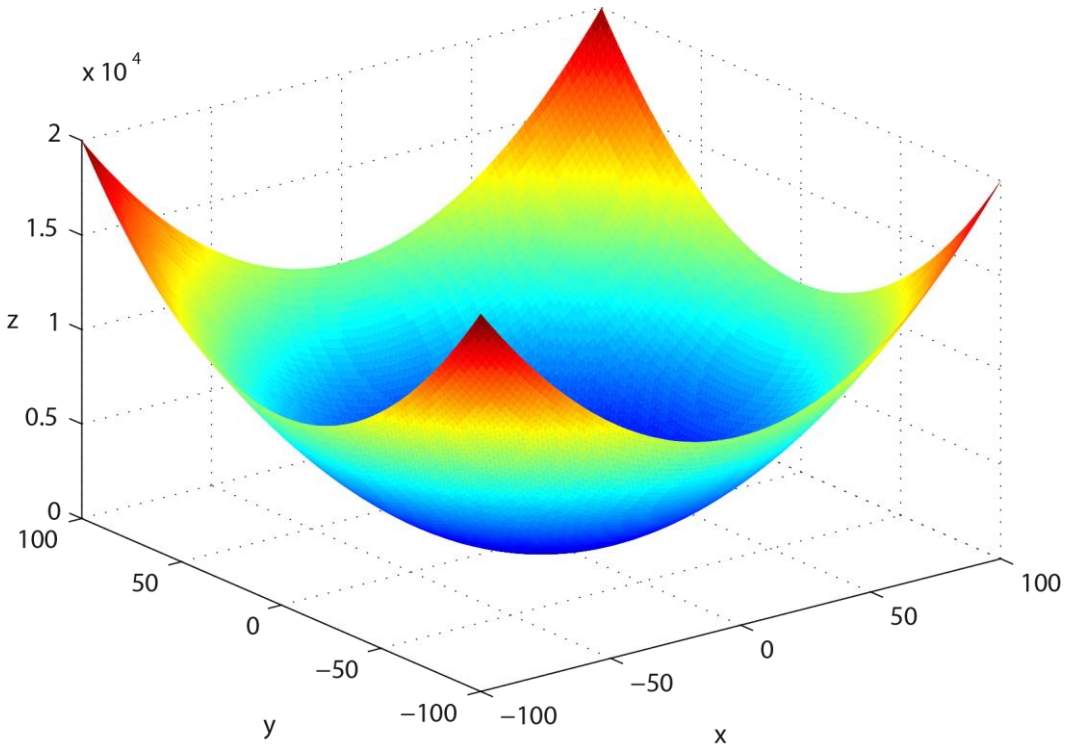


Abbildung 4.8. Grafische Darstellung der Sphere-Funktion im Intervall [-100, 100]. Quelle: Eigene Darstellung.

Zur Evaluierung der Berechnungszeit des JEP wurden die Vektorendimensionen 500, 1.000, 3.000, 5.000 und 10.000 gewählt. Diese Dimensionen liegen deutlich über den zu erwarteten Messwertdimensionen. Die im Folgenden dargestellte Messwertanalyse ist demnach als Worst-Case-Betrachtung zu interpretieren.

Abbildung 4.9 zeigt die gemessenen Mittelwerte und die Streuung der Antwortzeiten nach je 10.000 Berechnungsläufen pro Dimensionierung. Die Darstellung verdeutlicht, dass die Berechnung der Sphere-Funktion bei einer Vektordimension von 500 konstant unter 2 Millisekunden liegt und selbst bei einer Vektorgröße von bis zu 3.000 Elementen noch konstant unter 10 Millisekunden liegt. Erst bei noch höheren Vektorgrößen steigt die Berechnungszeit stark an.

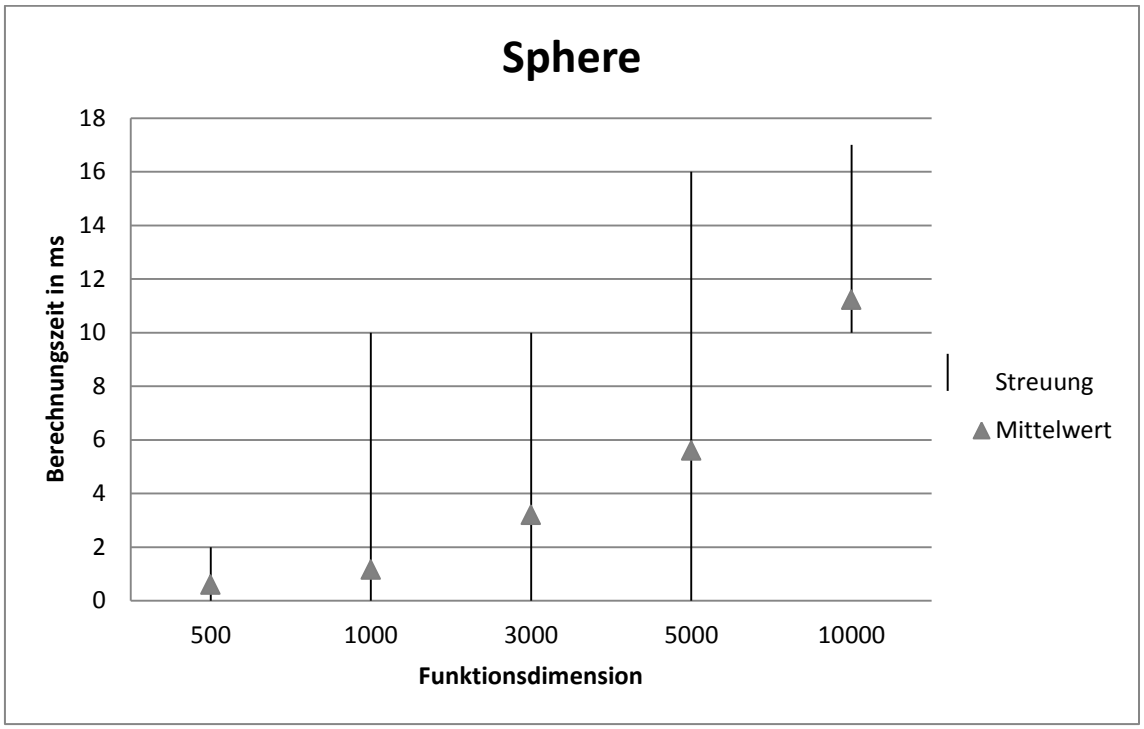


Abbildung 4.9. Darstellung der Rechenzeit-Verteilung des JEP für die Sphere-Funktion. Quelle: Eigene Darstellung.

Bei den erwarteten Messwert-Dimensionen von unter 20 ist von einer Berechnungszeit von weit unter 2 Millisekunden auszugehen.

4.3.2.4.2 Rosenbrock

In der mathematischen Optimierung ist die Rosenbrock-Funktion eine nicht-konvexe Funktion, eingeführt von Howard H. Rosenbrock (1960), um die Performance von Optimierungsalgorithmen zu bewerten. Sie wird auch als Rosenbrock-Tal oder die Rosenbrock'sche-Bananenfunktion bezeichnet. Formel 2 definiert die Rosenbrock-Funktion.

Formel 2. Die Rosenbrock-Funktion.

$$f(\vec{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right] \qquad \forall x \in R^n$$

Das globale Optimum der Funktion befindet sich in einem langen, schmalen, parabolisch geformten flachen Tal. Das Tal zu finden ist trivial. Die Annäherung an das globale Optimum hingegen stellt die eigentliche Herausforderung dar.

Den Vorgaben von Karaboga folgend stellt Abbildung 4.10 die Rosenbrock-Funktion $z = f(\vec{x})$ im Intervall [-30; 30] für den zweidimensionalen Eingabevektor $\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix}$ grafisch dar.

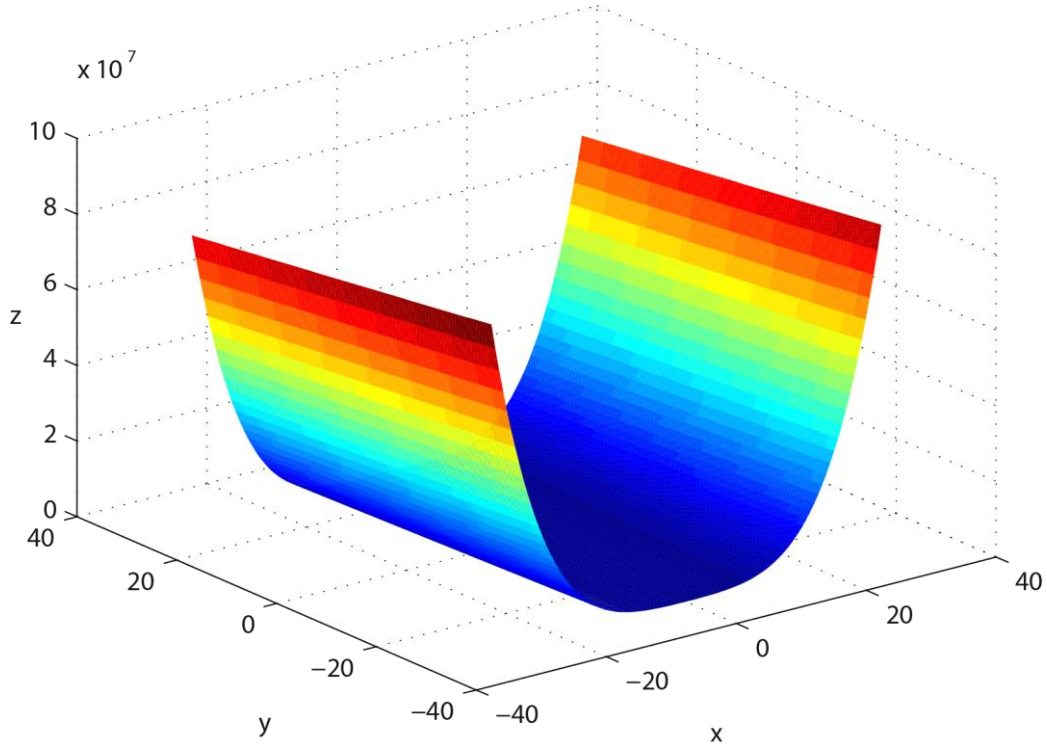


Abbildung 4.10. Grafische Darstellung der Rosenbrock-Funktion auf einem zweidimensionalen Eingabevektor im Intervall [-30; 30]. Quelle: Eigene Darstellung.

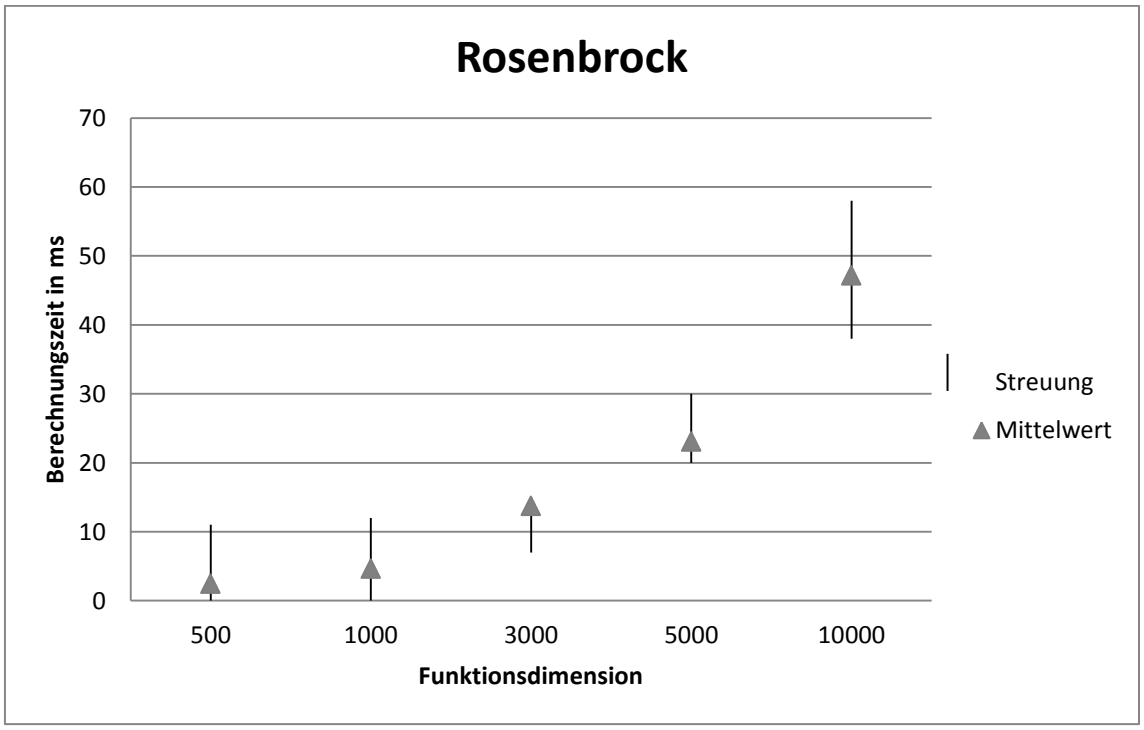


Abbildung 4.11. Darstellung der Rechenzeit-Verteilung des JEP für die Rosenbrock-Funktion. Quelle: Eigene Darstellung.

Die Berechnung der Rosenbrock-Funktion ist aufwändiger als die Berechnung der Sphere-Funktion. Abbildung 4.11 stellt den Mittelwert sowie die Streuung der Berechnungszeit der Rosenbrock-Funktion nach je 10.000 Berechnungsdurchläufen pro Vektorgröße dar.

Es ist ersichtlich, dass der JEP die Rosenbrock-Funktion bei Vektorgrößen bis zu 1.000 innerhalb weniger Millisekunden berechnet, die Streuung bis zu 12 Millisekunden ist für den Einsatz im evolutionären Algorithmus akzeptabel. Im Bereich der höheren Dimensionen ist ein ähnliches Verhalten wie bei der Sphere-Funktion zu beobachten.

4.3.2.4.3 Griewank

Die Griewank-Funktion wird weithin verwendet um das Konvergenzverhalten von Optimierungsfunktionen zu testen. Mit einem Anstieg der Dimensionszahl wächst die Anzahl der durch die Griewank-Funktion dargestellten lokalen Minima exponentiell. Formel 3 stellt die Griewank-Funktion mathematisch dar.

Formel 3. Die Griewank-Funktion.

$$f(\vec{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \qquad x_i \in [-600;600]$$

Andreas Griewank, der Erfinder der nach ihm benannten Funktion, definiert als Gültigkeitsbereich für die Griewank-Funktion das Intervall [-600; 600] (Griewank 1981). In diesem ist die Funktion in Abbildung 4.12 dargestellt.

Auf den ersten Blick wirkt die Griewank-Funktion ähnlich der vorab vorgestellten Sphere-Funktion. Erst ein Zoom wie in Abbildung 4.13 dargestellt auf das Intervall [-60; 60] für beide Eingabedimensionen eröffnet den Blick auf die zahlreichen lokalen Minimal und Maxima der Griewank-Funktion.

Diese zahlreichen, mit jeder Dimension exponentiell zunehmenden lokalen Minima und Maxima bilden die Herausforderung der Griewank-Funktion. Das globale Minimum liegt am Punkt $\vec{x} = \vec{0}$. Jeder Punkt der Funktion (außer $\vec{x} = \vec{0}$) ist jedoch umgeben von zahlreichen lokalen Extrema, von denen immer mindestens die Hälfte besser sind als die aktuelle Position. Diese Struktur macht es einem Optimierungsalgorithmus schwer in Richtung des globalen Optimums vorzustoßen.

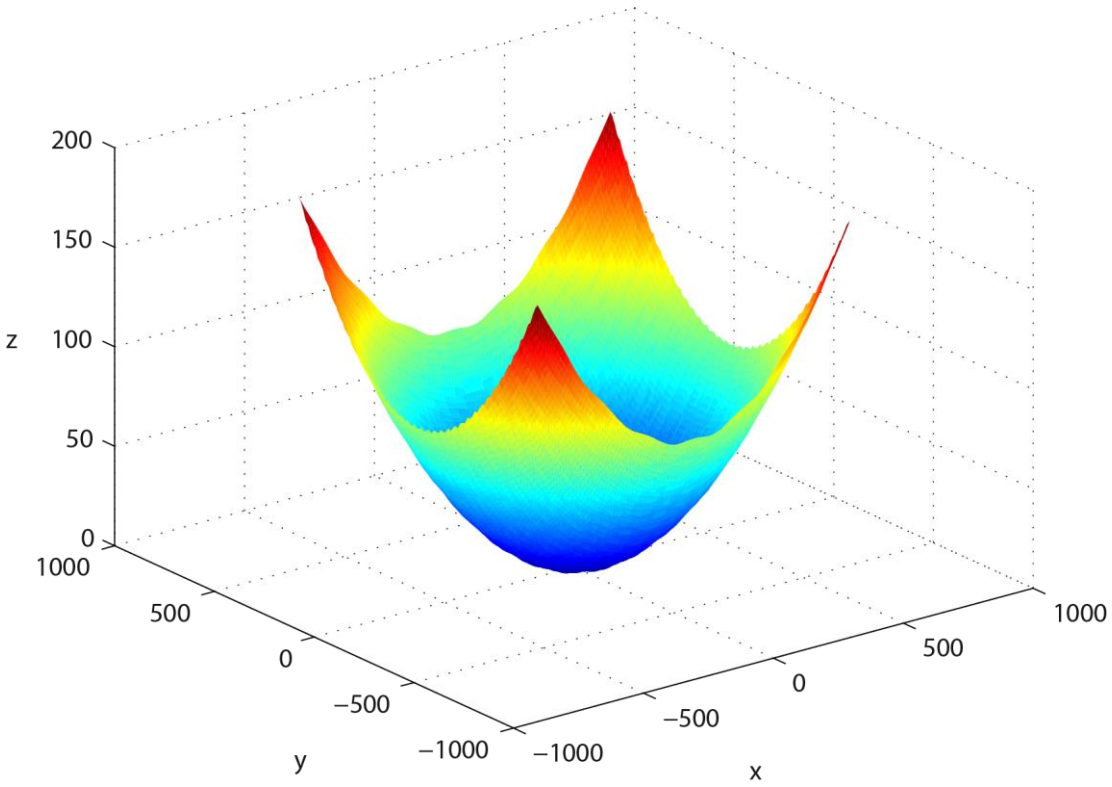


Abbildung 4.12. Grafische Darstellung der Griewank-Funktion auf einem zweidimensionalen Eingabevektor im Intervall [-600; 600]. Quelle: Eigene Darstellung.

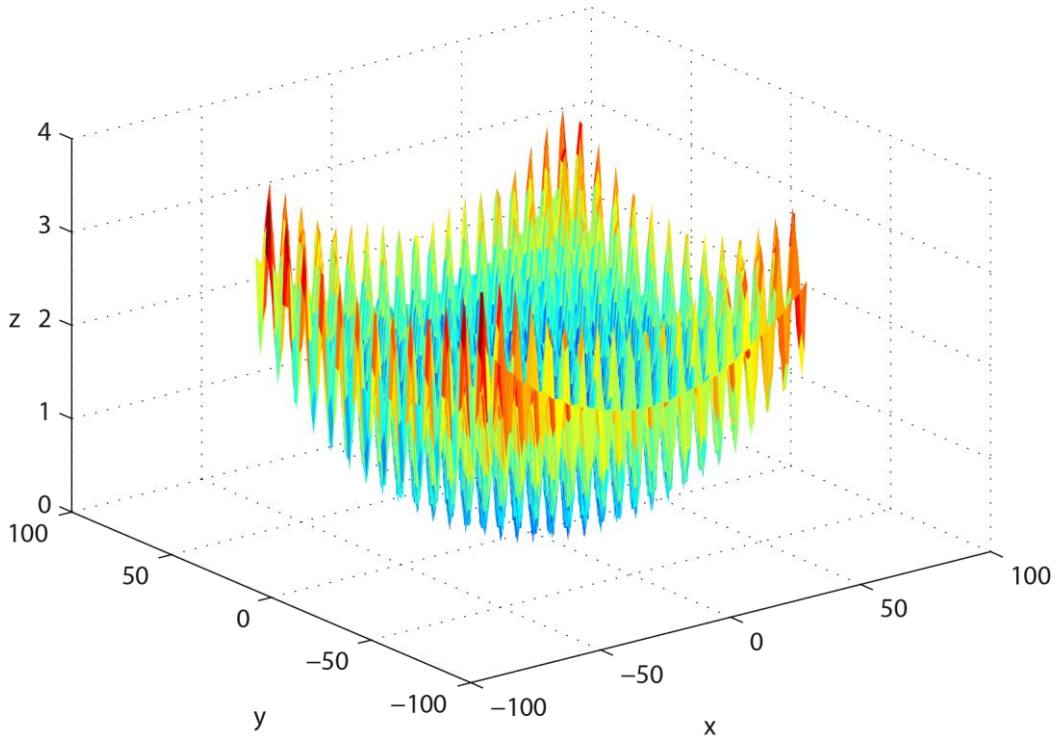


Abbildung 4.13. Zoom in die grafische Darstellung der Griewank-Funktion auf einem zweidimensionalen Eingabevektor. Dargestellt im Intervall [-60; 60]. Quelle: Eigene Darstellung.

Abbildung 4.14 zeigt die Berechnungszeit der Griewank-Funktion. Die Berechnungszeitkurve steigt etwas steiler an als die der Rosenbrock-Funktion, auch die Streuung ist größer. Insgesamt beläuft sich der zeitliche Aufwand für die Griewank-Funktion jedoch ähnlich dem der Rosenbrock-Funktion.

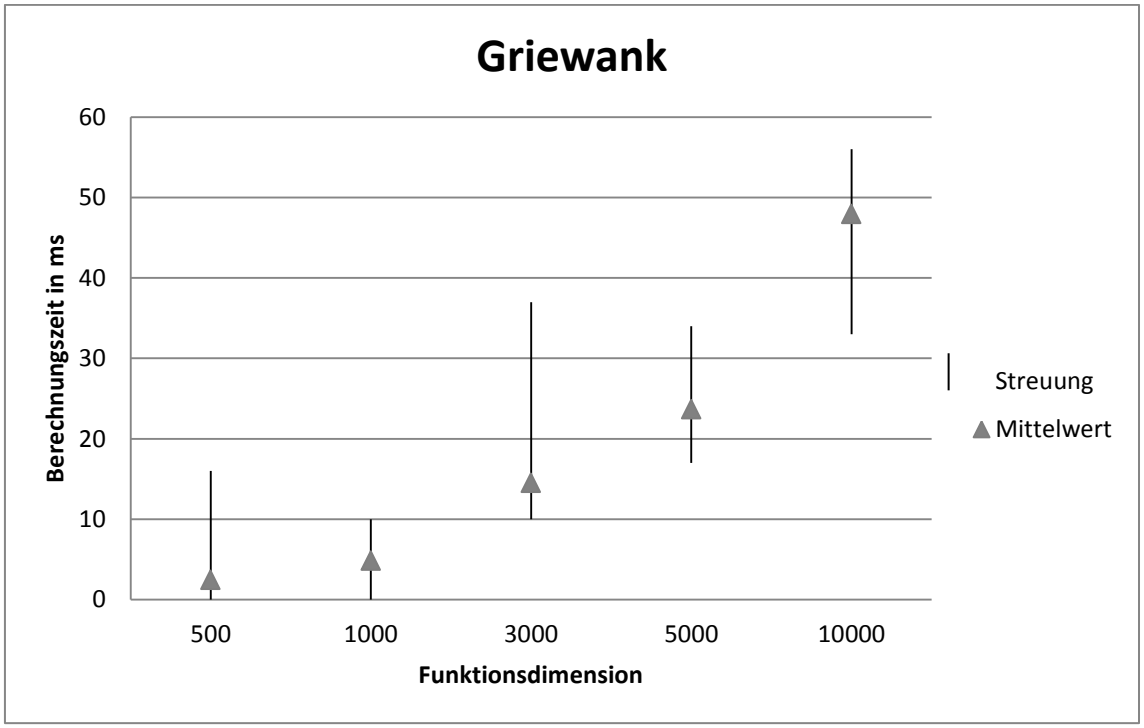


Abbildung 4.14. Darstellung der Rechenzeit-Verteilung des JEP für die Griewank-Funktion. Quelle: Eigene Darstellung.

4.3.2.4.4 Rastrigin

Die Rastrigin-Funktion ist eine nicht-konvexe Funktion aus dem Bereich der mathematischen Optimierung, die zur Performanceanalyse von Optimierungsalgorithmen eingesetzt wird. Die Rastrigin-Funktion ist ein typisches Beispiel einer nichtlinearen multimodalen Funktion. Sie wurde ursprünglich von Rastrigin als 2-dimensionale Funktion vorgeschlagen (vgl. (Törn/Zilinskas 1989)) und anschließend von Mühlenbein verallgemeinert (Mühlenbein et al. 1991). Die Rastrigin-Funktion bildet aufgrund ihres großen Suchraumes und der hohen Anzahl lokaler Minima ein herausforderndes Problem für jeden Optimierungsalgorithmus.

Formel 4 definiert die Rastrigin-Funktion.

Formel 4. Die Rastrigin-Funktion.

$$f(\vec{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right] \qquad x_i \in [-5{,}12; 5{,}12]$$

Abbildung 4.15 stellt die Rastrigin-Funktion in dem von Rastrigin vorgeschlagenen Intervall [-5,12; 5,12] grafisch dar.

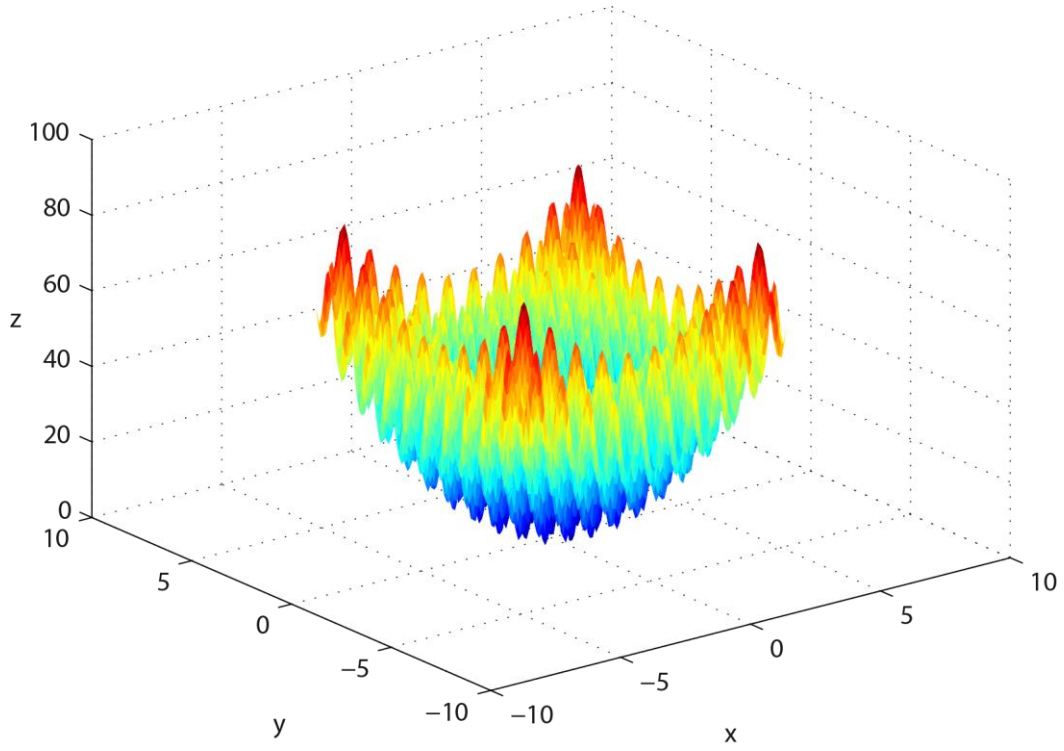


Abbildung 4.15. Grafische Darstellung der Rastrigin-Funktion auf einem zweidimensionalen Eingabevektor im Intervall [-5,12; 5,12]. Quelle: Eigene Darstellung

Es ist aus der Abbildung ersichtlich, dass die Rastrigin-Funktion der Griewank-Funktion ähnelt. Wie die Griewank-Funktion hat auch die Rastrigin-Funktion ihr globales Optimum im Punkt $\vec{x} = \vec{0}$ und umgibt jeden anderen Punkt mit zahlreichen lokalen Extrema. Nicht aus der Grafik ersichtlich ist (aufgrund der Granularität der Darstellungsbereiche), dass die Rastrigin-Funktion innerhalb ihres Definitionsbereichs deutlich mehr lokale Extrema abbildet als die Griewank-Funktion innerhalb dieses Bereiches. Die Rastrigin-Funktion stellt damit um das globale Optimum herum eine höhere Anforderung an den Optimierungsalgorithmus, während die Griewank-Funktion ein (absolut gesehen) deutlich größeres Suchfeld beschreibt.

Die Darstellung der Berechnungszeit der Rastrigin-Funktion in Abbildung 4.16 zeigt erneut die Ähnlichkeit zur Griewank-Funktion. Die Berechnung der Rastrigin-Funktion durch den JEP erfolgt im Schnitt etwas schneller und mit weniger Streuung. Dies ist jedoch primär auf den kleineren Definitionsbereich zurückzuführen.

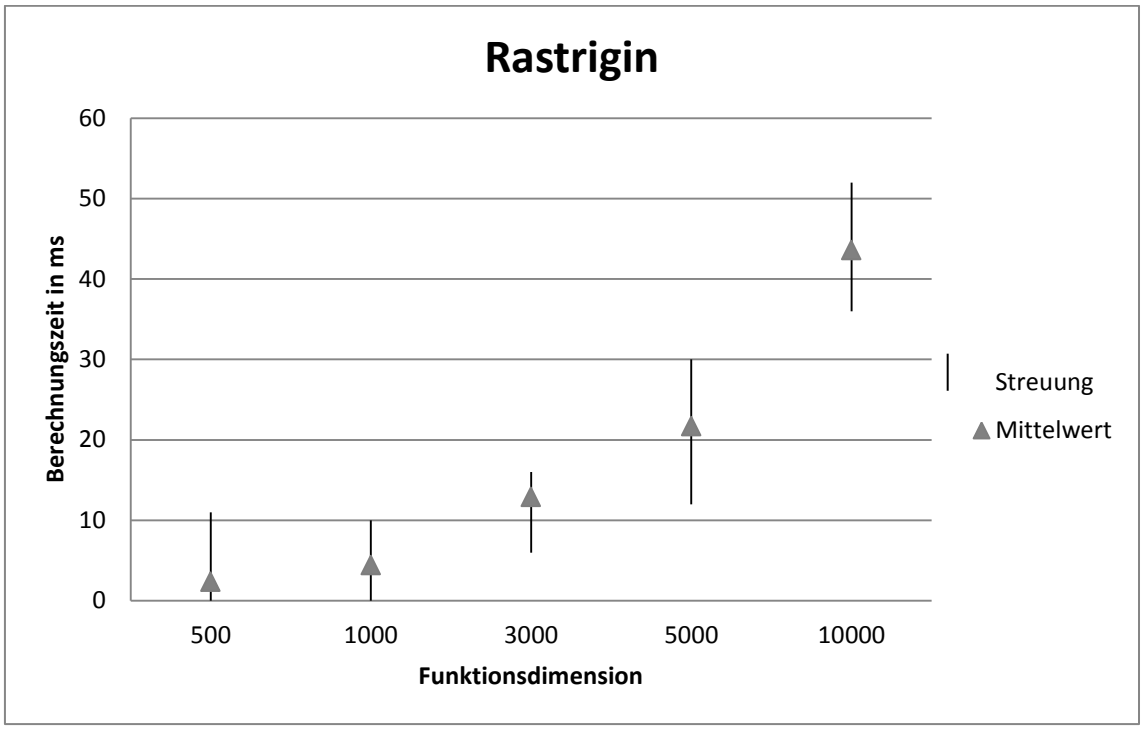


Abbildung 4.16. Darstellung der Rechenzeit-Verteilung des JEP für die Rastrigin-Funktion. Quelle: Eigene Darstellung.

4.3.2.4.5 Ackley

Die Ackley-Funktion ist eine n-dimensionale hochmultimodale Funktion, die eine große Anzahl von lokalen Minima, aber nur ein globales Minimum besitzt. Die Funktion wurde 1987 von Ackley für zwei Dimensionen definiert (Ackley 1987) und 1996 von Bäck verallgemeinert (Bäck 1996). Sie bildet ein typisches Beispielproblem für den Einsatz evolutionärer Algorithmen zur Lösungsfindung (vgl. (Potter/De Jong 1994)).

Die Ackley-Funktion ist in Formel 5 definiert.

Formel 5. Die Ackley-Funktion.

$$f(\vec{x}) = -a \cdot \exp\left(-b \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(c \cdot x_i)\right) + a + e \qquad x_i \in [-32{,}768; 32{,}768]$$

Ackley sowie auch Bäck empfehlen, die allgemeine Formel mit folgenden Parametern zu konkretisieren: $a = 20; b = 0.2; c = 2\pi$. Dieser Empfehlung folgend, und ebenso den Definitionsbereich wie von Ackley und Bäck vorgegeben auf das Intervall [-32; 32] für die beiden dargestellten Dimensionen x und y beschränkend ergibt sich für die Ackley-Funktion eine grafische Darstellung wie in Abbildung 4.17.

Die Ackley-Funktion besteht aus zahlreichen lokalen Extrema, mit einem Trichter um den Punkt $\vec{x} = \vec{0}$ herum. Der Punkt $\vec{x} = \vec{0}$ bildet damit das globale Optimum. Die Schwierigkeit bei der Ackley-Funktion ist nicht, den Trichter zu finden, sondern in diesem bis zum globalen Optimum herab zu steigen. Da auch die Trichterwände aus sich wiederholenden Folgen von lokalen Optima bestehen bedarf es einem optimalen Zusammenspiel zwischen Crossover und Mutation, damit der evolutionäre Algorithmus weder in einem lokalen Optimum hängen bleibt, noch von einer Trichterwand zur nächsten springt.

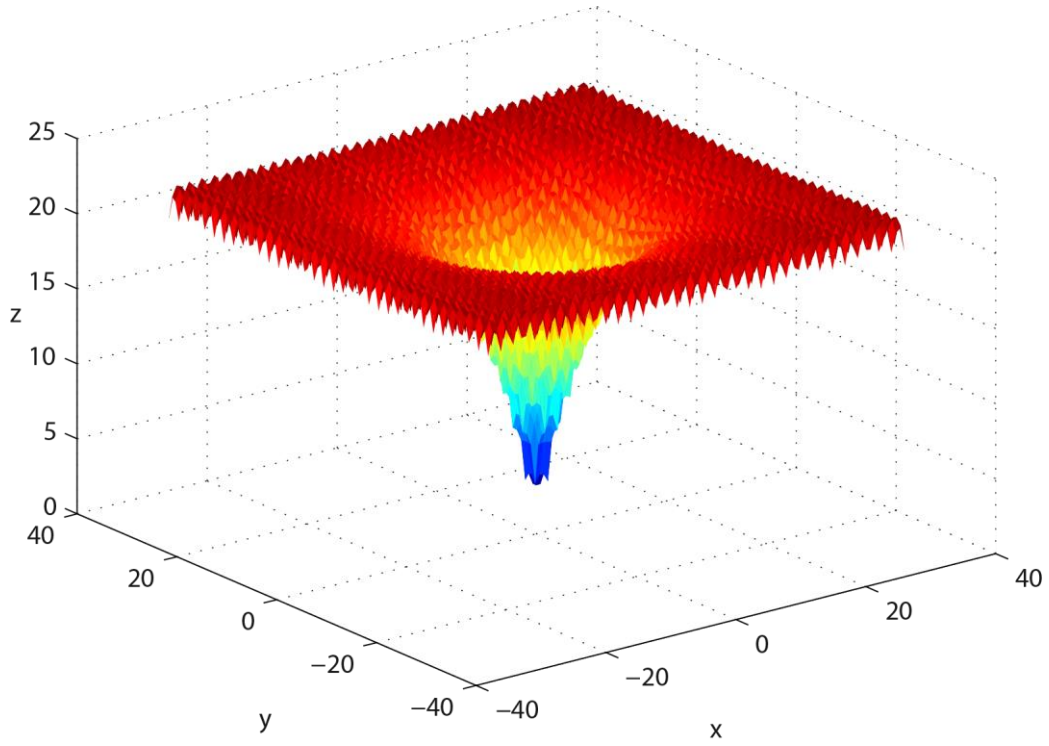


Abbildung 4.17. Grafische Darstellung der Ackley-Funktion auf einem zweidimensionalen Eingabevektor im Intervall [-32; 32]. Quelle: Eigene Darstellung.

Abbildung 4.18 zeigt das Berechnungszeitverhalten des JEP für die Ackley-Funktion. Es ergibt sich ein ähnliches Bild wie bei den vorhergehenden Funktionen – bis zu einer Dimensionsgröße von 1.000 bleibt die Berechnungszeit nahezu konstant, anschließend steigt sie stark an. Auffällig ist bei der Ackley-Funktion die starke Streuung der Berechnungszeit. Diese übersteigt jedoch auch bei einer hohen Dimensionszahl die 40 Millisekunden nicht und bleibt damit akzeptabel.

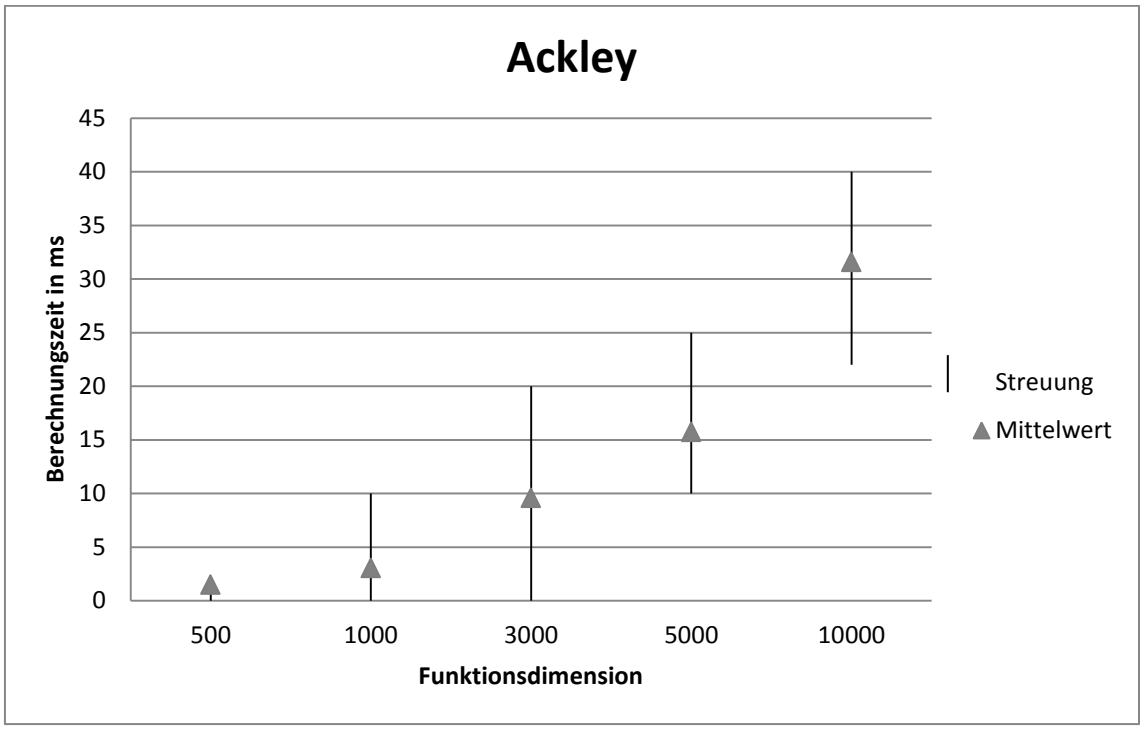


Abbildung 4.18. Darstellung der Rechenzeit-Verteilung des JEP für die Ackley-Funktion. Quelle: Eigene Darstellung.

4.3.2.5 Ergebnis

Aus den Berechnungszeitbetrachtungen ist ersichtlich, dass die Rechenzeit, die der JEP für selbst komplexere mathematische Funktionen im mehrdimensionalen Raum benötigt, kaum ansteigt, solange die Anzahl der Dimensionen die 500 nicht übersteigen. Dies ist bei der Modellierung des Antwortzeitverhaltens einer Softwarekomponente nicht zu erwarten, bedeutete es doch, dass mehr als 500 Einflussfaktoren auf das Antwortzeitverhalten gemessen und betrachtet würden. Die JEP-Bibliothek eignet sich damit für die Antwortzeitberechnung der *Formula*-Implementierung der Performance Curve Integration.

4.3.3 Validierung der *Formula*-Implementierung

Bevor die *Formula*-Implementierung zur Beantwortung der zweiten und dritten Forschungsfrage eingesetzt werden kann, muss die Korrektheit ihrer Implementierung sichergestellt werden. Zur Validierung der Korrektheit der *Formula*-Implementierung werden vier Experimente durchgeführt:

- Lineare Processor Sharing Performancekennlinie
    - Abbildung des Verhaltens einer Processor Sharing-Komponente. Die Komponente kann pro Zeiteinheit (1 Sekunde) genau den Umfang einer kompletten Anfrage abarbeiten. Bei mehreren (*n*) parallelen Aufrufen werden in einer Zeiteinheit alle *n* Anfragen zu einem 1/*n*-tel bearbeitet.

- Quadratische Performancekennlinie
    - Die quadratische Performancekennlinie impliziert, dass die Antwortzeit quadratisch von der Anzahl paralleler Zugriffe abhängt. Dies ist kein realistisches Szenario einer produktiven Softwarekomponente, ermöglicht aber die Validierung der *Formula*-Implementierung für extrem langlaufende Prozessschritte.

- Exponentielle Performancekennlinie mit einer freien Variablen
    - Exponentielle Performancekennlinien entsprechen in ihrer Form dem gängigen Antwortzeitverhalten von (nicht-limitierten) Softwarekomponenten. Unter geringer und mittlerer Last steigt das Antwortzeitverhalten kaum an, im Hochlastbereich hingegen nimmt es dramatisch zu. In diesem Experiment wird damit das erwartete Normalverhalten einer Komponente abgebildet. Die freie Variable stellt dabei einen Eingabewert dar, welcher Einfluss auf das Antwortzeitverhalten hat.

- Exponentielle Performancekennlinie mit zwei freien Variablen
    - Dieses Experiment entspricht dem vorherigen, mit dem Unterschied der weiteren freien Variablen. Hierdurch wird das Datenfeld um eine weitere Dimension erweitert, die Anzahl der Messwerte nimmt deutlich zu. Dieses Experiment dient auch dem Vergleich der Auswirkungen der Implementierungen auf die Laufzeit der Simulation. Dies wird in Abschnitt 4.3.4 beschrieben.

In jedem der Experimente wird die Korrektheit der *Formula*-Implementierung durch einen Vergleich mit der *DataTable*-Implementierung überprüft. Die Korrektheit der *DataTable*-Implementierung der Performance Curve Integration ist in (Wert 2011) belegt. Zur Überprüfung der Gleichheit der Ergebnisse beider Implementierungen wird der von Wert vorgeschlagene Aufbau erweitert. Der Experimentaufbau wird im folgenden Abschnitt beschrieben, gefolgt von der Beschreibung und den Ergebnissen der einzelnen Experimente.

4.3.3.1 Experimentaufbau

Das generierte Nutzerverhalten, insbesondere die Think Time und die Werte der Übergabeparameter durch den Nutzer (in diesem Experiment deklariert als zwei *Double*-Werte x und y) sind abhängig von durch die Simulationsumgebung generierten Zufallsvariablen. Aus diesem Grund ist keine Vergleichbarkeit der Ergebnisse gegeben,

wenn die Simulation einmal mit der *DataTable*-Implementierung der PCI und einmal mit der *Formula*-Implementierung aufgerufen wird. Die (mit sehr hoher Wahrscheinlichkeit) unterschiedlich generierten Zufallswerte würden zwangsläufig zu abweichenden Ergebnissen führen. Um exakt vergleichbare Ergebnisse zu generieren ist es notwendig, beide Implementierungen parallel aufzurufen, auf denselben Nutzereingaben. Hierzu wird eine Vergleichskomponente angelegt, in der Simulation *ComparisonComponent* genannt. Diese Vergleichskomponente ruft parallel (mittels des *Fork*-Elements) beide PCI-Implementierungen auf. Abbildung 4.19 zeigt das Repository-Diagramm des Experimentaufbaus mit den beiden Interfaces *FormulaPerformanceCurveInterface* und *DataTablePerformanceCurveInterface*. Mittels der beiden Interfaces werden die entsprechenden PCI-Implementierungen eingebunden.

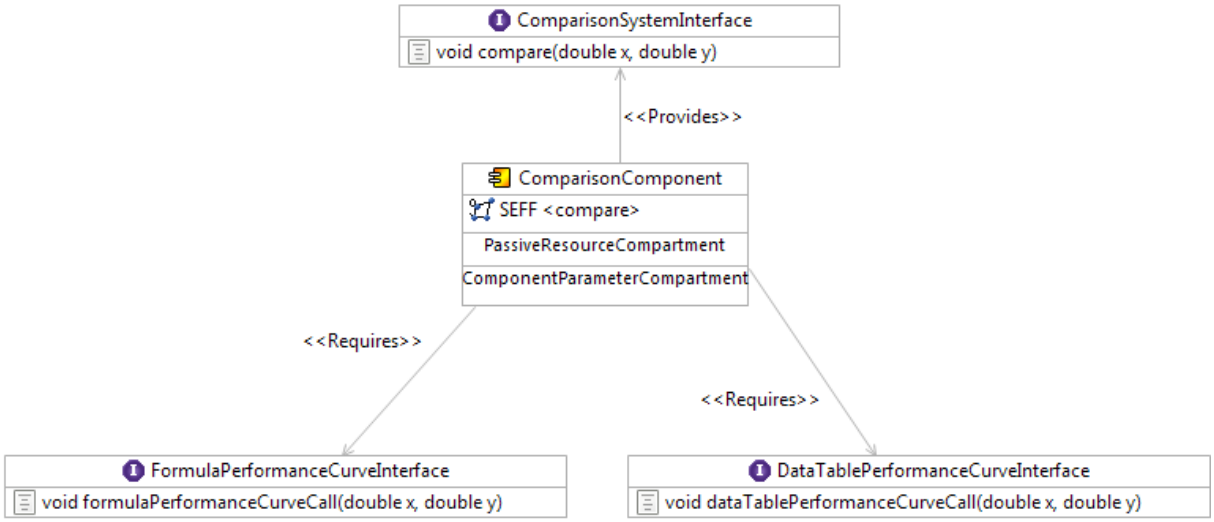


Abbildung 4.19. Repository-Diagramm des Experiment-Aufbaus. Quelle: Eigene Darstellung.

Die Logik innerhalb der Vergleichskomponente ist in Abbildung 4.20 als Service-Effekt-Spezifikation (SEFF) dargestellt.

Das *Fork*-Element erzeugt eine Parallelisierung im Kontrollfluss der Simulation. Innerhalb des *Fork*-Elements ist ein Synchronisierungspunkt definiert. Dieser Synchronisierungspunkt synchronisiert die beiden *Fork*-Zweige. Sollte einer der beiden Zweige schneller durchlaufen werden als der andere, so synchronisiert der Synchronisierungspunkt die parallelen Abläufe am Ende des Forks. Hierdurch wird gewährleistet, dass keine unerwünschten Mehrfachaufrufe verursacht werden, welche das Antwortzeitverhalten beeinflussen würden.

Innerhalb des Forks existieren zwei parallele Zweige. Im linken Zweig wird die *DataTable*-Implementierung der PCI ausgeführt, im rechten die *Formula*-Implementierung. Beide

Aufrufe geschehen parallel auf den gleichen Daten und Nutzern. Die Eingabeparameter x und y werden unverändert auf die Eingabeparameter der PCI-Methoden gemapped. Sie werden nicht bei jedem Experiment verwendet (siehe die folgenden Definitionen der verwendeten Performancekennlinien), jedoch immer übergeben, um den Experimentaufbau nicht zu verändern und die Ergebnisse damit vergleichbar zu halten. Werden die Eingabeparameter nicht verwendet, so werden sie ignoriert.

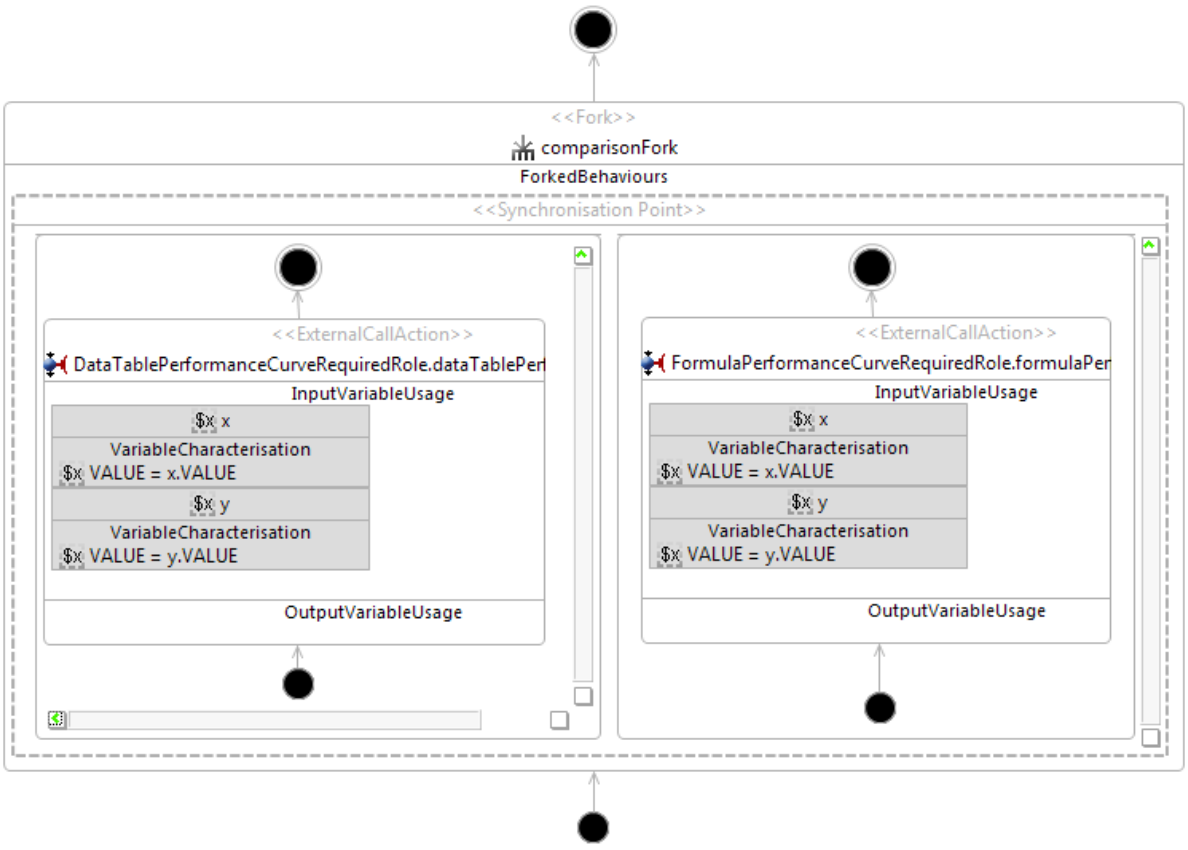


Abbildung 4.20. Service-Effekt-Spezifikation (SEFF) der Comparison-Komponente. Quelle: Eigene Darstellung.

Die im Folgenden aufgeführten Antwortzeitverteilungen beziehen sich immer auf die ExternalCallActions, mit denen die PCI-Implementierungen aufgerufen werden. In Abbildung 4.20 sind diese deutlich erkennbar.

4.3.3.2 Lineare Processor Sharing Performancekennlinie

Das Processor Sharing Performance Curve Experiment wird in (Wert 2011) genutzt, um die Korrektheit der *DataTable*-Implementierung zu belegen. In diesem Experiment wird die Performancekennlinie einer Processor Sharing Ressource abgebildet. Eine Processor Sharing Ressource ist ein in PCM implementiertes Konstrukt, das eine CPU mit Round Robin Scheduling darstellt, wobei die im Scheduling verwendeten Zeitscheiben unendlich klein sind. Hierdurch wird die Ressource exakt gleichmäßig auf alle Anfragen aufgeteilt. Hat

eine CPU-Ressource demnach eine Kapazität von *n* Rechenoperationen pro Sekunde, und erfolgen *m* parallele Anfragen, so wird jeder Anfrage $^{n}/_{m}$ Rechenoperationen zugeteilt. Das Antwortzeitverhalten einer Processor Sharing Ressource ist linear proportional zur Anzahl (gleicher) paralleler Anfragen. Es ergibt sich hierdurch eine einfache Performancekennlinie, wie sie in Abbildung 4.21 dargestellt ist.

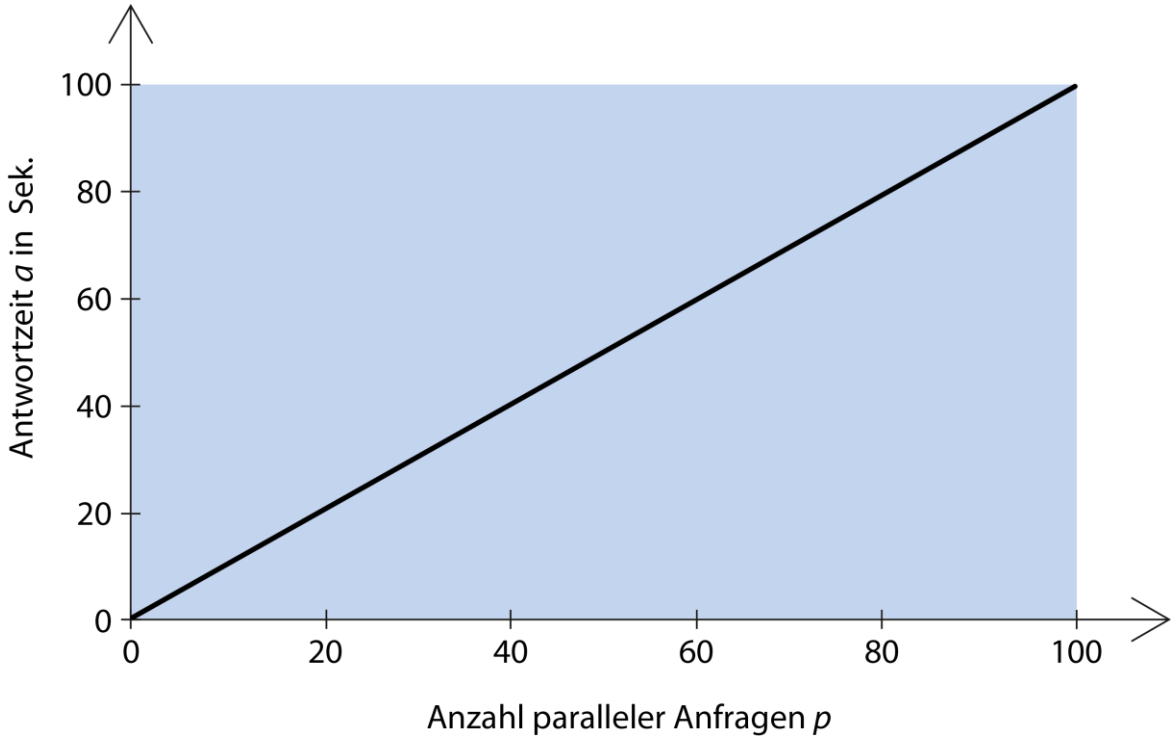


Abbildung 4.21. Antwortzeitverhalten einer Anfrage mit einem Bedarf von *n* Rechenoperationen auf einer Processor Sharing Ressource mit einer Kapazität von *n*. Quelle: Eigene Darstellung, nach (Wert 2011).

Für die *Formula*-Implementierung der PCM wird die Darstellung dieser Kurve in Form einer mathematischen Formel benötigt. Diese ist in Formel 6 gegeben.

Formel 6. Antwortzeitverhalten einer linearen Processor Sharing Ressource.

$$a(p) = p$$

Für die Validierung der *Formula*-Implementierung der Performance Curve Integration kann nach der Validierung der *DataTable*-Implementierung durch (Wert 2011) von deren Korrektheit ausgegangen werden. Die Validierung der *Formula*-Implementierung erfolgt durch einen Vergleich mit der *DataTable*-Implementierung, ein Vergleich mit einer Processor Sharing Ressource und weiteren Implementierungen in PCM kann entfallen.

Abbildung 4.22 zeigt den Vergleich der simulierten Antwortzeiten sowohl für die DataTable-ExternalCallAction als auch für die Formula-ExternalCallAction nach 20.000 simulierten Durchläufen. Es ist ersichtlich, dass die Antwortzeitverteilung in beiden Implementierungen identisch ist. Ein Vergleich der einzelnen gemessenen Antwortzeiten bestätigt dies. Auf der linearen Processor Sharing Performancekennlinie verhält sich die *Formula*-Implementierung demnach identisch zur *DataTable*-Implementierung.

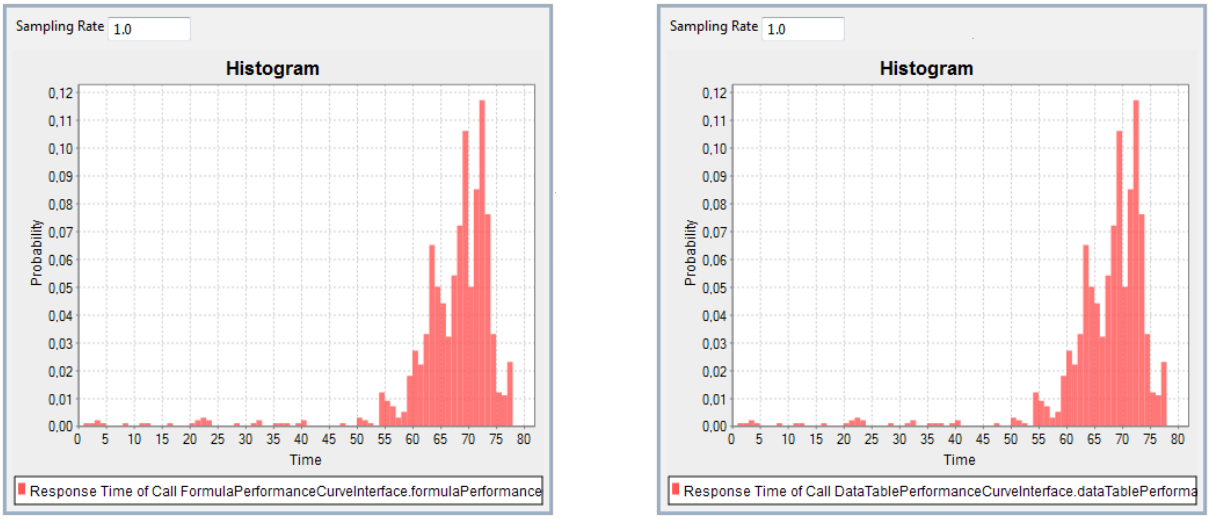


Abbildung 4.22. Vergleich der simulierten Antwortzeiten des „Processor Sharing“-Experimentaufbaus, ermittelt mit der PCI *Formula*-Implementierung (links) sowie der *DataTable*-Implementierung (rechts). Quelle: Eigene Darstellung

4.3.3.3 Quadratische Performancekennlinie

Die lineare Processor Sharing Performancekennlinie bildet das Verhalten einer Komponente mit kurzläufigen Operationen ab. Um die Korrektheit der *Formula*-Implementierung für das Verhalten der Performancekennlinien einer Komponente mit mittel- und langfristigen Operationen zu überprüfen wird ein Experiment mit einer quadratischen Performancekennlinie durchgeführt. Formel 7 definiert die quadratische Performancekennlinie.

Formel 7. Mathematische Darstellung der quadratischen Performancekennlinie.

$$a(p) = p^2$$

Die quadratische Performancekennlinie ist ein theoretisches Konstrukt, welches bei niedrigen Nutzerzahlen geringe Antwortzeiten erzeugt, jedoch mit zunehmender Anzahl paralleler Nutzer stark ansteigende Antwortzeiten liefert. Bei den betrachteten einhundert

parallelen Nutzern steigen die Antwortzeiten bis auf 10.000 Sekunden, ungefähr 2 ¾ Stunden.

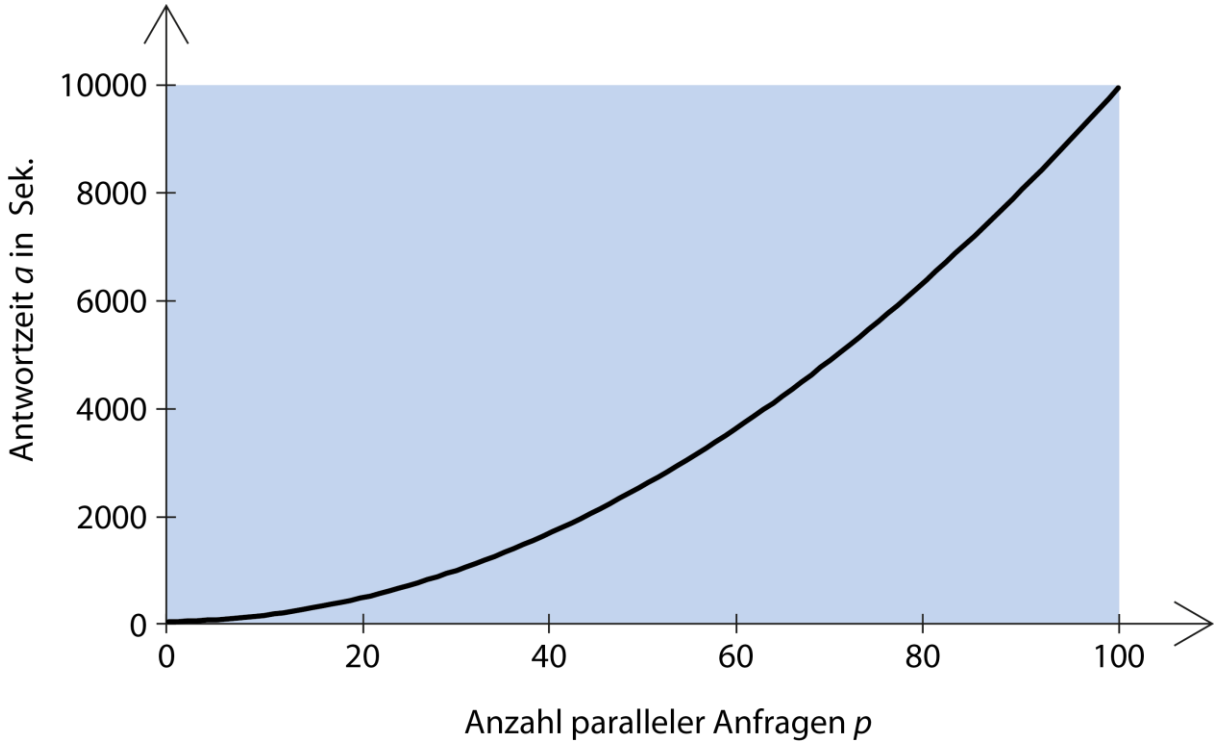


Abbildung 4.23. Antwortzeitverhalten einer Komponente mit quadratischer Performancekennlinie in Abhängigkeit von der Anzahl paralleler Anfragen. Quelle: Eigene Darstellung.

Abbildung 4.24 zeigt den Vergleich der simulierten Antwortzeitverteilung für die *Formula*- und *DataTable*-Implementierung. Die Ballung der Antwortzeiten im Hochlastbereich ergibt sich aus dem Verhältnis der Komponentenantwortzeit zur Nutzer-Think Time. Da eine Anfrage für zwischen 8.000 und 10.000 Sekunden benötigt sobald einmal alle Benutzer eine Anfrage gestartet haben, wird die Think Time von durchschnittlich drei Sekunden unerheblich. Nach kurzer Zeit warten demnach alle Nutzer nahezu die gesamte Zeit auf Rückmeldung des Systems, es sind nahezu immer alle Nutzeranfragen auf der Ressource.

Das Ergebnis der *Formula*- und *DataTable*-Implementierung ist bei der quadratischen Performancekennlinie ebenfalls identisch. Die *Formula*-Implementierung liefert auch im Hochlastbereich identische Ergebnisse zur *DataTable*-Implementierung.

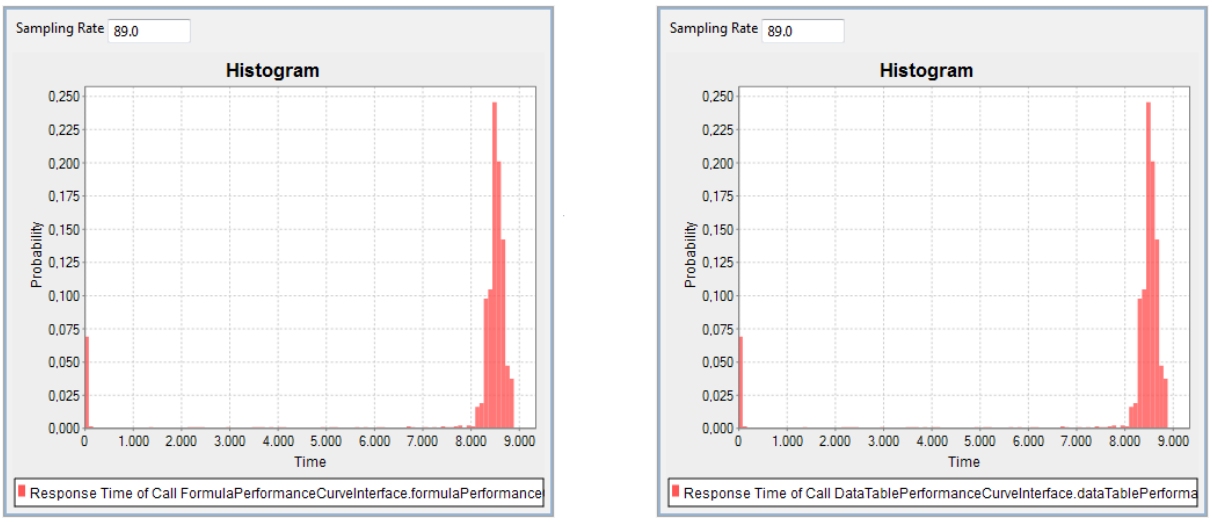


Abbildung 4.24. Vergleich der simulierten Antwortzeiten des „Quadratische Performancekennlinie“-Experimentaufbaus, ermittelt mit der PCI *Formula*-Implementierung (links) sowie der *DataTable*-Implementierung (rechts). Quelle: Eigene Darstellung.

4.3.3.4 Exponentielle Performancekennlinie mit einer freien Variablen

Die vorab vorgestellten Experimente untersuchen Performancekennlinien, die einzig von der Anzahl paralleler Zugriffe abhängen. In der Realität ist das Antwortzeitverhalten einer Komponente jedoch auch von der Art und dem Umfang der übergebenen Operationsparameter abhängig. So ist es für eine Operation, die einen Text in eine Datei oder Datenbank schreibt, relevant, ob der Text aus wenigen Byte oder mehreren Gigabyte besteht. Auch bei anderen Arten von Operationen bestehen solche Abhängigkeiten.

Im folgend vorgestellten Experiment *Exponentielle Performancekennlinie mit einer freien Variablen* wird die freie Variable *x* eingeführt. Dieses *x* dient als Offset für die Performancekennlinie und wird vom Nutzer übergeben. Die simulierte Antwortzeit ist also direkt abhängig von *x*.

Weiterhin wird ein exponentielles Antwortzeitverhalten simuliert. Dieses exponentielle Verhalten ist realitätsnah, da die Antwortzeit bei geringer und mittlerer Last nur gering ansteigt, im Hochlastbereich jedoch immens (siehe Abbildung 4.25). Dieses Verhalten wird in der Praxis regelmäßig beobachtet. Ein Grund hierfür ist beispielsweise der Scheduling-Overhead. Kommt ein System an seine CPU- oder I/O-Lastgrenzen, so nimmt dieser überproportional zu. Eine detaillierte Beschreibung dieses Verhaltens kann in (Jehle 2010) nachgelesen werden.

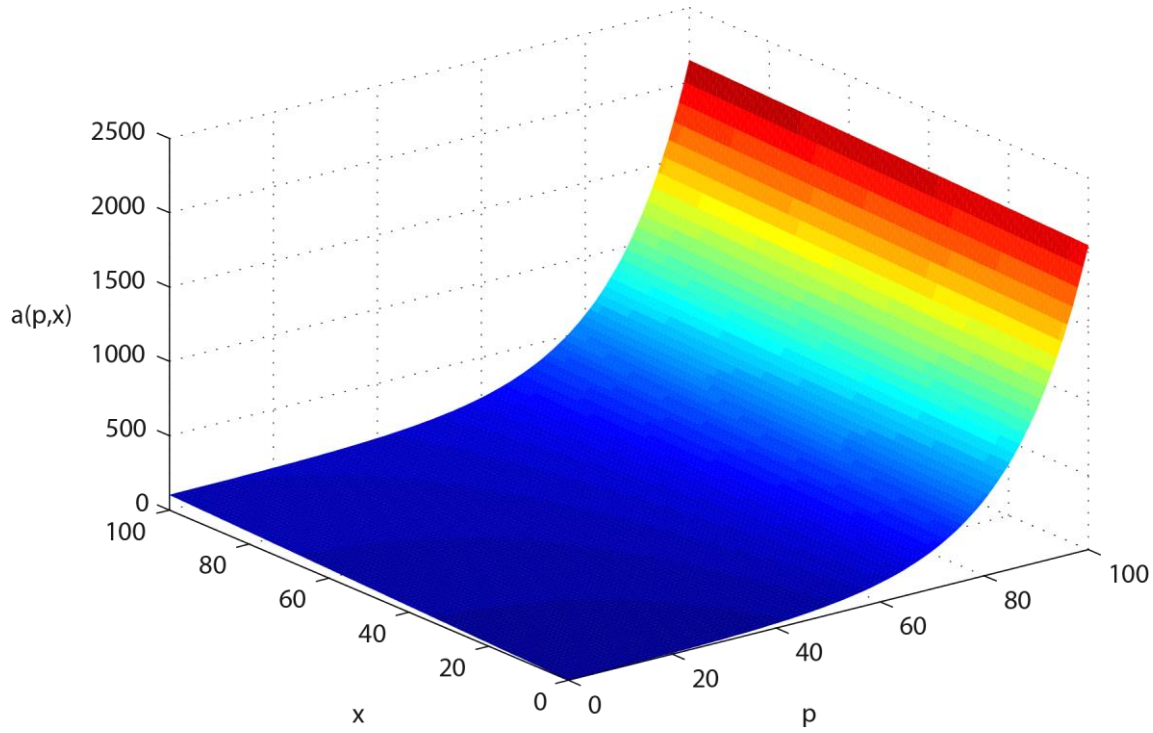


Abbildung 4.25. Antwortzeitverhalten einer Komponente mit exponentieller Performancekennlinie mit einer freien Variablen in Abhängigkeit von der Anzahl paralleler Anfragen. Quelle: Eigene Darstellung.

Formel 8 gibt die Definition der exponentiellen Performancekennlinie mit einer freien Variablen an.

Formel 8. Mathematische Darstellung der exponentiellen Performancekennlinie mit einer freien Variablen.

$$a(p,x) = x + 2^{(1+(p/10))}$$

Die simulierte Antwortzeitverteilung bei der Verwendung der exponentiellen Performancekennlinie mit einer freien Variablen ist in Abbildung 4.26 dargestellt. Die freie Variable x wird zu Beginn jedes Simulationsdurchlaufs und für jeden simulierten Nutzer neu belegt und dann an die beiden Performance Curve Integration-Implementierungen übergeben, so dass die Belegung der Variablen für beide Aufrufe identisch ist. Die Belegung von x erfolgt mittels einer Wahrscheinlichkeitsfunktion, die x mit einer Wahrscheinlichkeit von 30% den Wert 1, mit einer Wahrscheinlichkeit von 50% den Wert 2 und mit einer Wahrscheinlichkeit von 20% den Wert 3 zuweist. Diese Belegung von x resultiert in den drei Häufungspunkten rund um die Antwortzeiten 4, 5 und 6 Sekunden.

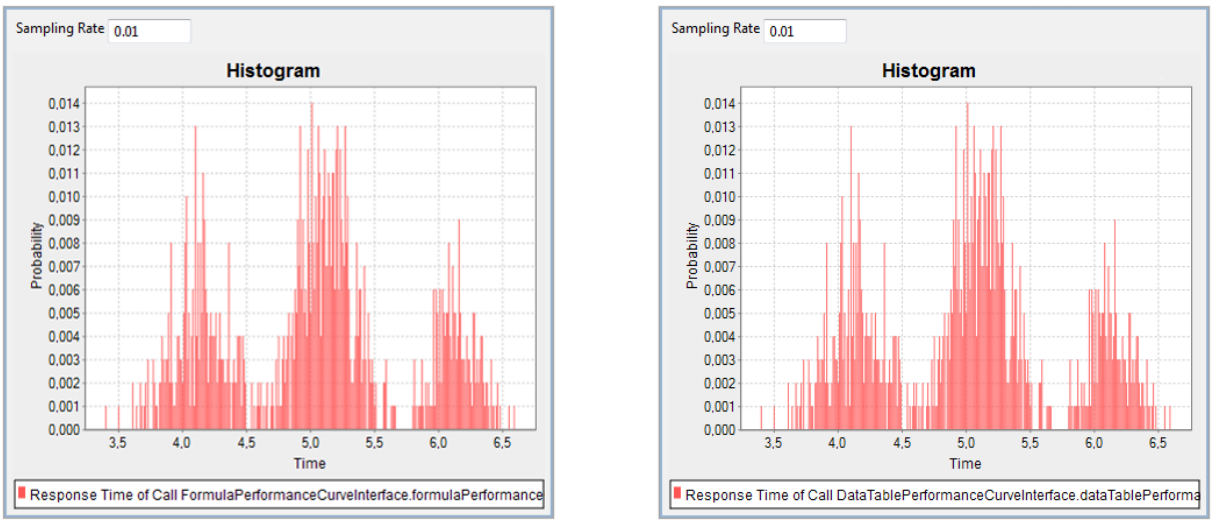


Abbildung 4.26. Vergleich der simulierten Antwortzeiten des „Exponentielle Performancekennlinie mit einer freien Variablen“-Experimentaufbaus, ermittelt mit der PCI *Formula*-Implementierung (links) sowie der *DataTable*-Implementierung (rechts). Quelle: Eigene Darstellung.

Aus Abbildung 4.26 ist erneut die Identität der Ergebnisse der *Formula*- und *DataTable*-Implementierung ersichtlich, auch die Auswertung der Einzelergebnisse bestätigen dies. Auf Basis gleicher Eingabewerte liefern die beiden PCI-Implementierungen identische Simulationsergebnisse für die exponentielle Performancekennlinie mit einer freien Variablen.

4.3.3.5 Exponentielle Performancekennlinie mit zwei freien Variablen

Zu guter Letzt erfolgt eine Abrundung des Experiments zur Validierung der Korrektheit der *Formula*-Implementierung durch die Hinzunahme einer weiteren freien Variablen. Wiederum wird eine exponentielle Performancekennlinie gewählt, da diese das natürliche Verhalten einer Softwarekomponente am ähnlichsten abbildet. Als Basis der Exponentialfunktion dient in der folgenden Untersuchung jedoch die freie Variable y. Dies simuliert eine starke Abhängigkeit des Antwortzeitverhaltens von den übergebenen Parametern, wie sie beispielsweise auftritt, wenn innerhalb einer Komponente Operationen unterschiedlichen Typs (wie beispielsweise lesende, schreibende und löschende Operationen) ausgeführt werden und die Auswahl der Operationen von den übergebenen Variablenwerten abhängen. Formel 9 definiert die exponentielle Performancekennlinie mit den zwei freien Variablen x und y.

Formel 9. Mathematische Darstellung der exponentiellen Performancekennlinie mit zwei freien Variablen.

$$a(p, x, y) = x + y^{(1+(p/10))}$$

Abbildung 4.27 stellt die exponentielle Performancekennlinie mit zwei freien Variablen grafisch dar. Zur Abbildung der vierten Dimension erfolgt eine viergeteilte Darstellung, separiert nach den Werten für y. Es ist ersichtlich, dass das Antwortzeitverhalten unter hoher Last (> 60 paralleler Nutzer) mit zunehmendem y immens zunimmt.

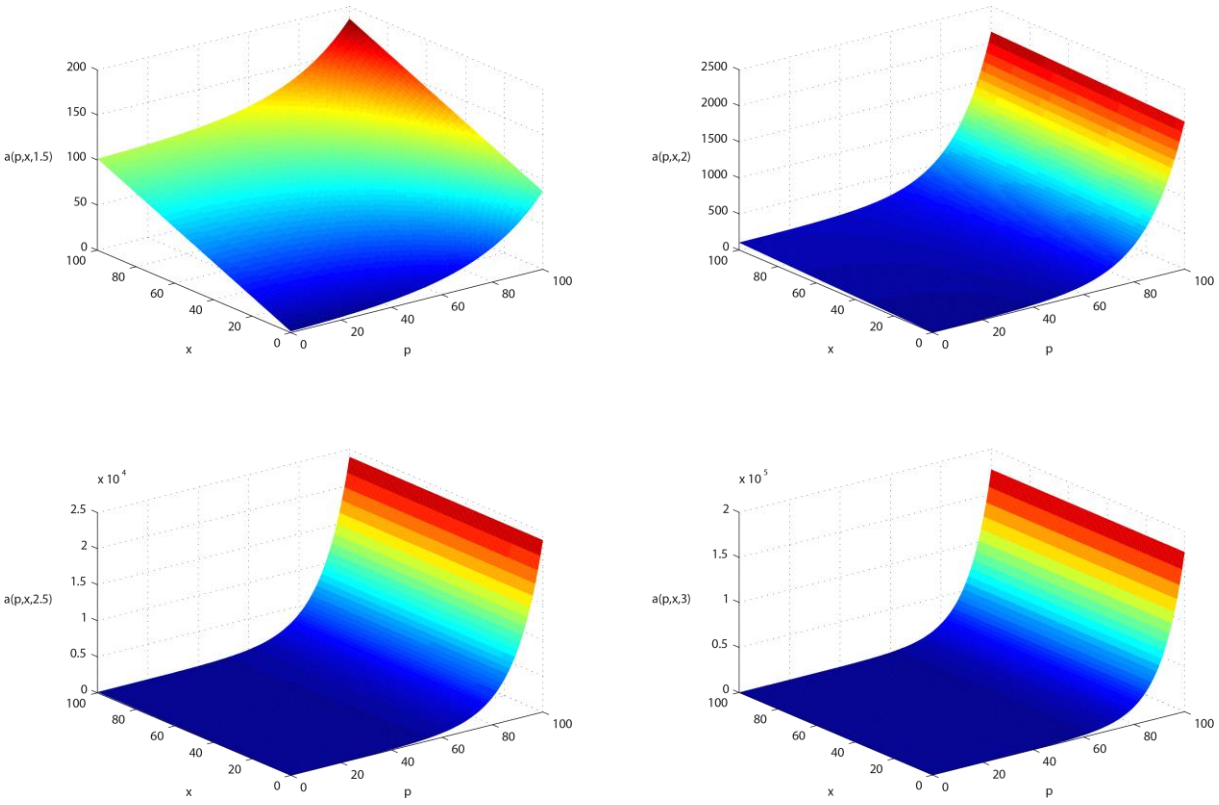


Abbildung 4.27. Antwortzeitverhalten einer Komponente mit exponentieller Performancekennlinie mit zwei freien Variablen in Abhängigkeit von der Anzahl paralleler Anfragen. Von links oben nach rechts unten: y = {1,5; 2; 2,5; 3}. Quelle: Eigene Darstellung.

Abbildung 4.28 stellt das simulierte Antwortzeitverhalten des Experimentaufbaus grafisch dar. Wie bereits in den vorhergehenden Untersuchungen sind die simulierten Antwortzeiten der *Formula*- und *DataTable*-Implementierung identisch, was auch die Einzelauswertung der Simulationsergebnisse bestätigt. Sowohl im Niedriglast- als auch im Hochlastbereich liefern die *Formula*- und *DataTable*-Implementierungen identische Ergebnisse.

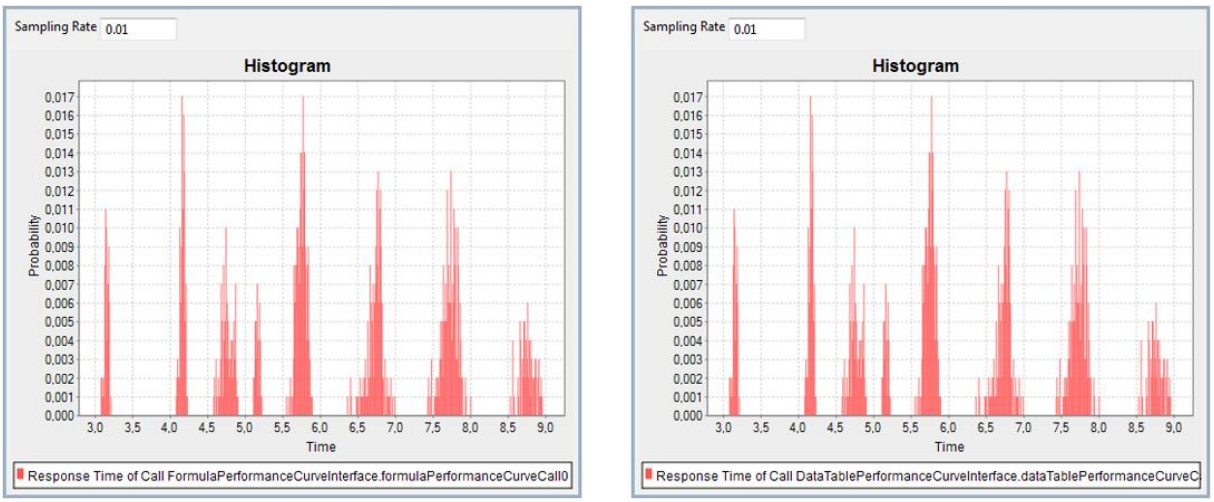


Abbildung 4.28. Vergleich der simulierten Antwortzeiten des „Exponentielle Performancekennlinie mit zwei freien Variablen“-Experimentaufbaus, ermittelt mit der PCI *Formula*-Implementierung (links) sowie der *DataTable*-Implementierung (rechts). Quelle: Eigene Darstellung.

4.3.3.6 Zusammenfassung

Die Ergebnisse dieses Experiments belegen die Korrektheit der *Formula*-Implementierung der PCI. Alle durchgeführten Untersuchungen an Performancekennlinien unterschiedlichster Ausprägungen zeigten die Identität der resultierenden Simulationsergebnisse bei der Verwendung der *Formula*- und *DataTable*-Implementierung der PCI. Die *Formula*-Implementierung kann damit für den in Kapitel 5 beschriebenen Anwendungsfall eingesetzt werden. Zudem entfällt bei der Formula-Implementierung die Beschränkung auf äquidistante Messdaten. Ob zudem Vorteile in der Simulationslaufzeit durch die Formula-Implementierung entstehen wird im folgenden Teilkapitel betrachtet.

4.3.4 Laufzeitverhalten der *Formula*-Implementierung

Das Laufzeitverhalten der PCI-Implementierungen hat große Auswirkung auf das Laufzeitverhalten der gesamten Simulation. Da die Performancekennlinie einer Komponente für jede Komponente und für jeden simulierten Anwender eines Simulationsdurchlaufs ausgewertet wird erfolgen in jeder Simulation mehrere Hunderttausend oder Millionen Performancekennlinien-Auswertungen. Bereits um wenige Millisekunden erhöhte Laufzeiten der PCI-Implementierungen summieren sich hierdurch zu einer deutlichen Verlängerung der Simulationslaufzeit.

Im Folgenden werden die Ergebnisse des Laufzeitvergleichs zwischen der *DataTable*- und der *Formula*-Implementierung der PCI vorgestellt. Als Datenbasis dient ein dreidimensionaler Messdatensatz einer SOA-Komponente aus dem in Kapitel 5.1 vorgestellten Anwendungsfall mit 20.000 Messwerten. Die Projekterfahrung zeigt, dass es sich bei dem Umfang um einen Datensatz durchschnittlicher Größe und Komplexität handelt.

Zur Untersuchung des Laufzeitverhaltens der PCI-Implementierungen dient der gleiche Experimentaufbau, der bereits zur Validierung der *Formula*-Implementierung eingesetzt wurde (siehe Abschnitt 4.3.3.1). Mittels dieses Experimentaufbaus erfolgen 100.000 Simulationsdurchläufe, jeweils mit je einem parallelen Durchlauf durch die *DataTable*-Implementierung und die *Formula*-Implementierung. Zur Messung der Laufzeiten der beiden Performancekennlinien-Aufrufe wurde der PCM-Programmcode so angepasst, dass die jeweiligen Messwerte in eine Datei geschrieben werden, ohne dass dieser Schreibaufwand die Messung beeinflusst.

Abbildung 4.29 stellt die Häufigkeit der Laufzeiten der *DataTable*-Implementierung sowie die kumulierte Wahrscheinlichkeit des Auftretens einer Laufzeit in Millisekunden dar. Es geht aus der Grafik deutlich hervor, dass die PCI-*DataTable*-Implementierung in nahezu allen Fällen eine Laufzeit zwischen 4,5 und 5 Millisekunden hat. Eine Laufzeit unter 4 Millisekunden tritt in keinem Fall auf, vereinzelt dauert ein PCI-*DataTable*-Aufruf bis zu 10 Millisekunden.

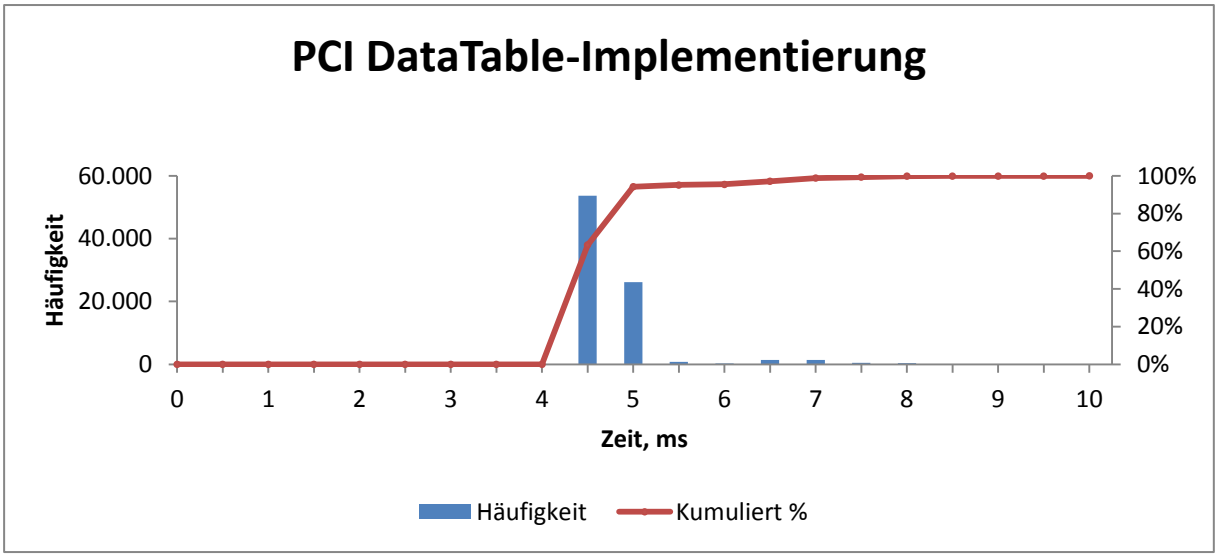


Abbildung 4.29. Häufigkeit und kumulierte Wahrscheinlichkeitsverteilung der Berechnungszeit der PCI-DataTable-Implementierung. Quelle: Eigene Darstellung

Im Vergleich hierzu stellt Abbildung 4.30 die gemessenen Laufzeiten der *Formula*-Implementierung dar. 70 Prozent aller Aufrufe beantwortet die *Formula*-Implementierung innerhalb von 0,002 Millisekunden, wenige Anfragen innerhalb von bis zu 0,01 Millisekunden, mit einer Obergrenze bei 0,02 Millisekunden.

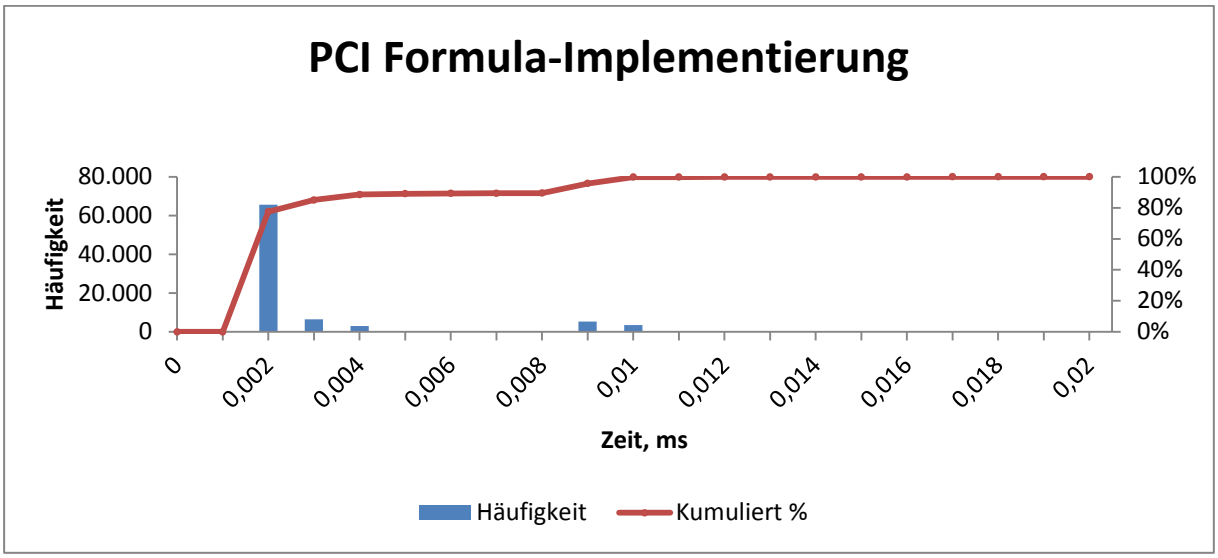


Abbildung 4.30. Häufigkeit und kumulierte Wahrscheinlichkeitsverteilung der Berechnungszeit der PCI-*Formula*-Implementierung. Quelle: Eigene Darstellung

Im Schnitt ist die *Formula*-Implementierung damit um einen Faktor 2.000 schneller als die *DataTable*-Implementierung, im schlechtesten Fall immer noch um den Faktor 200. Natürlich nimmt die Laufzeit der *Formula*-Implementierung mit der Komplexität der gesetzten Formel zu (wie in Abschnitt 4.3.2.4 gezeigt), die Laufzeit der *DataTable*-Implementierung steigt jedoch deutlich steiler mit zunehmender Größe der Messdatensätze. Für die Simulation umfangreicher Systeme eignet sich die *Formula*-Implementierung damit deutlich besser als die *DataTable*-Implementierung der PCI.

4.3.5 Zusammenfassung

In diesem Teilkapitel erfolgte die Vorstellung der *Formula*-Implementierung der Performance Curve Integration in das PCM-Framework. Das PCM-Framework unterstützt von sich aus keine Einbindung von Komponentenmodellen in Form von Antwortzeitkurven. Diese Lücke schließt die Performance Curve Integration. Die ursprüngliche Implementierung von (Wert 2011) weist jedoch einige Schwachstellen auf, beispielsweise im Laufzeitverhalten bei großen Datensätzen, in der zwingenden Äquidistanz der zugrundeliegenden Messwerte und im Grenzwertverhalten. Diese Probleme werden durch die vorgestellte *Formula*-Implementierung der Performance Curve Integration behoben.

Die Einbindung der Komponentenmodelle in das PCM-Framework mittels der *Formula*-Implementierung ist eine elementare Voraussetzung für die Durchführung des kontrollierten Softwareexperiments und beantwortet den ersten Teil der zweiten Forschungsfrage. Aus diesem Grund erfolgt eine umfangreiche Validierung der Implementierung. Diese Validierung wird in den vorangehenden Abschnitten ebenfalls umfangreich erläutert und zeigt, dass das

von der *Formula*-Implementierung umgesetzte Verhalten identisch ist zum Verhalten der *DataTable*-Implementierung. Da die Korrektheit der *DataTable*-Implementierung von (Wert 2011) bereits belegt wurde ist durch diese Identität des Verhaltens von der Korrektheit der *Formula*-Implementierung auszugehen.

Zuletzt erfolgt eine Betrachtung des Laufzeitverhaltens der *Formula*-Implementierung im Vergleich zu Werts *DataTable*-Implementierung. Die Betrachtung zeigt, dass die *Formula*-Implementierung bereits bei kleinen Messdatensätzen um ein Vielfaches performanter ist als die *DataTable*-Implementierung.

Die *Formula*-Implementierung löst damit die identifizierten Probleme der *DataTable*-Implementierung. Sie benötigt keine äquidistanten Messdaten und erlaubt eine Definition des Verhaltens in und über den Grenzwertbereich hinaus. Zudem ermöglicht die *Formula*-Implementierung den Einsatz vieler Performancekennlinien auch auf komplexen Datensätzen, da sie ein deutlich geringeres Laufzeitverhalten besitzt als die *DataTable*-Implementierung. Sie eignet sich damit hervorragend für die Einbindung der Antwortzeit-Komponentenmodelle in die Simulation des Anwendungsfalls in Kapitel 5.

4.4 Konfiguration des evolutionären Algorithmus

Die Approximationsgenauigkeit und das Konvergenzverhalten eines evolutionären Algorithmus hängen stark vom Typ und von der Konfiguration des Algorithmus ab. Für die Überprüfung der in Kapitel 1.2 beschriebenen Annahmen kommt ein genetischer Algorithmus nach (Goldberg 1989) zum Einsatz (vgl. Teilkapitel 2.4.3). Approximationsgenauigkeit und Konvergenzverhalten eines genetischen Algorithmus sind durch die fünf in Teilkapitel 2.4.4.1 beschriebenen Konfigurationsparameter bestimmt. Wie im genannten Teilkapitel beschrieben lässt sich im Vorhinein keine optimale Konfiguration des genetischen Algorithmus bestimmen, da diese vom Anwendungsfall abhängt. Im Folgenden wird beschrieben, wie durch eine statistische Analyse zahlreicher Modellierungsläufe eine möglichst optimale Konfiguration des genetischen Algorithmus für den in dieser Arbeit betrachteten Anwendungsfall ermittelt wurde. Anschließend wird diese Analyse erweitert und überprüft, indem ein evolutionärer Algorithmus eingesetzt wird, um eine optimale Konfiguration für den eingesetzten evolutionären Algorithmus zu ermitteln.

4.4.1 Optimale Konfiguration

Es ist ersichtlich, dass die einzelnen in Teilkapitel 2.4.4.1 beschriebenen Konfigurationsparameter nicht getrennt voneinander betrachtet werden können. So nimmt der Effekt der Mutationsoperation mit zunehmender Genome-Länge ab, und diese wiederum

ist abhängig von der Populationsgröße. Diese Korrelationen erschweren die Bestimmung einer optimalen Konfiguration für den genetischen Algorithmus.

Als optimal wird in dieser Arbeit eine Konfiguration des genetischen Algorithmus bezeichnet, wenn der Algorithmus mit ihr mit einer maximal hohen Wahrscheinlichkeit innerhalb einer möglichst kurzen Zeitspanne zu einem Modell mit einem Fehlerwert unterhalb des Fehlergrenzwerts wie in Teilkapitel 2.4.2.2 beschrieben gelangt. Der Fokus liegt dabei auf der Wahrscheinlichkeit für das akzeptable Modell, der Zeitfaktor wird nur nachrangig betrachtet, soweit er sich im Rahmen weniger Stunden befindet. Für die Simulation wird jede Systemkomponente nur einmal modelliert, so dass selbst eine Modellierungszeit von mehreren Stunden akzeptabel bleibt und durch den Einsatz von mehr Hardware ausgeglichen werden kann. In den durchgeführten Untersuchungen trat eine so lange Modellierungszeit nicht auf.

4.4.2 Analyse einer optimalen Konfiguration für den Anwendungsfall

Zur Identifikation einer optimalen Konfiguration für den in dieser Arbeit betrachteten Anwendungsfall werden auf einem repräsentativen Messdatensatz Modellierungsläufe mit verschieden konfigurierten genetischen Algorithmen gestartet. Im Rahmen dieser Untersuchung erfolgt mithilfe eines vierdimensionalen Rasters über die in Teilkapitel 2.4.4.1 definierten Konfigurationsparameter eine Betrachtung der Modellierungsergebnisse des genetischen Algorithmus unter allen Permutationen dieser Konfigurationsparameter. Um die Anzahl der Permutationen etwas einzugrenzen werden jedoch nicht alle möglichen Ausprägungen der Konfigurationsparameter berücksichtigt, sondern eine Auswahl. Tabelle 3 definiert die betrachteten Werte der einzelnen Konfigurationsparameter.

Tabelle 3. Betrachtete Wertebereiche der Konfigurationsparameter zur Identifikation einer optimalen Konfiguration

Parameter	**Betrachteter Wertebereich**
Populationsgröße	100, 500, 1.500, 3.000, 5.000, 10.000
Genome-Länge	11, 21, 41, 101
Mutationswahrscheinlichkeit	0..100 in Zehnerschritten
Crossover-Wahrscheinlichkeit	0..100 in Zehnerschritten

Sinnvolle Wertebereiche für die Populationsgröße und die Genome-Länge wurden vorab in Experimenten ermittelt. Eine Populationsgröße unter 100 besitzt nicht ausreichend Varianz in den erstellten Modellen der Individuen, so dass sich ein mutmaßlich erfolgreiches Modell zu dominant in der Population hält. Bei Populationen mit mehr als 5.000 Individuen entwickelt

sich der Evaluator-Singleton (siehe Teilkapitel 4.2.6) zum Bottleneck, und gleichzeitig erzeugt das CPU-Scheduling der daraus resultierenden mehr als 5.000 Threads (siehe Kapitel 4.2) so viel Overhead, dass die Laufzeiten für eine Generation sprunghaft ansteigen.

Eine solche experimentelle Einschränkung der Wertebereiche ist bei den Konfigurationsparametern Mutations- und Crossoverwahrscheinlichkeit nicht möglich. Die Vorab-Experimente zeigten jedoch, dass kleine Veränderungen in diesen Wahrscheinlichkeiten um wenige Prozent das Ergebnis nicht merklich veränderten. Aus diesem Grund ist eine Betrachtung der Wertebereiche der beiden Wahrscheinlichkeiten in Schritten von 10 Prozentpunkten sinnvoll.

4.4.2.1 Messdatensatz

Zur Gewinnung statistisch relevanter Ergebnisse wird für die in diesem Teilkapitel vorgestellte Untersuchung ein Messdatensatz ausreichender Größe und hoher Qualität benötigt, der repräsentativ für die im Anwendungsfall verwendeten Systeme (vgl. Kapitel 5.1) ist. Diese werden mittels des synthetischen Benchmarks Zachmanntest (Bögelsack et al. 2011) erhoben. (Bögelsack 2011) zeigt, dass mithilfe des synthetischen Benchmarks Performance-Informationen ermittelt werden können, die ein objektives Bild der Performance einer Unternehmensanwendung und ihrer Komponenten geben. Zudem ist die Messung mittels des Zachmanntests sehr stabil, so dass bei einem hierdurch gewonnenen Messdatensatz von einem geringen Messfehler um die fünf Prozent ausgegangen werden kann.

Unter Anwendung des Zachmanntests wird für die in diesem Teilkapitel beschriebene Untersuchung ein Messdatensatz von 2.240 Messwerten erzeugt. Die Messwerte spiegeln das Verhalten der betrachteten Unternehmensanwendung auf x86-Hardware mit 4 bis 24 Prozessoren wider. Auf jeder dieser Prozessor-Konfigurationen wird das Anwendungsverhalten bei der Konfiguration von 4 bis 32 Work-Prozessen und einer Last von 4 bis 32 Power-Usern (vgl. (Bögelsack et al. 2011)) untersucht. Jede Konstellation aus Prozessor-Anzahl, definierten Work-Prozessen und Power-Usern wird dabei fünf Mal vermessen. Die Messung der Benchmark-Werte ist für diese Arbeit nicht relevant und ist im Detail in (Tertilt/Krcmar 2012) und (Tertilt et al. 2012) beschrieben.

4.4.2.2 Laufzeit eines Modellierungslaufs

Ziel der in diesem Teilkapitel beschriebenen Untersuchung ist die Analyse, welche Permutationen der genannten Konfigurationsparameter zu akzeptablen Modellierungsergebnissen (also Modellen unter dem Fehlergrenzwert) führen. Die Laufzeit

der Modellierung ist, wie oben erwähnt, zweitrangig. Nichtsdestotrotz besteht die Notwendigkeit, die Modellierungszeit zu begrenzen.

In der vorliegenden Untersuchung werden die Modellierungsläufe nach zwei Stunden beendet und die bis dahin gewonnenen Modelle ausgewertet. Untersuchungen der Zwischenresultate ergeben, dass die maximale Konvergenz in über 90 Prozent der Modellierungsläufe bereits deutlich früher (in unter einer Stunde) erreicht wurde, der Algorithmus also keine besseren Modelle mehr identifizieren konnte.

4.4.2.3 Anzahl der Modellierungsläufe

Um zu einer statistisch belastbaren Aussage zu gelangen erfolgen pro Permutation zehn Modellierungsläufe. Basierend auf den oben angegebenen Wertebereichen ergibt sich damit eine Gesamtzahl von 2.904 Modellierungsläufen.

4.4.2.4 Verwendete Hardware

Für die Untersuchung werden vier x86-Server mit je vier 12 Core AMD Opteron 2.0 GHz Prozessoren und 256 GB Arbeitsspeicher eingesetzt. Um die große Anzahl von 2.904 Modellierungsläufen in einer für die Untersuchung akzeptablen Zeit durchführen zu können werden auf jedem der vier Server fünf virtuelle Maschinen aufgesetzt, jeweils mit 8 virtuellen CPU und 20 GB Arbeitsspeicher. Die restlichen 8 Prozessorkerne sowie der nicht verteilte Arbeitsspeicher werden bewusst frei gehalten, um CPU-Scheduling-Engpässe zwischen den virtuellen Maschinen zu vermeiden.

4.4.2.5 Laufzeit der Untersuchung

Durch diese Aufteilung ergibt sich eine Anzahl von 145 Modellierungsläufen pro virtueller Maschine. Bei einer Modellierungszeit von zwei Stunden pro Lauf zuzüglich einer Viertelstunde Cool-Down-Phase summierte sich die Laufzeit der Untersuchung auf insgesamt 290 Stunden. Die Steuerung der Untersuchung, insbesondere das Starten und Stoppen des genetischen Algorithmus auf jeder Maschine mit unterschiedlichen Konfigurationsparametern, erfolgt mittels eines Shell-Skripts. Hierdurch ist ein Betrieb der Modellierungen sieben Tage die Woche, 24 Stunden am Tag möglich. Für die Laufzeit der Untersuchung ergibt dies eine Gesamtzeit ohne Vorbereitung von etwas mehr als 12 Tagen.

4.4.2.6 Ergebnisse

Verursacht durch den Umfang der Untersuchung ergibt sich ein umfangreicher Datensatz an Modellierungsergebnissen von über fünf Millionen Einträgen. In Abbildung 4.31 und Abbildung 4.32 erfolgt eine Illustration der Ergebnisse der Untersuchung.

Die erste Grafik (Abbildung 4.31) verdeutlicht den durchschnittlichen Fehlerwert des genetischen Algorithmus in Abhängigkeit von den Konfigurationsparametern Genom-Länge und Populationsgröße. Auf den ersten Blick ist ersichtlich, dass ein zu großes Genom (Länge 201) zu einem deutlich schlechteren Ergebnis führt. Dies ist zum einen auf die längere Evaluierungszeit eines einzelnen Genoms zurückzuführen, zum anderen auf den durch die Länge stark reduzierten Effekt des Mutations-Operators (siehe Abschnitt 2.4.1.2.3). Weiterhin ist ersichtlich, dass der Fehlerwert mit zunehmender Population steigt – jedoch ausschließlich bei einem zu großen Genom. Betrachtet man die kürzeren Genome, so bleibt der durchschnittliche Fehlerwert nahezu konstant.

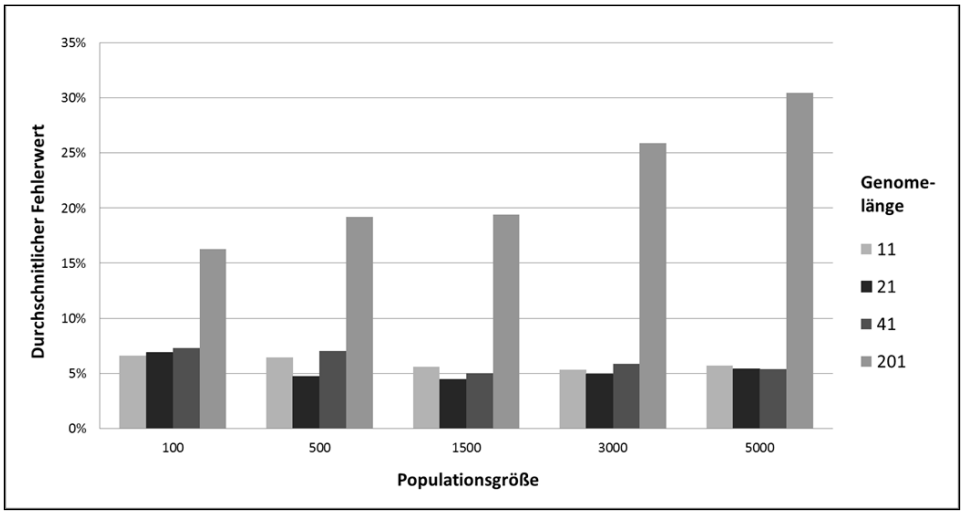


Abbildung 4.31. Durchschnittlicher Fehlerwert in Abhängigkeit von der Populationsgröße und der Genome-Länge. Quelle: (Tertilt et al. 2012).

Bis auf überlange Genome verhält sich der evolutionäre Algorithmus ab einer Populationsgröße von 1.500 Individuen nahezu konstant. Der durchschnittliche Fehlerwert von ungefähr fünf Prozent schwankt um Bruchteile von Prozentpunkten, was auf den Nichtdeterminismus des evolutionären Algorithmus zurückzuführen ist. Ein Versuch mit einer Populationsgröße von 10.000 Individuen führt durch die hohe Anzahl erzeugter Threads und den damit verbundenen Verwaltungs-Overhead der Java Virtual Machine jedoch zu einem deutlich schlechteren Ergebnis, so dass von einer weiteren Betrachtung größerer Populationen abgesehen wird.

Abbildung 4.32 stellt die Konfigurationsvarianten der Konfigurationsparameter Mutation- und Crossover-Wahrscheinlichkeit dar, die zu einem durchschnittlichen Fehlerwert von unter fünf Prozent führen, dem akzeptablen Modellfehler (vgl. Abschnitt 2.3.2.1).

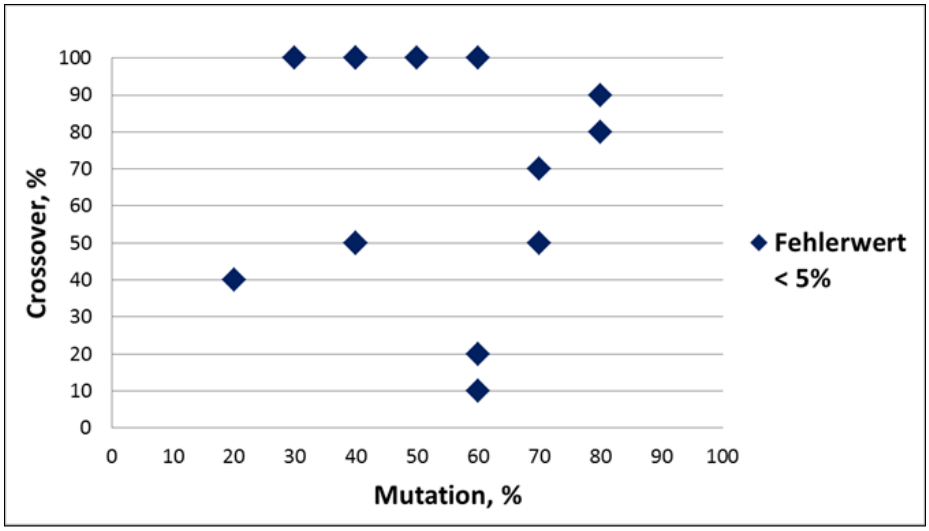


Abbildung 4.32. Zu einem Fehlerwert < 5% führende Kombinationen aus Mutations- und Crossover-Wahrscheinlichkeit. Quelle: (Tertilt et al. 2012).

Bei den Konfigurationsparametern Mutations- und Crossover-Wahrscheinlichkeit wird eine deutlich höhere Diversifikation möglicher Konfigurationswerte sichtbar. Eine Interpretation der grafischen Darstellung ergibt, dass eine Crossover-Wahrscheinlichkeit von 100 % dann zu guten Ergebnissen führt, wenn die Mutations-Wahrscheinlichkeit zwischen 30 und 60 % liegt. Bei höherer Mutations-Wahrscheinlichkeit werden gute Ergebnisse erreicht, indem die Crossover-Wahrscheinlichkeit gesenkt wird.

4.4.2.7 Zusammenfassung

Mittels statistischer Analyse lassen sich für die Konfigurationsparameter des evolutionären Algorithmus zur Modellierung des Performance-Verhaltens von Unternehmensanwendungen optimale Wertebereiche analysieren. Der verwendete Messdatensatz von 2.240 Messwerten dient hierbei als Modellierungsbasis und Vergleichswert, die durchgeführten 2.904 Modellierungsläufe liefern Messdaten zur Modellierung im Umfang von über 5 Millionen auswertbaren Datensätzen.

Für die Parameter Populationsgröße und Genom-Länge sind die identifizierten Wertebereiche recht klein und erlauben eine optimale Konfiguration. So liefert eine Populationsgröße von 1.500 Individuen, in Verbindung mit einer Genom-Länge von 21 Elementen das beste Ergebnis, gefolgt von einer Populationsgröße von 5.000 Individuen und einer Genom-Länge von 41 Elementen. Die Ergebnisse zu den Konfigurationsparametern Mutations- und Crossover-Wahrscheinlichkeit hingegen sind deutlich inhomogener, verschiedene widersprüchliche Kombinationen führen zu guten Modellen. Sie ermöglichen

eine gute Konfiguration, bedürfen jedoch einer weiteren Analyse. Diese wird in den folgenden Teilkapiteln durchgeführt.

4.4.3 Ermittlung einer optimalen Konfiguration mittels eines genetischen Algorithmus

Die Ermittlung der optimalen Konfiguration eines genetischen Algorithmus ist, wie im vorangehenden Abschnitt beschrieben, eine komplexe Problemstellung. Im Folgenden wird beschrieben, wie ein genetischer Algorithmus verwendet wird, um den in dieser Arbeit verwendeten genetischen Algorithmus zur Modellierung des Antwortzeitverhaltens von Softwarekomponenten (wie in Kapitel 2.4 beschrieben) zu konfigurieren. Zur besseren Unterscheidung der beiden genetischen Algorithmen wird der genetische Algorithmus zur Modellierung des Antwortzeitverhaltens im Folgenden Modellierungs-GA genannt, der übergeordnete, den Modellierungs-GA konfigurierende genetische Algorithmus Meta-GA. Diese Bezeichnungen finden nur in diesem Teilkapitel Anwendung, alle Bezüge auf den genetischen Algorithmus im übrigen Teil der Arbeit beziehen sich immer auf den Modellierungs-GA.

Ziel dieser Untersuchung ist der Abgleich der in Teilkapitel 4.4.2 gewonnenen Ergebnisse aus der statistischen Analyse mit der Konfiguration, die der Meta-GA für den Modellierungs-GA berechnet. Die statistische Analyse resultierte in mehreren möglichen Konfigurationen (siehe Abschnitt 4.4.2.6), das Ergebnis des Meta-GA dient damit der Auswahl der in der Experimentdurchführung (siehe Teilkapitel 5.3) eingesetzten Konfiguration. Um die Vergleichbarkeit zu gewähren erfolgt die Untersuchung ebenfalls auf dem in Abschnitt 4.4.2.1 definierten Messdatensatz.

4.4.3.1 Vorgehen

Der Meta-GA zur Konfiguration des Modellierungs-GA folgt dem Aufbau eines genetischen Algorithmus wie in Teilkapitel 2.4.3 beschrieben. Als Individuen werden jedoch Instanzen des Modellierungs-GA verwendet. Abbildung 4.33 stellt den Aufbau des Meta-GA grafisch dar.

Die Abbildung verdeutlicht die Zweistufigkeit des Ansatzes. Der links abgebildete Meta-GA operiert, dem Prinzip der genetischen Algorithmen folgend, auf einem Satz von Individuen. Für jeden dieser rechts abgebildeten Individuen wird in jeder Generation ein Fitnesswert berechnet. Anhand dieses Fitnesswertes erfolgen eine Auswahl sowie eine Modellanpassung mittels Rekombination und Mutation. Dass es sich bei den einzelnen Individuen wiederum um genetische Algorithmen handelt ist für den Meta-GA nicht, oder besser gesagt nur für seine Laufzeit, relevant.

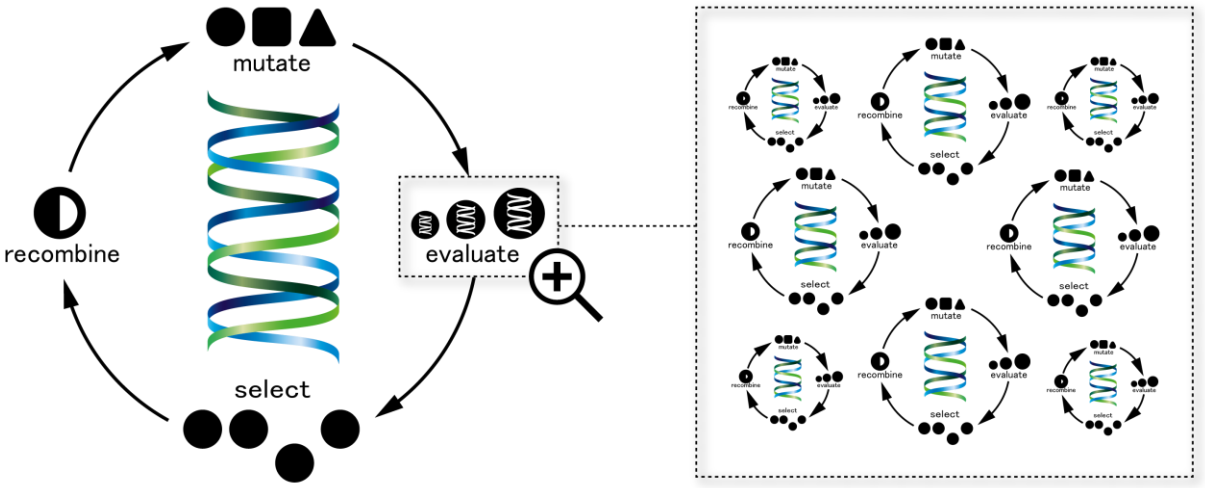


Abbildung 4.33. Struktur des Meta-GA zur Konfiguration des Modellierungs-GA.

4.4.3.1.1 Ablauf des Meta-GA

Jedes Individuum des Meta-GA enthält einen Satz von Konfigurationsparametern (wie in Abschnitt 4.4 beschrieben) sowie eine ausführbare Instanz des Modellierungs-GAs. Die Individuen werden bei ihrer Erzeugung in der Initialisierungsphase des Meta-GA zufällig konfiguriert. Nach der Erzeugung der Individuen erfolgt die Ausführung der anhand der Konfigurationsparameter konfigurierten Modellierungs-GAs in je einem eigenen Thread. Hierbei ist zu beachten, dass die Populationsgröße des Meta-GA kleiner der Anzahl vorhandener physischer CPU ist, um Verzerrungen des Ergebnisses durch Scheduling-Effekte zu vermeiden.

Nach dem vorgegebenen Zeitlimit von einer Stunde werden die Modellierungs-GAs unterbrochen. Die Analyse des Approximationsverhaltens des genetischen Algorithmus in der in Teilkapitel 4.4.2 vorgestellten Untersuchung ergibt, dass mehr als neun von zehn Modellierungsläufe in deutlich unter einer Stunde einen Zustand erreichen, in welchem nur noch vernachlässigbare Modellverbesserungen erfolgen (siehe hierzu insbesondere Abschnitt 4.4.2.2). Ein längerer Modellierungszeitrahmen des Modellierungs-GAs würde die Laufzeit des Meta-GA deutlich erhöhen, jedoch nur mit einer Wahrscheinlichkeit von unter zehn Prozent zu einem besseren Ergebnis führen.

Nach der Unterbrechung aller Modellierungs-GAs werden die Ergebnisse der Individuen verglichen. Die Fitness eines Individuums ergibt sich aus dem Modell-Fehlerwert des jeweiligen Modellierungs-GAs. Anhand dieses Fehlerwerts werden die Individuen sortiert. Jedes Individuum der schlechtesten 50 Prozent wird (als Kind-Individuum) durch je zwei zufällig gewählte Individuen der besten 50 Prozent (den Eltern-Individuen) für die nächste Generation neu konfiguriert. Hierbei kommen der Crossover- und der Mutationsoperator zum

Einsatz (wie im Folgenden beschrieben). Abbildung 4.34 beschreibt grafisch den Ablauf des Meta-GA.

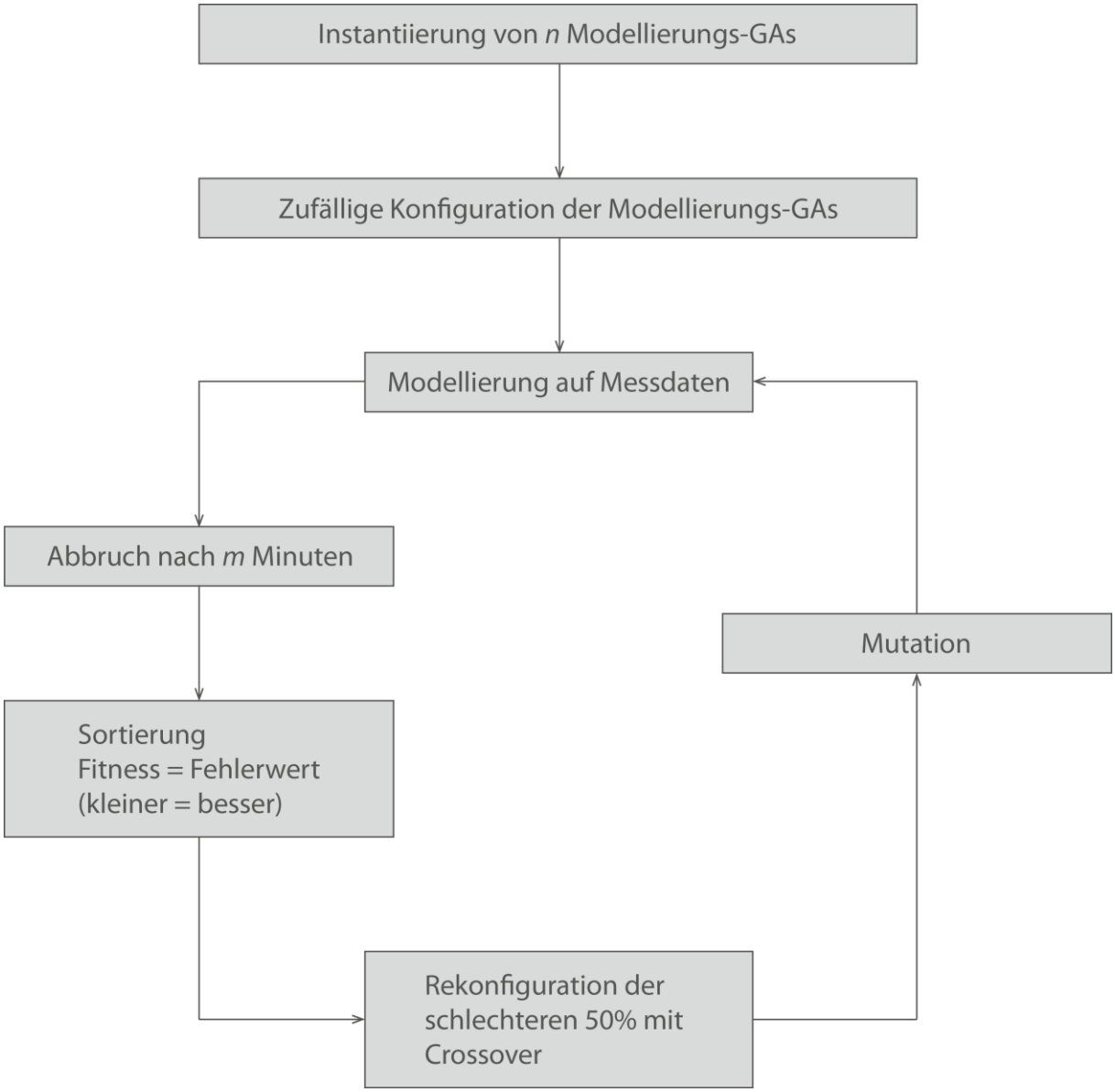


Abbildung 4.34. Ablauf des Meta-GA zur optimalen Konfiguration des Modellierungs-GAs.

4.4.3.1.1.1 Crossover-Operator

Beim Crossover-Operator des Meta-GA werden die einzelnen Konfigurationsparameter des Kind-Individuums zufällig von je einem Eltern-Individuum übernommen. So kann es sein, dass das Kind-Individuum komplett wie das erste oder zweite Eltern-Individuum konfiguriert ist, oder aber wie eine Mischung aus beiden. Die Reihenfolge der Eltern-Individuen (hier erstes und zweites Eltern-Individuum genannt) deutet dabei auf keine Priorisierung hin, beide Eltern-Individuen werden beim Crossover-Operator exakt gleichberechtigt behandelt.

4.4.3.1.1.2 Mutations-Operator

Der Mutations-Operator verändert zufällig einen der Konfigurationsparameter. Anhand einer Zufallszahl wird exakt einer der Konfigurationsparameter ausgewählt. Handelt es sich bei dem ausgewählten Konfigurationsparameter um die Populationsgröße oder die Genom-Länge, so erfolgt durch die Mutation eine Verschiebung des Wertes auf einen der beiden Nachbarwerte (wie in Tabelle 3 beschrieben), bei Randwerten auf den Nachbarn oder auf den entgegengesetzten Randwert. Handelt es sich hingegen um die Mutations- oder Crossoverwahrscheinlichkeit, so erfolgt eine Verschiebung des Werts um einen ganzzahligen Wert zwischen 1 und 10 entweder nach oben oder nach unten. Wird hierbei eine der definierten Grenzwerte überschritten, so erfolgt die Verschiebung in die andere Richtung.

4.4.3.1.1.3 Resultat des Meta-GA

Nachdem die neuen Kind-Individuen mittels des Crossover- und Mutations-Operators konfiguriert wurden erfolgt im Rahmen einer neuen Generation ein weiterer Modellierungslauf aller Individuen. Hierbei beginnen alle Individuen die Modellierung von neuem, ohne Verwendung von Vorwissen.

Dieser Kreislauf aus Konfiguration, Modellierung, Auswertung, Crossover und Mutation erfolgt über den angesetzten Untersuchungszeitraum von 30 Tagen. Hierdurch ergibt sich eine Gesamtzahl von 720 Generationen des Meta-GA.

Aufgrund des Nichtdeterminismus genetischer Algorithmen darf das Resultat des Meta-GA jedoch nicht einfach als die Konfiguration des Individuums angesehen werden, welches in der 720. Generation das beste Ergebnis liefert. Vielmehr erfolgt eine Analyse der besten fünf Konfigurationen der 720. Generation. Die Existenz dieser fünf Konfigurationen nach 720 Generationen ist ein starkes Indiz dafür, dass sie sich als Optimal erweisen könnten. Eine statistische Untersuchung dieser Konfigurationen ist jedoch notwendig, um dieses Indiz zu belegen oder zu widerlegen. Hierzu erfolgen je 50 Modellierungsläufe mit jeder der fünf Konfigurationen auf dem verwendeten Messdatensatz. Die hierbei am besten abschneidende Konfiguration ist die optimale Konfiguration für den Anwendungsfall dieser Arbeit.

4.4.3.1.2 Anpassungen am Modellierungs-Algorithmus

Um die Vergleichbarkeit der Ergebnisse zu gewährleisten wird der Modellierungs-Algorithmus (siehe Teilkapitel 4.2) für die Untersuchung so wenig wie möglich verändert. Die eingesetzte Version des Modellierungs-Algorithmus unterstützt bereits eine Übergabe der

Konfiguration als Kommandozeilenparameter, so dass an dieser Stelle keine Anpassung notwendig ist.

Die Ausführung mehrerer paralleler Modellierungs-Algorithmen erfordert jedoch eine Anpassung der Ausführung der einzelnen Individuen. In der ursprünglichen Version des Modellierungs-Algorithmus werden die Individuen in je einem Thread ausgeführt. Dies ist beim Einsatz des Meta-Algorithmus nicht möglich. Im Schnitt verwendet jeder Modellierungs-Algorithmus 3.350 Individuen (bei der Verwendung der in Teilkapitel 4.4.2 beschriebenen Konfigurationsparameter). Vorab durchgeführte Experimente mit dem Meta-Algorithmus identifizierten eine optimale Populationsgröße des Meta-Algorithmus zwischen 40 und 45 Individuen. Die hieraus resultierende Thread-Anzahl von 134.000 bis 150.750 Threads wird von der Java Virtual Machine nicht unterstützt. Aus diesem Grund ist eine Anpassung des Modellierungs-Algorithmus auf eine Single-Threading-Architektur notwendig, so dass alle Individuen in einem einzigen Thread ausgeführt werden. Diese Anpassung erhöht die Ausführungszeit einer Generation, hat ansonsten jedoch keine Auswirkung auf das Ergebnis.

4.4.3.1.3 Verwendete Fitness-Funktion

In dieser Untersuchung wird der durchschnittliche quadratische Fehlerwert zur Berechnung der Fitness-Funktion verwendet. Je niedriger dieser Fehlerwert ausfällt, desto höher ist die Fitness eines Individuums. Der Vergleich linearer Fehlerwertberechnung mit quadratischer Fehlerwertberechnung ergab eine deutliche Erhöhung der Modellierungsgenauigkeit bei quadratischer Fehlerwertberechnung auf nicht-äquidistanden Datensätzen, wenn von einer geringen Messfehlerwahrscheinlichkeit ausgegangen wird. Eine detaillierte Betrachtung der Fehlerwerte erfolgt in Teilkapitel 4.5.

4.4.3.2 Umsetzung

Die Umsetzung der Untersuchung erfolgt in Form von zwei Java-Projekten. Das Projekt SimpleGA implementiert den Modellierungs-Algorithmus als Singlethreading-Anwendung. SimpleGA ist so konzipiert, dass es (für Testzwecke) als Stand-Alone-Anwendung ausgeführt werden kann, zur Einbindung in den Meta-Algorithmus jedoch auch als Thread-Implementierung.

Das Projekt MetaGA implementiert den Meta-Algorithmus. Meta-GA ist selber ein genetischer Algorithmus, der ein Set von Threads verwaltet, die wiederum durch SimpleGA implementiert sind. MetaGA verwaltet zudem die Ergebnisse der einzelnen SimpleGAs und führt die Generationsfortführung durch.

4.4.3.2.1 SimpleGA

Das Projekt SimpleGA implementiert den Modellierungs-Algorithmus. Der Aufbau entspricht der in Teilkapitel 4.2 vorgestellten *Mendel*-Implementierung, der einzige strukturelle Unterschied liegt in der Implementierung als Singlethreading-Anwendung. Dies ist notwendig, da bis zu 100 SimpleGAs parallel auf der Hardware ausgeführt werden und eine Multithreading-Implementierung des SimpleGAs, wie dies beispielsweise bei *Mendel* der Fall ist, schnell die durch das Betriebssystem gegebenen Obergrenzen für die Thread-Anzahl übersteigt.

Abbildung 4.35 stellt die Klassenstruktur des SimpleGA strukturell als Klassendiagramm dar. Die Struktur des SimpleGA lässt sich in zwei Teile aufteilen. Die Klassen *SimpleGA*, *SimpleGARunner*, *Evaluator* und *Individual* bilden die Kernstruktur des evolutionären Algorithmus, während die Klassen *AbstractGenome*, *OperatorGenome*, *FixedDoubleGenome* und *VariableGenome* sowie die Enumeration *AllowedFuncts* (zusammengefasst im Folgenden unter dem Begriff *Klassenstruktur Genome*) die einzelnen Genome-Elemente und damit die Repräsentation des Modells im evolutionären Algorithmus darstellen.

4.4.3.2.1.1 Klasse *SimpleGA*

Die Klasse *SimpleGA* bildet das Kernelement des SimpleGA. Sie erzeugt, initialisiert und verwaltet die Population an Individuen. Zudem speichert sie das beste Ergebnis und stellt es dem MetaGA zu Verfügung. Die Klasse *SimpleGA* implementiert die Prozesssteuerung, sie bietet Methoden zum Starten und Stoppen der *SimpleGA*-Instanz. Die Klasse *SimpleGA* implementiert zudem eine main-Methode, über welche der SimpleGA als Stand-Alone-Anwendung gestartet wird. Die main-Methode in *SimpleGA* dient ausschließlich dem Test.

4.4.3.2.1.2 Klasse *SimpleGARunner*

Die Klasse *SimpleGARunner* dient als Schnittstelle zwischen dem MetaGA und dem SimpleGA. Der *SimpleGARunner* implementiert Methoden zum Erzeugen und Unterbrechen sowie zum Konfigurieren eines SimpleGA-Threads. Zudem kann der MetaGA über die Klasse *SimpleGARunner* feststellen, ob eine *SimpleGA*-Instanz geendet hat und welches das beste generierte Ergebnis ist.

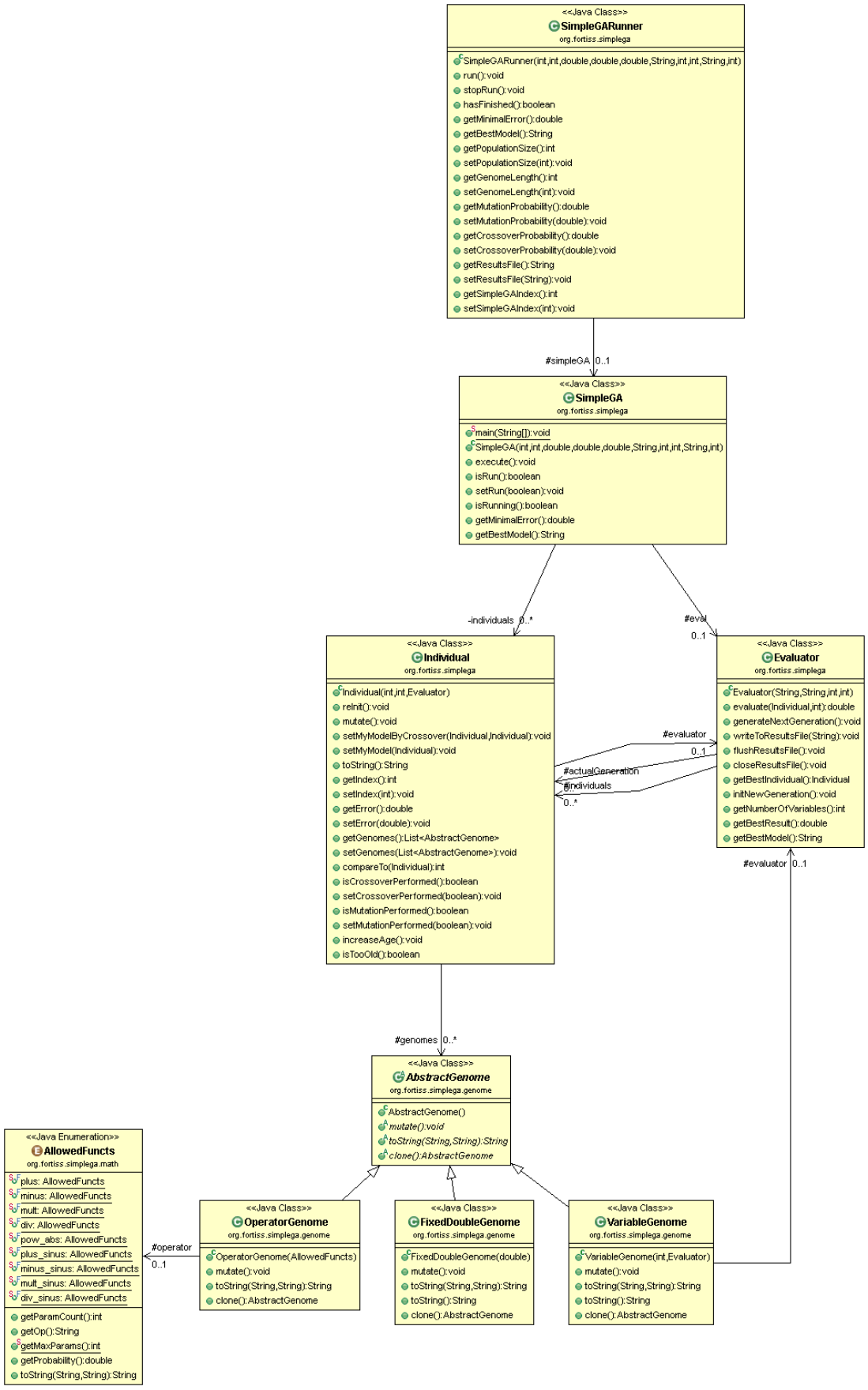


Abbildung 4.35. Klassendiagramm des SimpleGA-Projekts. Das SimpleGA-Projekt bildet die Implementierung des Modellierungs-Algorithmus. Quelle: Eigene Darstellung.

4.4.3.2.1.3 Klasse *Individual*

Die Klasse *Individual* implementiert ein Individuum der Population und die Repräsentation eines generierten Modells. Ein Individual enthält eine Struktur von Genome-Elementen und implementiert Methoden zur Durchführung von Mutation und Crossover, zum Setzen und Auslesen des generierten Modells sowie zum Auslesen des Status des Individuums. Zudem führt ein Individuum ein Alter, das beim Übersteigen eines konfigurierten Wertes zu einer vollständigen Neu-Initialisierung führt.

4.4.3.2.1.4 Klasse *Evaluator*

Die Klasse *Evaluator* führt die Bewertung von durch Individuals generierten Modellen durch. Hierzu liest der *Evaluator* die Messdatendatei aus und berechnet die Fehlerwerte für die Modelle. Auf Basis dieser Bewertungen entscheidet der *Evaluator*, welche Individuals in die nächste Generation übernommen und welche ersetzt werden. Weiterhin stellt der *Evaluator* Informationen über das beste Modell zu Verfügung und schreibt die Informationen zur Auswertung in eine Ergebnisdatei.

4.4.3.2.1.5 Klassenstruktur Genome

Die Darstellung des Modells eines Individuums erfolgt in SimpleGA in Form einer Genome-Struktur (vergleiche für Details Abschnitt 2.4.3.1). Die abstrakte Klasse *AbstractGenome* mit ihren konkreten Unterklassen *OperatorGenome*, *FixedDoubleGenome* und *VariableGenome* stellen die möglichen Genome-Elemente dieser Struktur dar.

AbstractGenome

AbstractGenome bildet die abstrakte Oberklasse aller Genome-Elemente. Die im Individual gespeicherte Liste zur Implementierung der Genome-Struktur ist auf *AbstractGenome* typisiert. *AbstractGenome* deklariert die abstrakten Methoden *mutate*, *toString* und *clone*.

OperatorGenome

Die Klasse *OperatorGenome* implementiert das Verhalten eines mathematischen Operators, im Fall von SimpleGA eines der Operatoren +, -, *, /, ^ oder sin. Mutation erfolgt durch zufälliges Austauschen des Operators. Zur Auswahl des Operators verwendet *OperatorGenome* die Enum *AllowedFuncts*. Diese enthält alle verfügbaren Operatoren sowie einen Wahrscheinlichkeitswert, der angibt, mit welcher Wahrscheinlichkeit ein spezifischer Operator gewählt wird. Die Wahrscheinlichkeit dient zur Priorisierung einzelner Operatoren – so werden beispielsweise die zahlreich vorhandenen Sinus-Operatoren herunterpriorisiert, da die Erfahrung gezeigt hat, dass dies zu deutlich besseren Ergebnissen führt (vgl. (Chawki 2012)).

FixedDoubleGenome

Ein *FixedDoubleGenome* repräsentiert einen fixen Double-Wert, also eine Zahl. Initialisiert wird das *FixedDoubleGenome* durch einen zufälligen Wert zwischen -5 und 5. In der Praxis hat sich dieser Wertebereich bewährt. Mutation erfolgt durch das Verändern des aktuellen Werts um einen Schritt zwischen -5 und 5.

VariableGenome

Ein *VariableGenome* repräsentiert einen Variable im Modell. Bei der Bewertung eines Modells werden die Variablen durch Eingabewerte aus der Messdatendatei belegt. Die Benennung der Variablen erfolgt als X[<VariablenIndex>] mit <VariablenIndex>, beginnend bei 0. Die Anzahl der vorhandenen Variablen wird vom Evaluator zu Verfügung gestellt. Mutation erfolgt durch das zufällige Austauschen des VariablenIndex. Gibt es nur eine Variable, so ist die Mutation des *VariableGenomes* ohne Effekt.

4.4.3.2.2 MetaGA

Das Projekt MetaGA implementiert den Meta-Algorithmus. MetaGA besteht aus den drei Kernklassen *MetaGA*, *Individual* und *Configurator* (siehe Abbildung 4.36). Im Folgenden werden die Klassen vorgestellt.

4.4.3.2.2.1 Klasse *MetaGA*

Die Klasse *MetaGA* stellt den Einstiegspunkt und die Verwaltungsklasse des Meta-Algorithmus dar. In *MetaGA* wird die Messwertliste ausgelesen und gespeichert. Zudem werden hier die Individuen der Population erstellt, bewertet und die Auswahl für Mutation und Crossover getroffen. Die Klasse *MetaGA* erzeugt zudem die Ausgabe.

4.4.3.2.2.2 Klasse *Individual*

Die Klasse *Individual* repräsentiert, ähnlich dem SimpleGA, ein Individuum der Population des MetaGA. Im Projekt MetaGA jedoch stellt ein Individuum einen SimpleGA, also einen vollständigen evolutionären Algorithmus mit einer bestimmten Konfiguration dar. Die Klasse *Individual* bietet dementsprechend Methoden an, um den SimpleGA konfigurieren sowie zu starten und zu stoppen. Zudem bietet sie Methoden für Mutation und Crossover und zum Auslesen des aktuellen Fehlerwerts. Mutation und Crossover erfolgen auf der Konfiguration des SimpleGA: die Mutation erfolgt mittels zufälliger Veränderung eines Konfigurationsparameters unter Verwendung des im Folgenden vorgestellten Configurators, Crossover wird durch das zufällige Mischen der Konfigurationen zweier SimpleGAs erreicht.

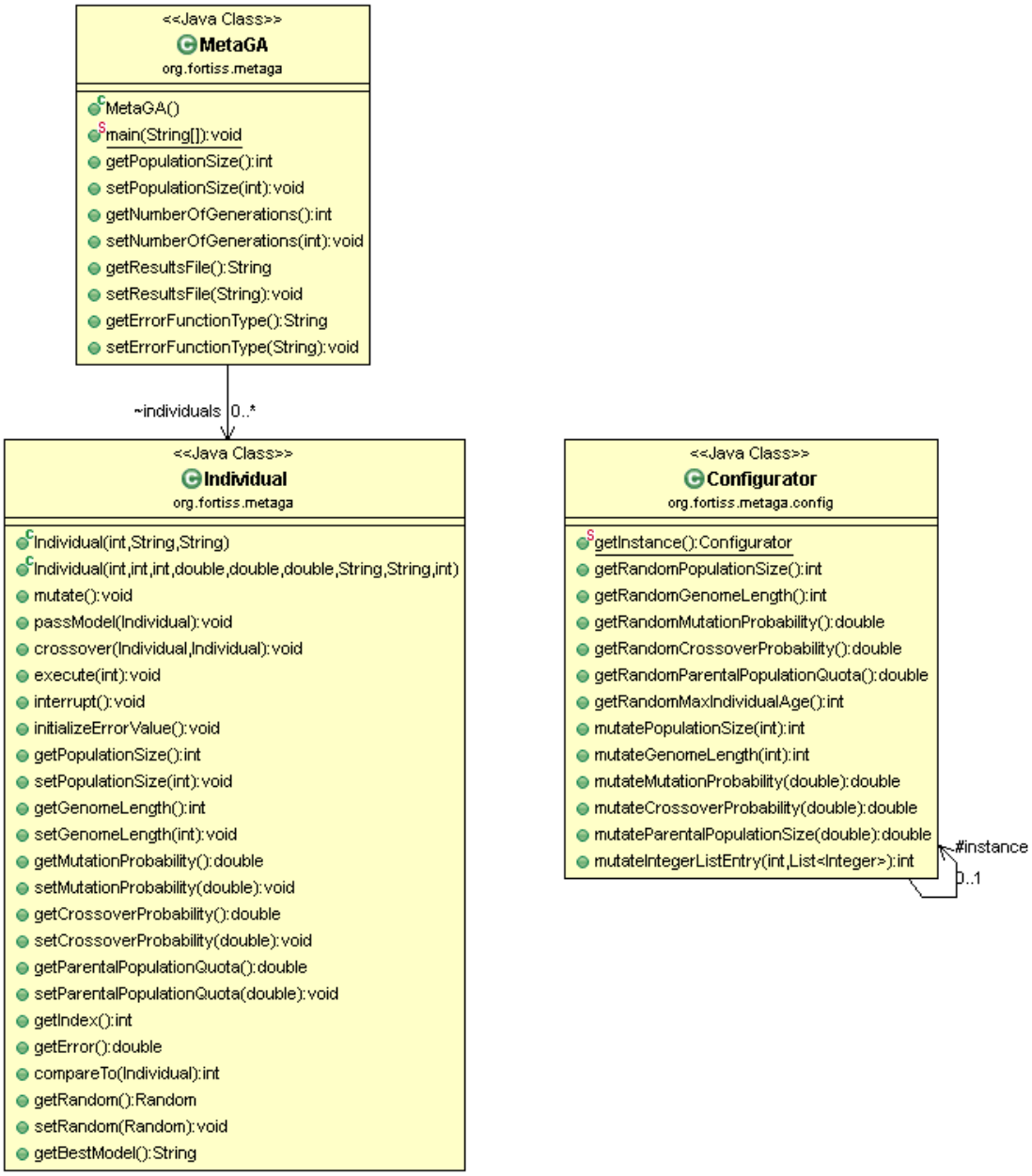


Abbildung 4.36. Klassendiagramm des MetaGA-Projekts. Das MetaGA-Projekt bildet die Implementierung des Meta-Algorithmus. Quelle: Eigene Darstellung.

4.4.3.2.2.3 Klasse *Configurator*

Die Klasse *Configurator* dient zur Generierung zufälliger Konfigurationsparameter für SimpleGA-Instanzen. Zudem implementiert der *Configurator* die Mutationsoperationen für die Konfigurationsparameter. Der Einsatz des *Configurators* ist notwendig, da die Konfigurationsparameter eines SimpleGAs vorgegebenen Kriterien entsprechen müssen. Eine Mutation durch reine Zufallszahl-Addition ist nicht ausreichend. Der *Configurator* ist zustandslos und als Singleton implementiert.

4.4.3.2.3 Zusammenspiel zwischen MetaGA und SimpleGA

Das MetaGA-Projekt nutzt und verwaltet SimpleGA-Instanzen. Jedes Individual der Population des MetaGA entspricht einer SimpleGA-Instanz. Abbildung 4.37 zeigt den Zusammenhang zwischen MetaGA und SimpleGA.

Die Klasse *MetaGA* bildet über die main-Methode den Einstiegspunkt. Sie instanziiert eine Population an Individuen, indem Sie eine Liste von *Individual*-Objekten anlegt. Jedes dieser Individuen erzeugt ein Objekt der Klasse *SimpleGARunner*. Hierdurch entsteht die Verbindung zwischen dem MetaGA und dem SimpleGA-Projekt. Der *SimpleGARunner* wiederum erstellt ein *SimpleGA*-Objekt als Thread und führt dieses aus.

Abbildung 4.37. Verbindung zwischen dem MetaGA und dem SimpleGA. Quelle: Eigene Darstellung.

Durch diese Vorgehensweise entstehen bei einer Populationsgröße von n (im MetaGA) n+1 Threads, je einer für jeden SimpleGA sowie ein Thread für den MetaGA. Unterbricht der MetaGA den Lauf eines SimpleGA (beispielsweise wenn die erlaubte Berechnungszeit überschritten ist), so informiert das Individual-Objekt den SimpleGARunner über den Unterbrechungsbefehl. Der SimpleGARunner wiederum setzt das run-Flag des SimpleGA auf false. Hierdurch erzeugt der SimpleGA keine weitere Generation, sondern beendet den eigenen Thread. Je nach Populationsgröße im SimpleGA kann es auf diese Weise zu einer kurzen Nachlaufzeit eines SimpleGA von wenigen Sekunden kommen. Um Konflikte durch doppelt angestoßene SimpleGAs zu vermeiden synchronisiert sich der MetaGA auf das Ende aller SimpleGAs, bevor er eine neue Generation initialisiert. Der Vorteil dieser Lösung ist jedoch, dass keine Threads während ihres Laufes unterbrochen, sondern alle Threads korrekt beendet werden.

4.4.3.3 Verwendete Hardware

Die Ausführung des Meta-Algorithmus erfolgt auf einem IBM System x3550 M3 mit 2x Intel Xeon 6 Core 2,4 GHz-Prozessoren und 96 GB Arbeitsspeicher. Die zahlreichen Threads des MetaGA erzeugen eine hohe Last auf den CPU, so dass die Entscheidung auf eine native Installation von Ubuntu Linux 12.04 als Minimalsystem gefallen ist. (Bögelsack 2011) zeigt, dass ein natives System im Hochlastbereich deutlich stabiler und besser skaliert als ein virtualisiertes. Auf eine Virtualisierungslösung wurde aus diesem Grund verzichtet. Das System betreibt außer dem Betriebssystem keine weiteren Anwendungen, dem Meta-Algorithmus stehen so die CPU und der Hauptspeicher nahezu vollständig zur Verfügung.

Eine stichprobenartige Betrachtung der Systemauslastung während des Meta-Algorithmus-Laufes ergibt, dass die CPU vollständig ausgelastet arbeiten, während der Arbeitsspeicher zu gut einem Drittel genutzt wird.

4.4.3.4 Ergebnisse

Die Ergebnisse des MetaGA lassen sich auf zwei Arten interpretieren: zum einen ist das finale, global beste Ergebnis relevant, zum anderen der langfristige Trend. Tabelle 4 enthält die zum global besten Ergebnis (Fehlerwert 0,0203) führende Konfiguration.

Diese Konfiguration führt in allen Durchläufen der SimpleGA-Instanzen zu einem Fehlerwert < 3%, im besten Fall zu den genannten 2,03%. Der MetaGA ist jedoch, wie jeder evolutionäre Algorithmus, nichtdeterministisch. Die von den Individuen generierten Konfigurationen unterliegen damit einer kontinuierlichen Veränderung, die Konfigurationen der fitten Individuen „umschwirren“ die optimale Konfiguration. Aus diesem Grund ist zur

Identifikation einer tatsächlichen optimalen Konfiguration die Betrachtung der Trends unerlässlich. Ebendiese Betrachtung erfolgt in den folgenden Abschnitten.

Tabelle 4. Durch den MetaGA ermittelte optimale Konfiguration des Modellierungs-Algorithmus.

Konfigurationsparameter	**Wert**
Populationsgröße	10.000
Genome-Länge	31
Mutations-Wahrscheinlichkeit	82%
Crossover-Wahrscheinlichkeit	91%
Parental Population Quota	34%

4.4.3.4.1 Populationsgröße

Die Populationsgröße hat einen großen Einfluss auf die Diversifikation innerhalb der Population. Eine große Population führt zu einem breiten Spektrum an Varianten innerhalb einer Generation, reduziert jedoch auch die Wahrscheinlichkeit, dass sich ein gutes Genome weiter vererbt. Allgemein betrachtet erhöht eine größere Population die Chance auf das Auffinden eines Optimums, während eine kleinere Population das Konvergenzverhalten verbessert. Aus technischer Sicht wirkt sich die Populationsgröße auf die Auslastung der zu Verfügung stehenden Rechenressourcen, jedoch auch auf den Verwaltungsoverhead aus.

Abbildung 4.38 zeigt die Häufigkeits- und Wahrscheinlichkeitsverteilung des Auftretens einer Populationsgröße in den Ergebnissen des Meta-GA. Mit nahezu 60 Prozent Wahrscheinlichkeit dominiert die Klasse *Populationsgröße 10.000* gefolgt von der *Populationsgröße 15.000*. Eine Analyse des zeitlichen Ablaufs des Meta-GA ergibt zudem, dass die Häufigkeit der Vorkommnisse der Klasse 10.000 im späteren Verlauf des Algorithmus zunimmt. Es ist damit eindeutig, dass eine optimale Konfiguration die Populationsgröße 10.000 enthält.

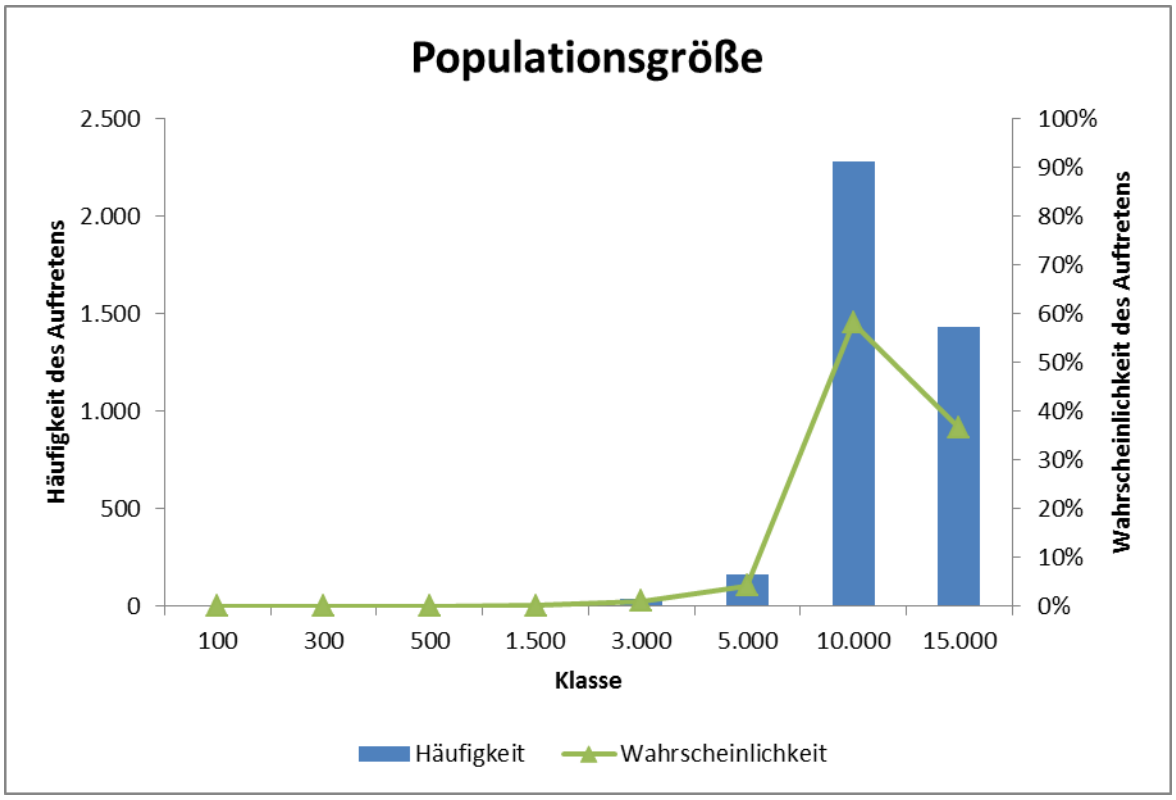


Abbildung 4.38. Ermittelte Häufung und Wahrscheinlichkeitsverteilung der Populationsgröße, ermittelt durch den Meta-GA. Quelle: Eigene Darstellung.

4.4.3.4.2 Genome-Länge

Die Genome-Länge bestimmt die Komplexität, die das Modell eines Individuums annehmen kann. Die Genome-Länge ist fest definiert, so dass jedes Genome und damit jedes generierte mathematische Modell diese exakt definierte Länge enthält. Für das mathematische Modell bedeutet dies, dass es exakt $\frac{n-1}{2}$ Operationen besitzt (n entspricht der Genome-Länge).

Wählt man das Genom zu kurz, so besitzt es bei komplexen Datensätzen nicht ausreichend Flexibilität, um die Datenstrukturen abzubilden. Wird das Genom jedoch zu lang gewählt, so verlieren die Mutations- und Crossover-Operatoren einen Teil ihrer Wirkung. Weiterhin steigt die Berechnungszeit pro Generation und damit die Laufzeit des evolutionären Algorithmus stark an.

Abbildung 4.39 zeigt die Häufigkeit und die Wahrscheinlichkeit des Auftretens von Genomen einer Länge. Die Klasse der Genome mit einer Länge von 41 Elementen ist mit 40% Wahrscheinlichkeit dominant, gefolgt von der Klasse mit Genome-Länge 31. Andere Genome-Längen finden kaum Beachtung. Kürzere Genome bilden die Dimensionen und Schwankungen der Messwerte nicht ausreichend detailliert ab, längere Genome hingegen

führen zu einem Overhead, der nicht in Relation zu dem gewonnenen Detaillierungs-Vorteil steht. Eine optimale Konfiguration des Modellierungs-Algorithmus enthält demnach eine Genome-Länge von 41.

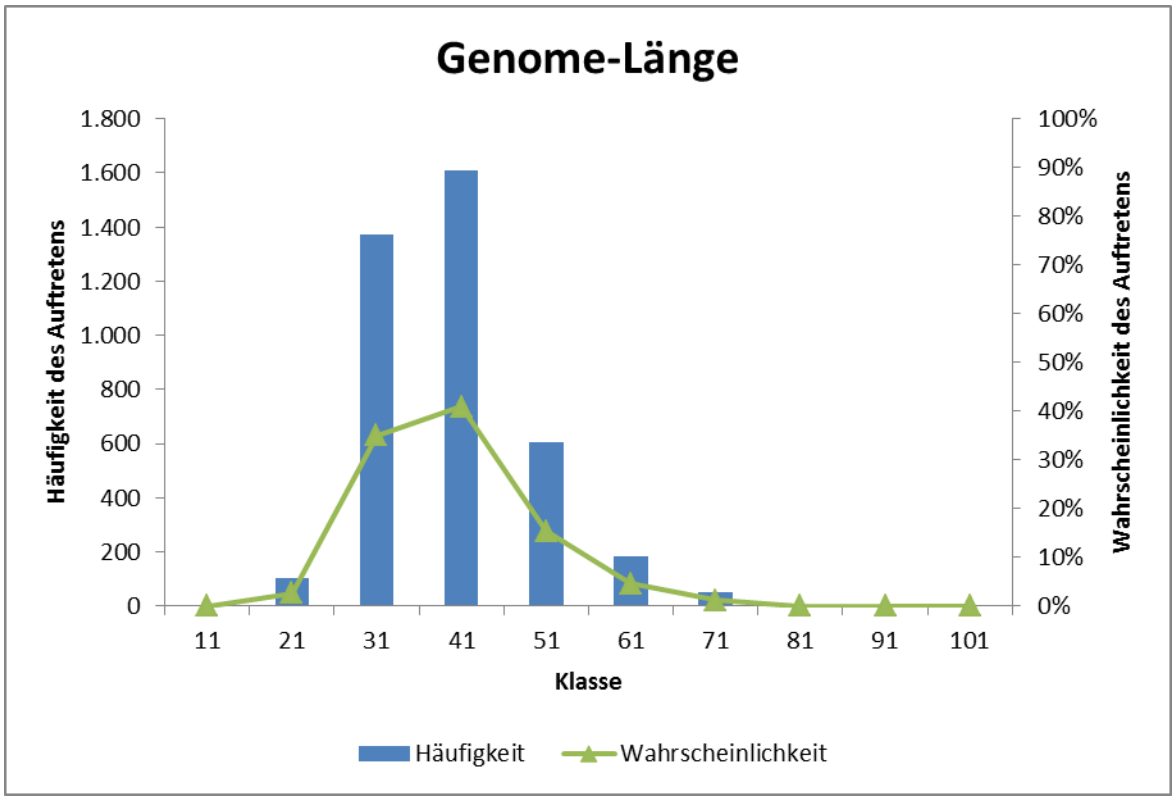


Abbildung 4.39. Ermittelte Häufung und Wahrscheinlichkeitsverteilung der Genome-Länge, ermittelt durch den Meta-GA. Quelle: Eigene Darstellung.

4.4.3.4.3 Crossover-Wahrscheinlichkeit

Die Crossover-Wahrscheinlichkeit gibt an, mit welcher Wahrscheinlichkeit bei einer Vererbungs-Operation die Crossover-Operation angewandt wird. Ein verstärktes Anwenden der Crossover-Operation führt zu einer breiteren Suche, verringert jedoch das Konvergenzverhalten des evolutionären Algorithmus (vgl. Abschnitt 2.4.1.2.2).

Abbildung 4.40 zeigt die Häufigkeit und die Wahrscheinlichkeitsverteilung der Crossover-Wahrscheinlichkeiten in den Ergebnissen des Meta-GA. Es lassen sich zwei Spitzen erkennen: eine Spitze von gut 30 Prozent um die Klasse 75, sowie eine weitere von gut 15 Prozent um die Klasse 95. Summiert man jedoch die Wahrscheinlichkeiten um die 75-Prozent-Spitze, so ergibt sich eine Häufung von gut 70 Prozent im Bereich der Klassen 65 bis 80. Für die optimale Konfiguration bietet sich damit die Crossover-Wahrscheinlichkeit von 75% an.

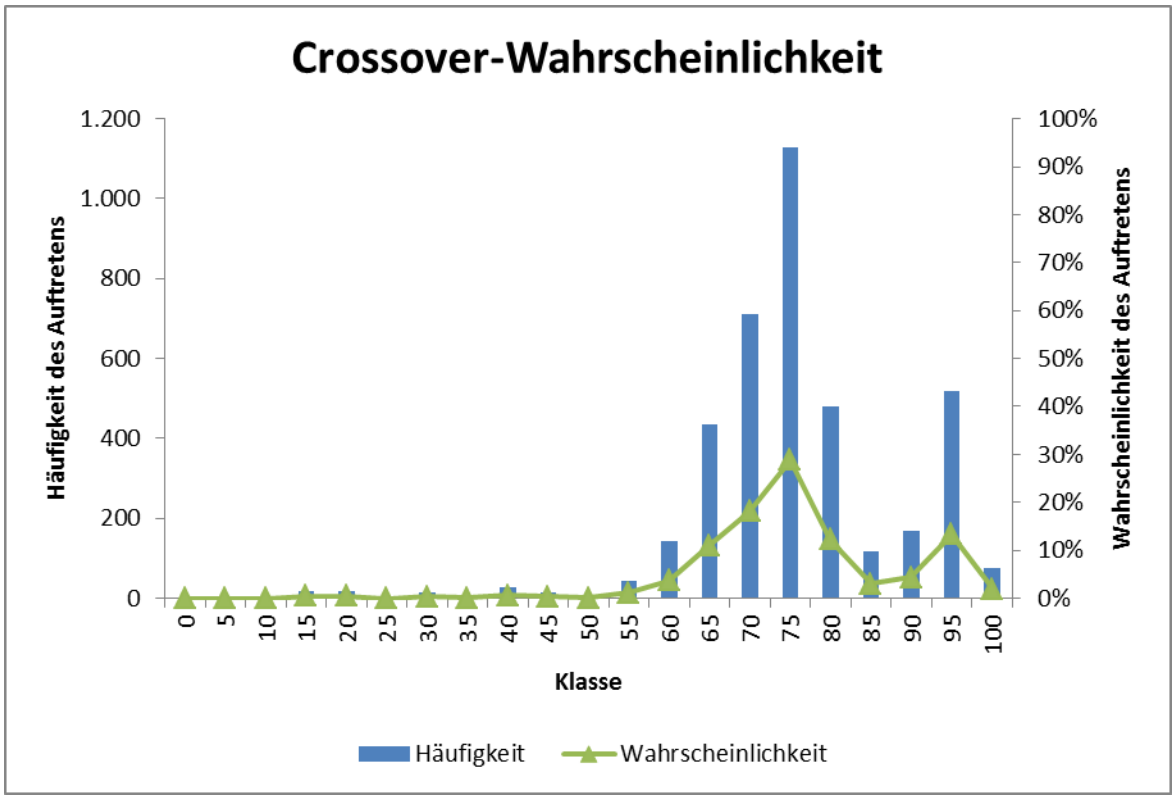


Abbildung 4.40. Ermittelte Häufung und Wahrscheinlichkeitsverteilung der Crossover-Wahrscheinlichkeit, ermittelt durch den Meta-GA. Quelle: Eigene Darstellung.

4.4.3.4.4 Mutations-Wahrscheinlichkeit

Die Mutations-Wahrscheinlichkeit gibt, synchron zur Crossover-Wahrscheinlichkeit, an, mit welcher Wahrscheinlichkeit bei der Vererbungs-Operation eine Mutation durchgeführt wird. Die Mutation führt zufällige Einzelveränderungen an den generierten Modellen durch und verursacht dadurch das Konvergieren hin zu einem Optimum (vgl. Abschnitt 2.4.1.2.3).

Das Ergebnis des Meta-GA hinsichtlich der Mutationswahrscheinlichkeit ist sehr eindeutig. Aus Abbildung 4.41 geht klar hervor, dass eine Mutations-Wahrscheinlichkeit von 85 bis 95 Prozent zu den von den Individuen des Meta-GA bevorzugten Konfigurationen zählt. Die Flanke bis hinab zu der Klasse 75 Prozent entsteht durch die kontinuierlichen Mutationen im Meta-GA, welche die Individuen zwingen, Alternativen bei der Wahl des Konfigurationsparameters *Mutations-Wahrscheinlichkeit* auszuprobieren. Das kontinuierliche Abfallen dieser Flanke zeugt jedoch davon, dass alle Individuen mit einer Mutations-Wahrscheinlichkeit unter 90 Prozent im Rahmen des Generationenwechsels aussortiert werden. Eine Analyse der Einzelergebnisse bestätigt dies.

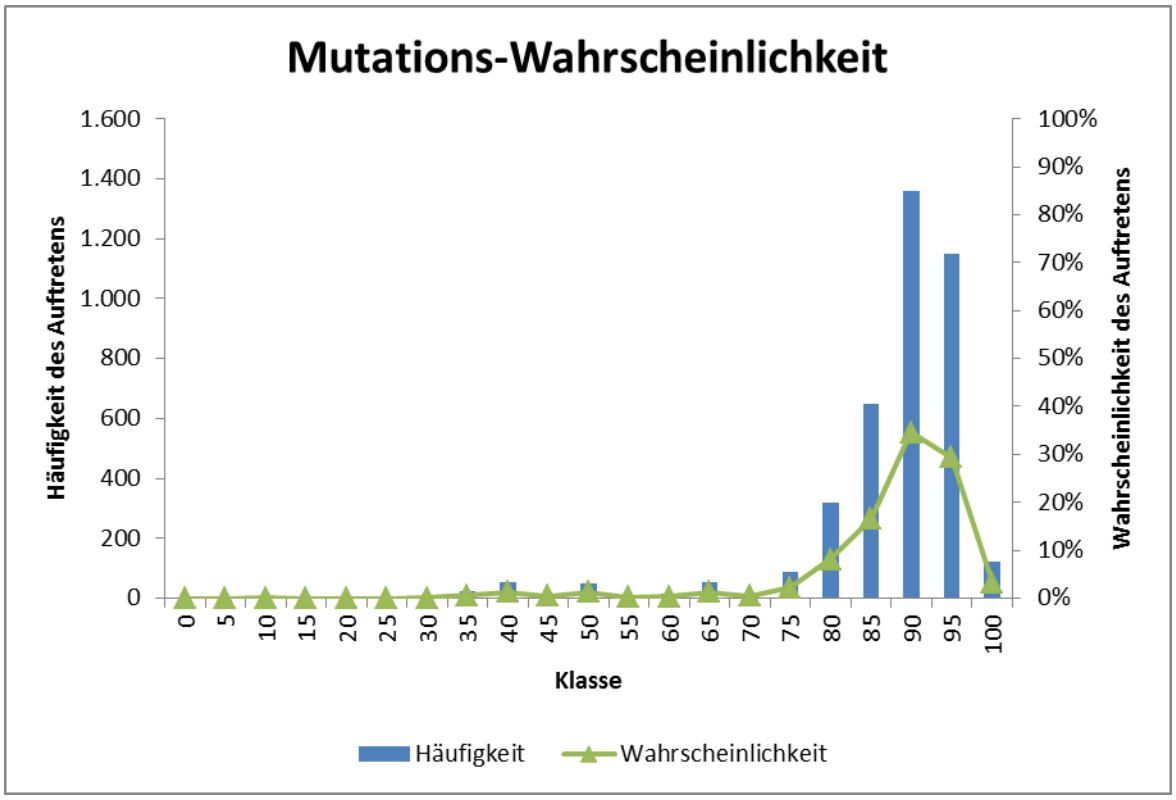


Abbildung 4.41. Ermittelte Häufung und Wahrscheinlichkeitsverteilung der Mutations-Wahrscheinlichkeit, ermittelt durch den Meta-GA. Quelle: Eigene Darstellung.

4.4.3.4.5 Parental Population Quota

Die Parental Population Quota gibt an, welcher Prozentsatz einer Population den Übergang zur nächsten Generation überlebt und dort als Eltern-Individuen zur Erzeugung der neuen Individuen fungiert. Der Konfigurationsparameter beeinflusst damit die Konstanz in der Population auf zwei Weisen.

Ist die Parental Population Quota niedrig, so entsteht eine hohe Anzahl an neuen Modellen, da viele Individuen ersetzt werden. Gleichzeitig stammen diese vielen neuen Individuen jedoch von wenigen Individuen der Vor-Generation ab, so dass die Diversifikation eher gering ist.

Ist die Parental Population Quota jedoch hoch, so werden pro Generation wenige neue Individuen erschaffen. Die Veränderungsrate ist gering. Die wenigen neuen Individuen stammen jedoch mit einer hohen Wahrscheinlichkeit von einer großen Variation von Eltern-Individuen, ihre Mischung ist damit hoch und die Chance, dass ein Individuum einen neuen, besseren Modellstamm erzeugt (und damit ein besseres lokales oder sogar globales Optimum findet) ist höher.

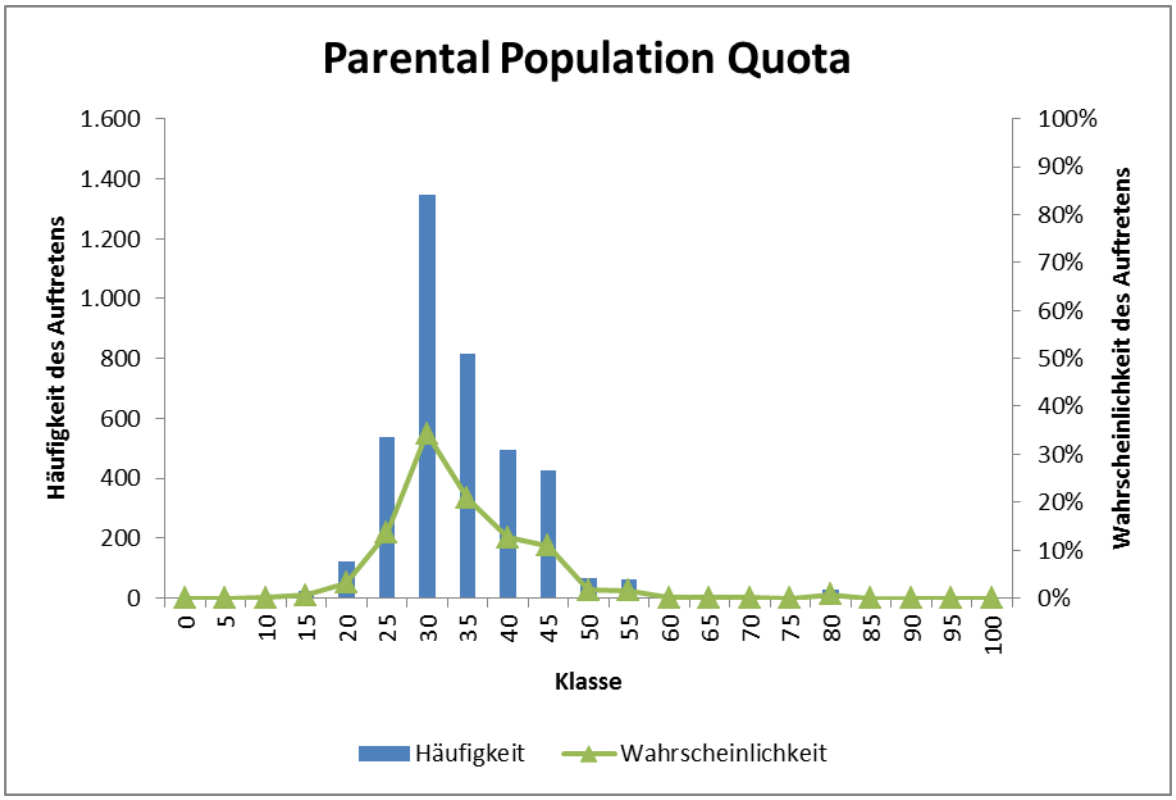


Abbildung 4.42. Ermittelte Häufung und Wahrscheinlichkeitsverteilung der Parental Population Quota, ermittelt durch den Meta-GA. Quelle: Eigene Darstellung.

Abbildung 4.42 zeigt die Verteilung der Parental Population Quota über die Ergebnisse des Meta-GA. Eine breitere Streuung ist sichtbar im Vergleich zu den vorab vorgestellten Konfigurationsparametern. Die relevanten Klassen mit einer Wahrscheinlichkeit größer 10 Prozent erstrecken sich von 25 bis 45 Prozent Parental Population Quota. Die Klassen 30 und 35 bilden die einzigen Elemente mit einer Auftrittswahrscheinlichkeit größer 20 Prozent. Dennoch bleibt festzuhalten, dass der Konfigurationsparameter Parental Population Quota in einem breiteren Spektrum gute Ergebnisse erzielen lässt.

4.4.3.5 Approximationsverhalten

Ein evolutionärer Algorithmus basiert auf Zufallsveränderungen und ist damit nichtdeterministisch. Ein Erfolg der Modellierung ist damit nicht per se gegeben. Um den vorgestellten Ansatz dennoch für den Anwendungsfall nutzen zu können ist es wichtig, dass die Wahrscheinlichkeit, dass der Algorithmus zu einem akzeptierten Modell gelangt, hoch ist.

In diesem Abschnitt wird das Approximationsverhalten der Durchläufe der MetaGA-Individuen untersucht. Hierzu werden die optimalen Modelle aller im Zuge der MetaGA-Durchführung generierten Individuen, also der einzelnen SimpleGA-Instanzen, auf ihr Laufzeitverhalten analysiert. Hierbei erfolgt zum einen eine Betrachtung des allgemeinen

Approximationsverhaltens, zum anderen eine Betrachtung des Einflusses einzelner Konfigurationsparameter auf dieses.

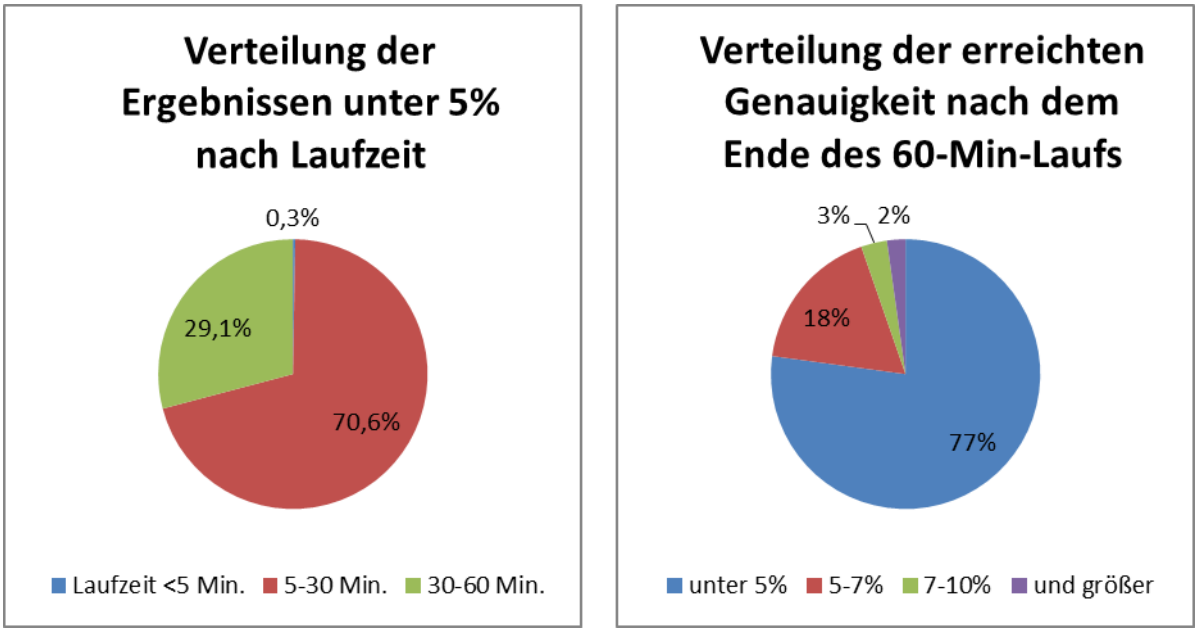


Abbildung 4.43. Links: Laufzeitverteilung, die die Individuen bis zum Erreichen eines Modells mit Fehlerwert < 5% benötigten. Rechts: Verteilung des erreichten Fehlerwerts jedes einzelnen Individuums nach 60 Minuten. Quelle: Eigene Darstellungen.

Die linke Grafik aus Abbildung 4.43 zeigt, wie lange die einzelnen Instanzen benötigten, um ein Modell mit einem Fehlerwert kleiner 5% zu erzielen. Aus der Grafik wird deutlich, dass ein Großteil der akzeptablen Modelle innerhalb der ersten 30 Minuten erzeugt werden. Über 70 Prozent aller Modelle mit akzeptablem Fehlerwert werden innerhalb dieser Zeitspanne erzeugt. Die rechte Grafik in Abbildung 4.43 stellt den minimalen Fehlerwert aller Individuen des MetaGA nach der Maximallaufzeit von 60 Minuten dar. 77 Prozent aller Individuen des MetaGA erzeugen innerhalb dieses Zeitrahmens ein Modell mit einem Fehlerwert unter 5 Prozent, weitere 18 Prozent erreichen einen Fehlerwert von unter 7 Prozent. Nur 5 Prozent aller Individuen erreichen schlechtere Ergebnisse.

Zusammengefasst bedeuten diese Zahlen, dass 77 Prozent aller Individuen innerhalb von 60 Minuten zu einem verwendbaren Modell gelangen, 54 Prozent sogar bereits innerhalb einer halben Stunde (70,6 Prozent von 77 Prozentpunkten). Setzt man demnach zwei parallele Modellierungsläufe auf einen Datensatz an, so ist die Wahrscheinlichkeit groß, dass innerhalb einer halben Stunde ein gutes Modell generiert wird.

Einzelne Konfigurationsparameter haben einen Einfluss auf die Laufzeit und das Approximationsverhalten der Individuen. Im Folgenden wird dieser Einfluss dargestellt.

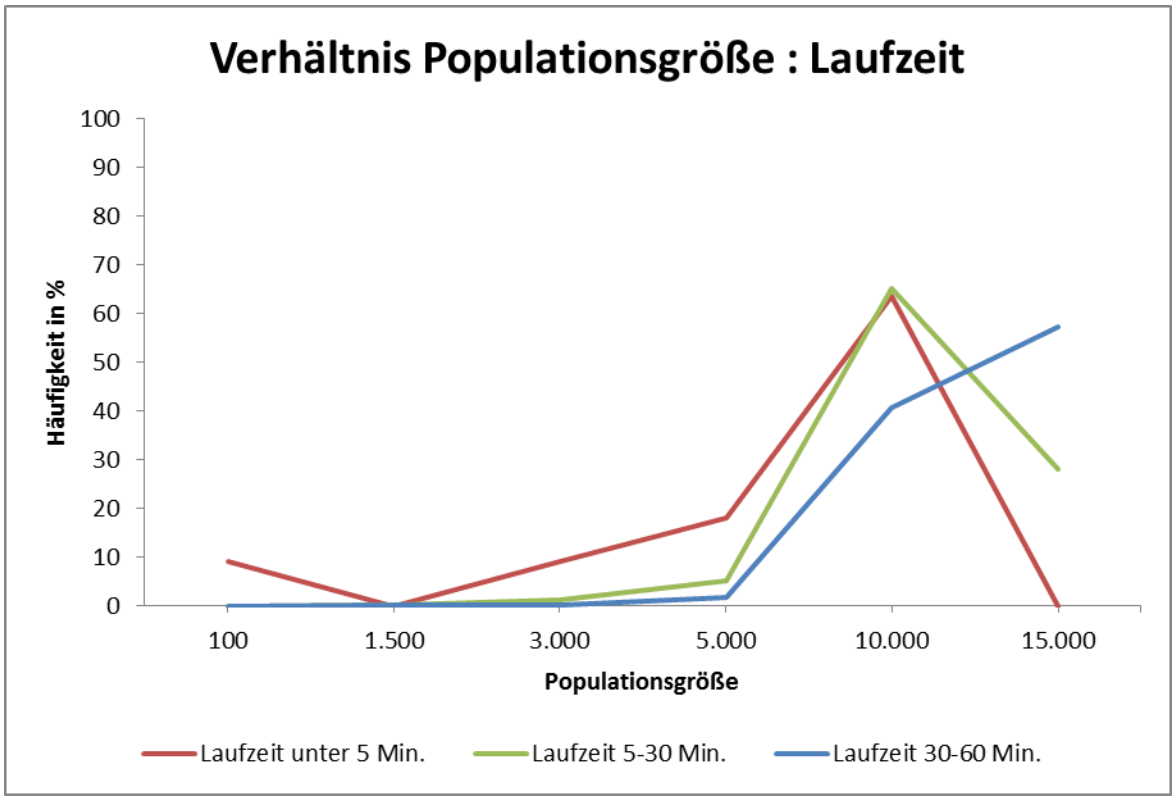


Abbildung 4.44. Einfluss des Konfigurationsparameters "Populationsgröße" auf das Approximationsverhalten. Quelle: Eigene Darstellung.

Abbildung 4.44 zeigt die prozentuale Häufigkeit des Auftretens einzelner Populationsgrößen in den Laufzeitklassen 5 Minuten, 5-30 Minuten sowie 30-60 Minuten mit Modellen mit einem Fehlerwert kleiner 5 Prozent. Aus der Abbildung wird ersichtlich, dass etwa 70 Prozent aller Individuen, die in unter 5 Minuten ein Modell mit einem Fehlerwert kleiner 5 Prozent generierten, eine Populationsgröße von 10.000 hatten. Anders herum finden 60 Prozent aller erfolgreichen Individuen mit einer Populationsgröße 15.000 erst innerhalb des Intervalls 30-60 Minuten ein entsprechendes Modell. Zusammengefasst geht aus der Grafik hervor, dass Individuen mit großer Population generell länger brauchen um ein Modell mit Fehlerwert kleiner 5 Prozent zu generieren. Anders herum zeigt jedoch Abschnitt 0, dass größere Populationen deutlich zuverlässiger gute Modelle generieren.

Eine ähnliche Analyse ergibt sich aus Abbildung 4.45 für die Genome-Länge. Erfolgreiche Individuen mit kurzer Genome-Länge werden im Schnitt deutlich früher fündig als Individuen mit längeren Genomen. Dies gilt jedoch nur innerhalb des Intervalls [21,41]. Danach fallen die Kurven steil ab, passend zu den Ergebnissen aus Abschnitt 4.4.3.4.2.

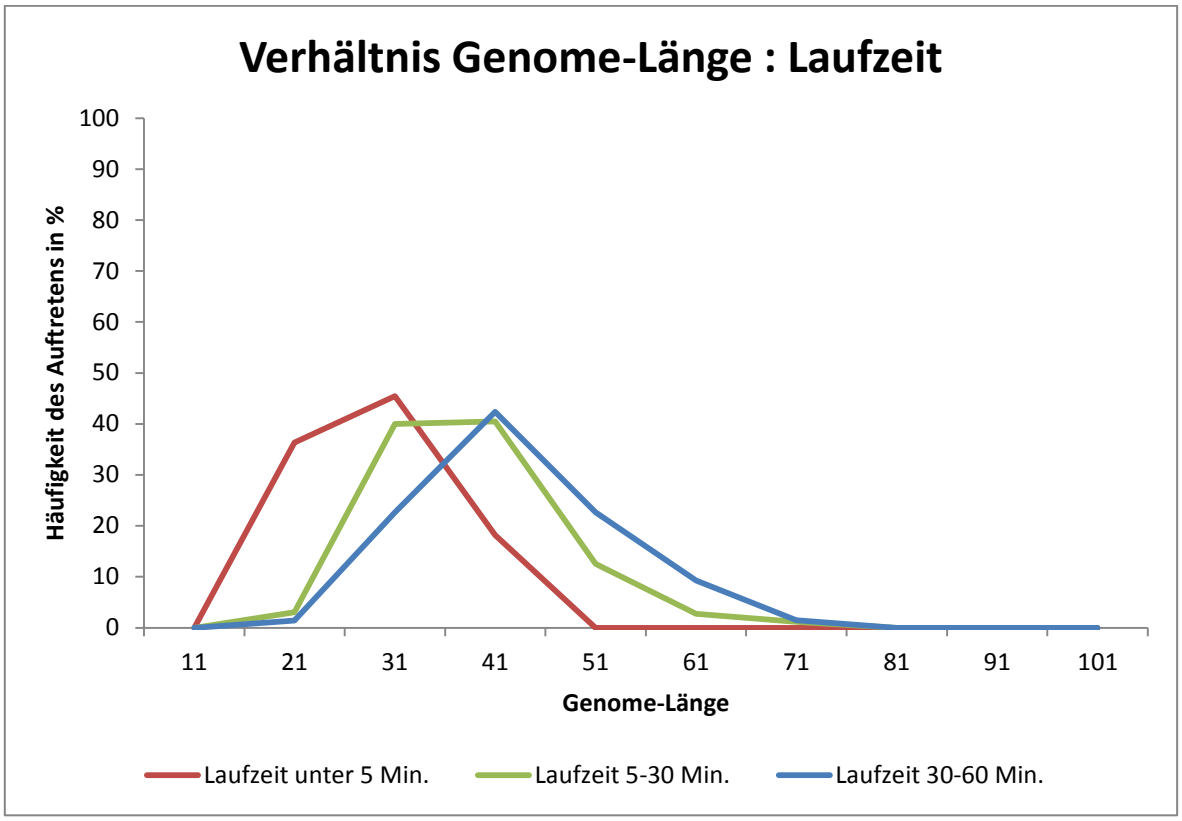


Abbildung 4.45. Einfluss des Konfigurationsparameters "Genome-Länge" auf das Approximationsverhalten. Quelle: Eigene Darstellung.

Als Ursache für die letztgenannten Ergebnisse liegt die erhöhte Berechnungszeit eines Individuums bei einem längeren Genom, beziehungsweise die erhöhte Berechnungszeit einer Generation bei einer großen Population nahe. Hierdurch werden innerhalb einer Zeiteinheit weniger Generationen durchgerechnet, was zu einem reduzierten Konvergenzverhalten und damit zu einer verlängerten Laufzeit führt. Gleichzeitig erhöht sich die Diversifikation, resultierend in einer höheren Erfolgswahrscheinlichkeit.

Für die anderen Konfigurationsparameter ist kein Effekt auf das Approximationsverhalten identifizierbar.

4.4.4 Zusammenfassung

Die Anwendung des evolutionären Algorithmus *Meta-GA* zur Generierung einer optimalen Konfiguration für den Modellierungs-Algorithmus liefert eine nutzbare und nahezu eindeutige Konfiguration. Mithilfe dieser Konfiguration ist es dem Modellierungs-Algorithmus möglich, innerhalb der vorgegebenen Zeit ein Modell der komplexen, mehrdimensionalen Messdaten zu generieren, das mit einem Fehlerwert unter 2,1 Prozent deutlich unter dem erwarteten Messfehler von fünf Prozent liegt.

Das vom Meta-GA generierte Ergebnis ist in der Aussage mit der statistisch ermittelten optimalen Konfiguration des Modellierungs-Algorithmus vereinbar. Die Abweichung von wenigen Prozentpunkten bei der Mutations- und Crossover-Wahrscheinlichkeit lässt sich auf den Nichtdeterminismus evolutionärer Algorithmen zurückführen und fällt bei der Modellierung nicht ins Gewicht. Die größere Abweichung bei der Populationsgröße ergibt sich durch die unterschiedliche Implementierung von *Mendel* und *SimpleGA*. Während *Mendel* als Multithreading-Anwendung konzipiert ist arbeitet der SimpleGA im Singlethreading-Modus. Bei *Mendel* ist jedes Individuum ein eigener Thread, wodurch bei einer großen Population zusätzlicher Overhead entsteht.

Da im Anwendungsfall (siehe Kapitel 5) *Mendel* zum Einsatz kommt orientiert sich die optimale Konfiguration der Populationsgröße an den statistischen Ergebnissen. Bei den anderen Konfigurationsparametern ergibt sich eine Deckung. Die statistischen Ergebnisse machen hinsichtlich der Mutations- und Crossover-Wahrscheinlichkeit keine eindeutige Aussage, so dass hier die deutlich konkreteren Ergebnisse des MetaGA verwendet werden. Die so gewonnene Konfiguration bildet die optimale Konfiguration für den Einsatz des evolutionären Algorithmus in der Experimentdurchführung.

Die Betrachtung des Approximationsverhaltens der MetaGA-Individuen ergibt, dass der evolutionäre Algorithmus trotz seines Nichtdeterminismus ein zuverlässiger Ansatz zur Modellierung ist. In zwei Dritteln aller Modellierungsläufe generiert der evolutionäre Algorithmus ein Modell mit einem Fehlerwert kleiner 5 Prozent, ein Fehlerwert kleiner 7 Prozent wird sogar in 95 Prozent aller Fälle erreicht. Hierbei haben die Konfigurationsparameter *Populationsgröße* und *Genome-Länge* Einfluss auf das Approximationsverhalten: je größer die Population und je länger das Genome, desto länger braucht der Algorithmus für die Approximation. Gleichzeitig steigt jedoch auch die Wahrscheinlichkeit, dass tatsächlich ein verwendbares Modell gefunden wird. Für die anderen Konfigurationsparameter ist ein solcher Einfluss nicht identifizierbar.

Neben den beschriebenen und untersuchten Konfigurationsparametern bildet die Art der Fehlerwertberechnung ein weiteres veränderliches Element der Konfiguration eines evolutionären Algorithmus. Die Varianten der Fehlerwertberechnung beeinflussend grundlegend das Verhalten und das Ergebnis eines evolutionären Algorithmus, und sind stark abhängig von der Art des gestellten Problems. Der Fehlerwertberechnung ist das folgende Teilkapitel gewidmet.

4.5 Fehlerwert

Die Abweichung eines Modells von dem tatsächlichen Performance-Verhalten einer Softwarekomponente wird durch den Fehlerwert beschrieben. In der Literatur wird diese Abweichung meist durch die durchschnittliche geometrische Distanz des Modells von den Messwerten definiert (Koza 1992a; Tertilt et al. 2010). Je nach Ziel der Performance-Analyse ist die durchschnittliche Distanz zwischen gemessenem und simuliertem Performance-Verhalten jedoch nicht zielführend. So führt bei der Durchschnittsbildung eine sehr exakte Modellierung im Niedrig- und Mittellastbereich zu einem geringen Fehlerwert, selbst wenn im Hochlastbereich das Modell stark vom tatsächlichen Verhalten der Komponente abweicht. Ist das Ziel der Performance-Analyse nun die Schwachstellenerkennung im Hochlastbetrieb, so kann ein Modell mit einem geringen herkömmlichen Fehlerwert zu gravierenden Fehlern in der Simulation des Gesamtsystems führen.

Im den folgenden Teilkapiteln werden verschiedene Verfahren zur Fehlerwertbestimmung vorgestellt und ihre Vor- und Nachteile aufgeführt. Anschließend werden sie miteinander verglichen und das für die vorliegende Arbeit optimale Verfahren ausgewählt. Zudem wird ein entwickeltes Verfahren zur Berechnung eines komplexen Fehlerwerts vorgestellt, welches die Nachteile der existierenden Fehlerwertberechnungen kompensiert.

4.5.1 Linearer Fehlerwert

Der lineare Fehlerwert bildet die intuitivste Art der Fehlerwertberechnung. Er ergibt sich aus der durchschnittlichen geometrischen Distanz zwischen den gemessenen Werten und den modellierten Werten. Formel 10 definiert die Berechnung des Modellfehlerwerts s_{err}. n bestimmt dabei die Anzahl der Messpunkte, die zur Modellierung genutzt werden. $r_{measured_i}$ definiert den Messwert des i-ten Messpunkts, $r_{modeled_i}$ entsprechend den Modellwert des i-ten Messpunkts.

Formel 10. Linearer Fehlerwert.

$$s_{err} = \frac{\sum_{i=1}^{n} \frac{|r_{measured_i} - r_{modeled_i}|}{r_{measured_i}}}{n}$$

Der lineare Fehlerwert ergibt sich damit aus dem durchschnittlichen absoluten Abstand zwischen dem Modell und den Messwerten. Die Betrachtung des absoluten Abstands ist notwendig, damit sich Modellwerte, die über den Messwerten liegen, nicht mit Modellwerten, die unter den Messwerten liegen, negieren.

Abbildung 4.46 illustriert den linearen Fehlerwert.

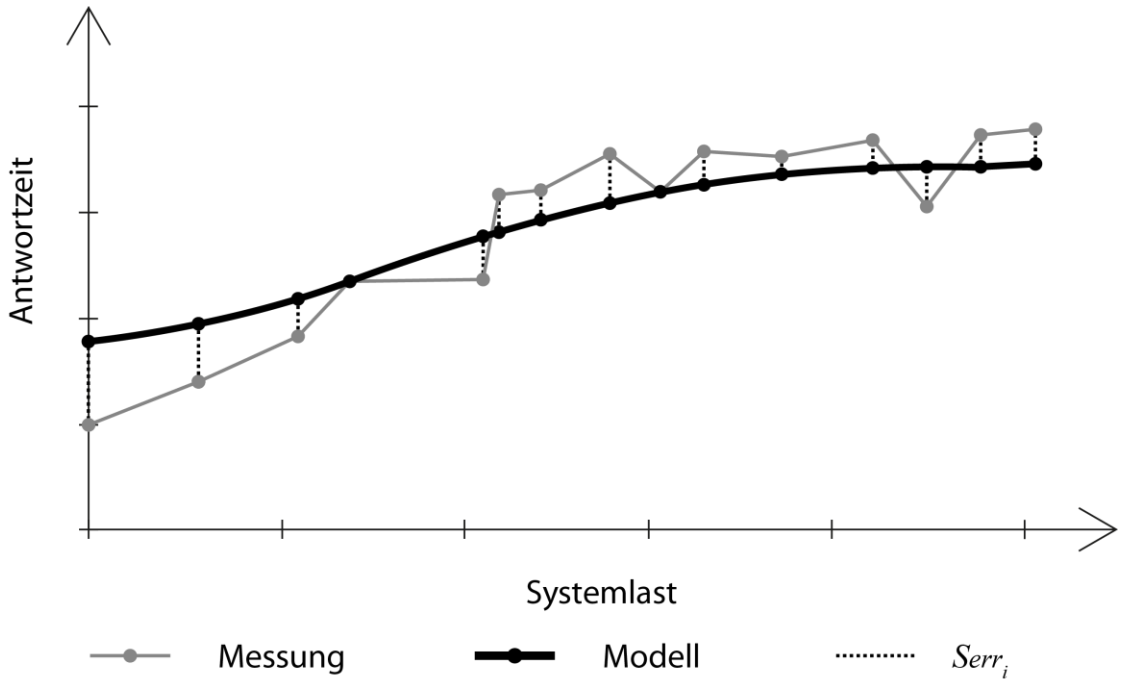


Abbildung 4.46. Grafische Darstellung des linearen Fehlerwerts.

Der lineare Fehlerwert erlaubt die einfachste Fehlerwertberechnung und liefert eine intuitiv verständliche Aussage über die Qualität eines Modells. Neben diesen Vorteilen bringt der lineare Fehlerwert jedoch auch einige Nachteile mit sich. Sind die zu modellierenden Messwerte nicht gleichverteilt, so erlaubt es der lineare Fehlerwert, dass wenige Messwerte bei der Modellierung ignoriert werden (und bei diesen damit ein hoher Fehlerwert bestehen bleibt), solange die Mehrheit der Messwerte möglichst exakt modelliert wird. Existieren für eine Softwarekomponente beispielsweise große Mengen an Messwerten aus dem Niedriglastbereich, jedoch nur wenige Messwerte aus dem Hochlastbereich, so erlaubt der lineare Fehlerwert das Ignorieren der Hochlast-Messwerte zugunsten eines passenden Modells der vielen Niedriglast-Messwerte – ein Verhalten, welches das Modell unbrauchbar macht. Abbildung 4.47 verdeutlicht dieses Problem.

In der Abbildung wurde ein (sehr einfaches, lineares) Modell gewählt, welches die zahlreichen Messwerte im Niedriglastbereich gut abbildet. Die wenigen Messwerte im Hochlastbereich hingegen liegen weit vom Modell entfernt. Sie fließen mit einem hohen Fehlerwert in die Berechnung des Gesamtfehlerwerts ein – da die Berechnung jedoch linear erfolgt und der Durchschnitt als Gesamtfehlerwert berechnet wird gleichen die zahlreichen geringen Fehlerwerte die wenigen hohen Fehlerwerte gut aus. Erfolgt jedoch mithilfe des auf diese Weise erstellten Modells eine Betrachtung eines Systems mit der modellierten

Komponente im Hochlastbereich, so liegt das Modell weit vom tatsächlichen Verhalten der Komponente entfernt.

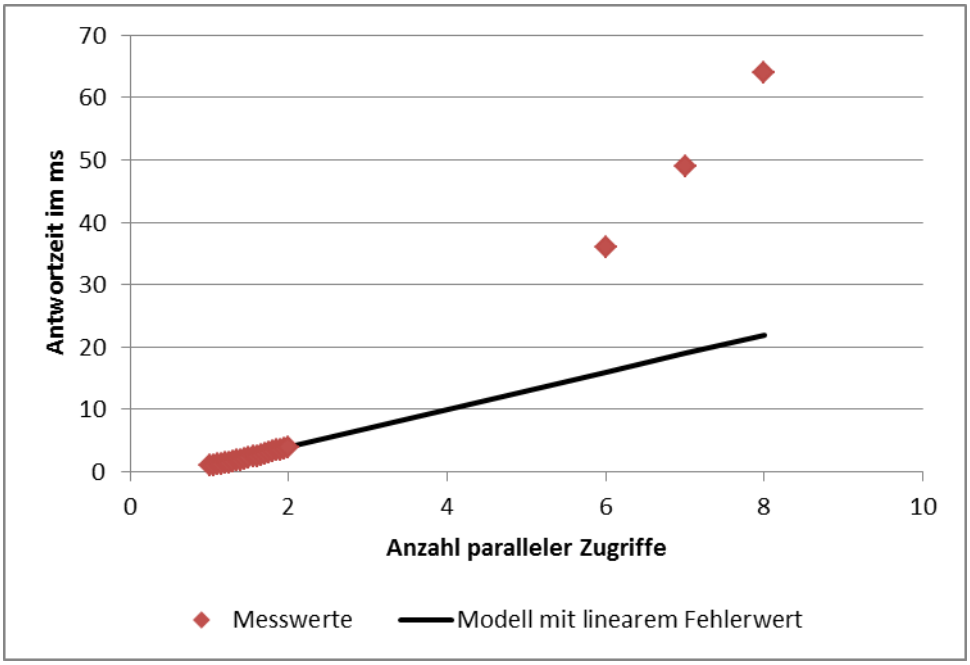


Abbildung 4.47. Problem der nicht gleichverteilten Messwerte bei der linearen Fehlerwertberechnung, synthetisches Beispiel. Quelle: Eigene Darstellung.

4.5.2 Quadratischer Fehlerwert

Um das im vorherigen Abschnitt vorgestellte Problem der ungleichverteilten Messwerte zu lösen ist es erforderlich, dass die wenigen Messpunkte im Hochlastbereich stärker auf den Fehlerwert einwirken. Ein Vorgehen hierzu ist, große Fehlerwerte deutlich stärker zu gewichten als geringe.

In die quadratische Fehlerwertberechnung geht die geometrische Distanz zwischen Modellierungswert und Messwert für einen Messpunkt quadratisch ein. Auf diese Weise erfahren hohe Fehlerwerte eine deutliche Auswirkung auf das Gesamtergebnis – während der Modellfehler an Punkt x = 8 in Abbildung 4.47 bei der linearen Fehlerwertberechnung mit etwa 40 Punkten eingeht bewirkt er in der quadratischen Fehlerwertberechnung sogleich einen Fehlerwert von etwa 1.600. Formel 11 definiert die Formel zur Berechnung des quadratischen Fehlerwerts.

Formel 11. Quadratischer Fehlerwert.

$$s_{err} = \sqrt{\frac{\sum_{i=1}^{n}(r_{measured_i} - r_{modeled_i})^2}{n}}$$

Unter Verwendung des quadratischen Fehlerwerts besteht kaum noch die Möglichkeit, einzelne Messwerte zu ignorieren, um andere Messwerte möglichst exakt abzubilden. Ein evolutionärer Algorithmus ist vielmehr gezwungen, alle Messwerte so nah wie möglich abzubilden. Dies führt in der Praxis zu deutlich realistischeren Modellen, verursacht jedoch Probleme, sobald gravierende Messfehler in den zu modellierenden Messdaten vorliegen.

4.5.3 Exponentieller Fehlerwert

Die Erhöhung des Gesamtfehlerwerts bei starken Abweichungen einzelner Messwerte vom Modell kann durch die Verwendung eines exponentiellen Fehlerwerts weiter verstärkt werden, indem exponentielle Fehlerwerte berechnet werden. Exponentielle Fehlerwerte haben sich jedoch in der Praxis nicht bewährt, da bereits einzelne Messfehler das Konvergenzverhalten des evolutionären Algorithmus stark verlangsamen oder sogar die Konvergenz zu einem stabilen Modell komplett verhindern können. Sie werden deshalb in dieser Arbeit nicht weiter betrachtet.

4.5.4 Komplexer Fehlerwert

Neben den vorab vorgestellten einfachen Fehlerwertberechnungen besteht die Möglichkeit komplexer Fehlerwerte. Komplexe Fehlerwerte unterscheiden sich von den einfachen Fehlerwertberechnungen dadurch, dass sie nicht mit einer einfachen Formel beschrieben werden können, sondern weitere Faktoren wie beispielsweise die im folgenden beschriebene Klassifizierung heranziehen. Komplexe Fehlerwerte können auf diese Weise anwendungsfallspezifische Probleme, wie nicht äquidistante Messwerte, besser behandeln als einfache Fehlerwertberechnungen, verursachen im Allgemeinen jedoch einen höheren Berechnungsaufwand für den Fehler. Ein komplexer Fehlerwert ist immer anwendungsspezifisch.

Im Folgenden wird ein möglicher komplexer Fehlerwert für den in dieser Arbeit behandelten Anwendungsfall vorgestellt. Die Betrachtung von komplexen Fehlerwerten ist eine umfangreiche Aufgabe, die alleine eine Arbeit ausfüllen kann, so dass die komplexen Fehlerwerte hier nur angesprochen werden können. In den im Rahmen dieser Arbeit vorgenommenen Untersuchungen kommen ausschließlich einfache Fehlerwertberechnungen zum Einsatz, so dass dieses Teilkapitel als Anregung und Inspiration für zukünftige Arbeiten zu sehen ist.

4.5.4.1 Klassifizierung der Messwerte

Zur Vermeidung des im einleitenden Abschnitt dieses Teilkapitels beschriebenen Problems der Ungleichverteilung der Messwerte wird eine Klassifizierung auf den Messwerten durchgeführt. Eine Klassifizierung ist ein Verfahren zur eindeutigen Einteilung der Elemente

einer Datenmenge in eine fixe, vordefinierte Menge an Klassen. Innerhalb einer Klasse befinden sich alle Datenpunkte, die eine definierte Bedingung erfüllen, beziehungsweise eine oder mehrere definierte Eigenschaften aufweisen.

Für die in dieser Arbeit betrachteten Datenbestände erfolgt eine Klassifizierung auf jeder Dimension, die kontinuierliche oder großzahlig diskrete Elemente enthält. Solche Dimensionen sind beispielsweise die Parametergröße in Byte oder die Anzahl der parallelen Nutzer bei stark frequentierten Systemen. Bei Dimensionen mit einer geringen Anzahl diskreter Elemente ergibt sich die Klassenbildung durch die Elemente selber. So macht eine weitere Klassifizierung bei der Dimension *Anzahl CPU* wenig Sinn, wenn hier nur die Elemente *4*, *8*, *12* und *16* existieren.

Messungen des Performanceverhaltens von Komponenten von ERP Systemen (siehe beispielsweise (Gradl et al. 2011), (Bögelsack 2011) und (Jehle 2010)) zeigen deutlich ein wiederkehrendes Verhaltensmuster. Dieses Verhaltensmuster lässt sich grob in fünf Bereiche einteilen – den Niedriglastbereich, den unteren Mittellastbereich, den oberen Mittellastbereich, den Hochlastbereich und den Überlastbereich. Eine Analyse von Lastkurven des SAP ERP Systems ergab, dass die folgende Klassenbildung diese Aufteilung gut abbildet.

4.5.4.1.1 Voraussetzung für die Klassifizierung

Voraussetzung für die erfolgreiche Klassifizierung nach dem folgenden Prinzip ist, dass die Messungen zu Beginn des Niedriglastbereichs (z.B. 1 User) begonnen wurden und bei Erkennung der Überlast (System ist ausgelastet, es reagiert nicht mehr. Anfragen werden wegen eines Timeouts regelmäßig abgebrochen) beendet werden. Ist dies nicht der Fall – wenn beispielsweise zu Beginn der Messung bereits erhöhte Last auf dem System ist oder die Last im Überlastbereich weiter gesteigert wird – so kann die vorgeschlagene Klassifizierung zu Fehlern führen.

In Betracht kommen für die Klassifizierung nach dem nachfolgend beschriebenen Verfahren weiterhin nur Dimensionen, die Last abbilden und die sortierbar sind. Die Anzahl paralleler Zugriffe ist damit ein Beispiel für eine klassifizierbare Dimension nach dem beschriebenen Verfahren, während es die Anzahl vorhandener CPU nicht ist.

4.5.4.1.2 Klassifizierung

Erfüllt eine Menge von Messwerten auf einer zu klassifizierenden Dimension die im vorherigen Abschnitt genannten Bedingungen, so liefert eine Fünfteilung der Skala nach dem folgenden Verfahren gute Klassen.

- Die Skala der Dimension wird vom ersten Element (z.B. 1 User) bis zum letzten Element (z.B. 1000 parallele User) betrachtet.
- Das erste Viertel (1 User – 250 User) bildet den Niedriglastbereich.
- Das zweite Viertel (251 User – 500 User) bildet den unteren Mittellastbereich.
- Das dritte Viertel (501 User – 750 User) bildet den oberen Mittellastbereich.
- Das vierte Viertel (751 User – 1000 User) wird erneut in zwei Teile aufgeteilt.
- Das siebte Achtel (751 User – 875 User) bildet den Hochlastbereich.
- Das achte Achtel (876 User – 1000 User) bildet den Überlastbereich.

Abbildung 4.48 stellt die fünf Klassen des komplexen Fehlerwerts grafisch dar.

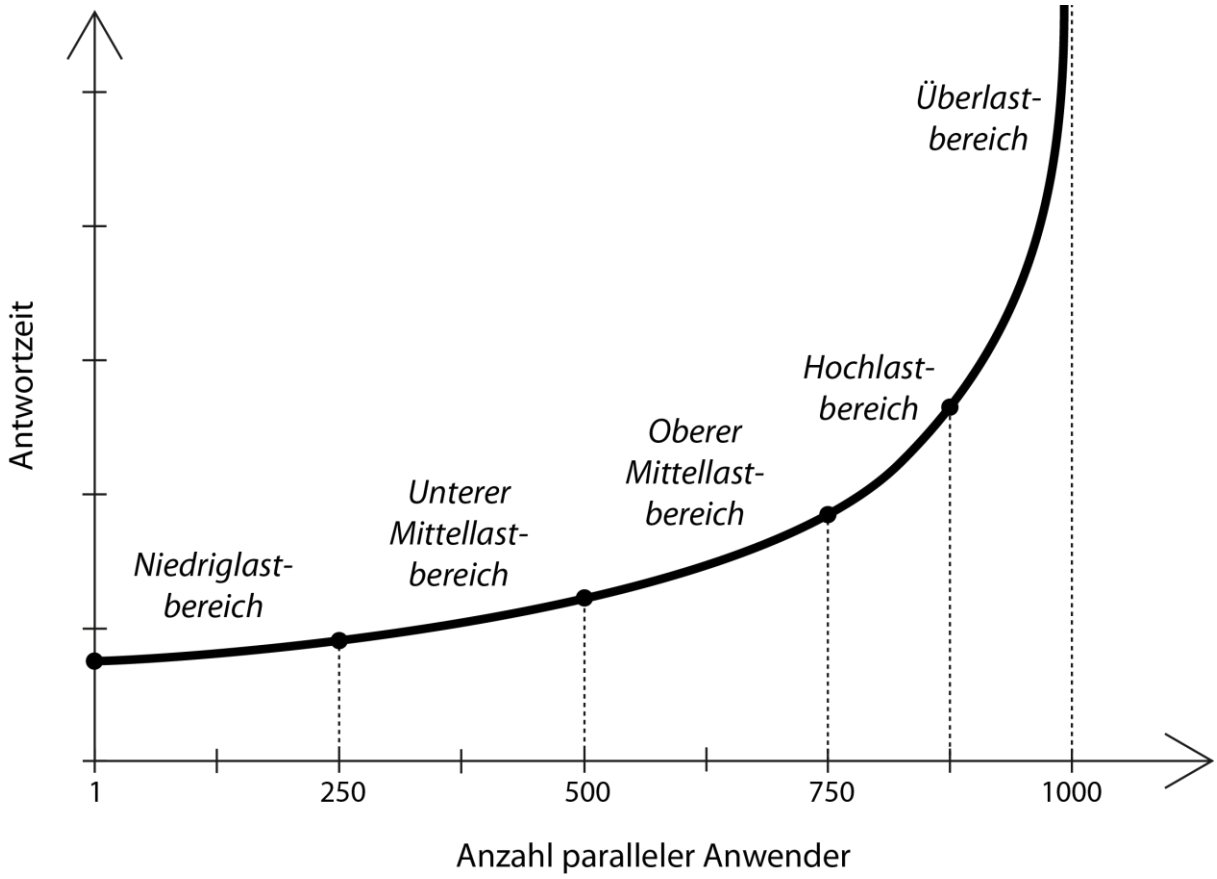


Abbildung 4.48. Grafische Darstellung der fünf Klassen des komplexen Fehlerwerts. Quelle: Eigene Darstellung.

Diese Klassifizierung beruht auf Erfahrungswerten und ist in den meisten Fällen nur ungefähr korrekt. Weiterhin ist sie nur auf ihre Anwendbarkeit auf den in dieser Arbeit betrachteten Messdaten (vgl. Abschnitt 4.4.2.1) überprüft. Die Klassifizierung ist jedoch ausreichend genau, um das einleitend beschriebene System ungleich verteilter Messwerte zu beheben. Erfahrungsgemäß liegen zumeist mehr Messwerte für den Niedrig- und Mittellastbereich vor als für den Hoch- und Überlastbereich. Die vorgeschlagene Klassifizierung berücksichtigt

dies durch die feingranularere Klassifizierung im Hochlastbereich. Zugleich ist die Anzahl der Klassen gering, so dass die Evaluierung des Fehlerwerts auch für Datensätze mit mehreren Dimensionen effizient erfolgen kann.

4.5.4.1.3 Alternative Klassifizierung

Zusätzlich zu der in Abschnitt 4.5.4.1.2 vorgeschlagenen Klassifizierung wurde eine gleichmäßige Zehnteilung der Skala betrachtet. Der Zehnteilung liegt die Annahme zugrunde, dass der durch die Ungleichverteilung der Messwerte verursachte Fehler negiert wird, wenn die Klassen klein gewählt werden.

Die Klassifizierung nach dem Prinzip der gleichmäßigen Zehnteilung birgt den Vorteil, dass die Einschränkung auf Lastkurven entfällt. Solange die Skalenwerte sortierbar sind ist die Klassifizierung mithilfe der Zehnteilung möglich. Eine semantische Voranalyse der zu klassifizierenden Dimension entfällt damit.

Gleichzeitig entsteht bei dieser Klassifizierungsmethode der Nachteil einer großen Menge von Klassen bei mehrdimensionalen Datenfeldern. Besteht beispielsweise das Datenfeld aus drei Dimensionen, so gibt es bereits 1.000 Permutationen von Klassen. Da für jede dieser Permutationen ein Fehlerwert bestimmt wird, wie im folgenden Abschnitt beschrieben, resultiert die Klassifizierung durch Zehnteilung in deutlich schlechteren Laufzeiten des evolutionären Algorithmus, ohne messbare Vorteile in der Genauigkeit des Fehlerwerts zu liefern.

4.5.4.2 Fehlerwertberechnung

Der Fehlerwert eines Modells ergibt sich aus der Summe aller Fehlerwerte der einzelnen Klassen. Für jede Klasse wird ein eigener Fehlerwert berechnet, die Summe der einzelnen Klassenfehlerwerte, dividiert durch die Anzahl der Klassen, ergibt den Fehlerwert des Modells.

Der Fehlerwert einer Klasse ergibt sich durch die durchschnittliche geometrische Distanz des Modells zu den in der Klasse enthaltenen Messwerten. Dies ist in Formel 12 beschrieben. s_{err_i} bezeichnet den Fehlerwert der i-ten Klasse, n_i ist die Anzahl der in dieser Klasse enthaltenen Elemente. $r_{measured_j}$ bezeichnet den gemessenen Performancewert am j-ten Punkt der Klasse, $r_{modeled_j}$ entsprechend den modellierten Performancewert an diesem Punkt.

Formel 12. Berechnung des Fehlerwerts einer Klasse.

$$s_{err_i} = \frac{\sum_{j=1}^{n_i} \frac{|r_{measured_j} - r_{modeled_j}|}{r_{measured_j}}}{n_i}$$

Formel 13 beschreibt den Modellfehlerwert $s_{err_{Model}}$. Dieser ergibt sich als durchschnittlicher Fehlerwert aller k Klassen.

Formel 13. Berechnung des Modellfehlerwerts als Summe der Fehlerwerte der Klassen.

$$s_{err_{Model}} = \frac{\sum_{i=1}^{k} s_{err_i}}{k}$$

Durch die Klassenbildung wird das eingangs beschriebene Problem der ungleich über die betrachteten Dimensionen verteilten Messwerte auf ungleich verteilte Messwerte innerhalb der Klassen reduziert. Da die Klassen gleich gewichtet in den Modellfehlerwert einwirken ist dieser von der Verteilung der Messwerte abstrahiert. Durch die fein granulare Klassifizierung im Hoch- und Überlastbereich wirken die dort klassifizierten Messwerte sogar mit doppeltem Gewicht auf die Fehlerwertermittlung ein. Dies ist der Tatsache geschuldet, dass die Messwerte im Hoch- und Überlastbereich deutlich volatiler sind als im Niedrig- und Mittellastbereich.

4.5.5 Vergleich der Fehlerwert-Berechnungsverfahren

Während der Durchführung der zu dieser Arbeit führenden Experimente hat sich die quadratische Fehlerwert-Berechnung als geeignetste Methode herausgestellt. Die im Rahmen dieser Arbeit durchgeführten Messungen umfassen zahlreiche Messdatensätze mit mehreren Hunderttausend Einzelmessungen. Diese Messdatensätze enthalten einige wenige Messfehler, von denen nur wenige gravierende Abweichungen aufweisen (siehe beispielsweise (Tertilt/Krcmar 2012)). Die Messungen sind jedoch durchweg nicht-äquidistant.

Der Einsatz einer linearen Fehlerwertberechnung führt auf den gemessenen Messdatensätzen zu dem in Abschnitt 4.5.1 beschriebenen Verhalten: die gefundenen Modelle schmiegen sich eng an das Groß der Niedriglastmessdaten an und laufen weit an den wenigen Hochlastmessdaten vorbei. Der Einsatz einer exponentiellen Fehlerwertberechnung hingegen bewertet die wenigen Messfehler über und führt dazu, dass der evolutionäre Algorithmus zumeist gar nicht konvergiert. Die quadratische Fehlerwertberechnung hingegen führt zu einer Glättung der Modelle und filtert auf diese

Weise teilweise sogar Messfehler heraus, ohne die Hochlastmesswerte zu ignorieren, solange von diesen ausreichend (ungefähr im Verhältnis von 5:1) Werte vorliegen.

Die komplexe Fehlerwertberechnung kommt in dieser Arbeit nur als Konzept zum Einsatz und wird in den Experimenten aus Zeitgründen nicht umgesetzt. Liegen Klassifikationsinformationen auf den umfangreichen Dimensionen vor, so ist zu vermuten, dass der komplexe Fehlerwert zu einer Verbesserung der Modellbildung- und Konvergenzfähigkeit des evolutionären Algorithmus führt. Diese mögliche Verbesserung geschieht jedoch auf Kosten zusätzlicher Berechnungen und damit einer potentiellen Verlängerung der Modellierungszeit auf gleich bleibender Hardware. Eine ausführliche Untersuchung liegt jedoch außerhalb des Rahmens dieser Arbeit und stellt einen Ansatzpunkt für zukünftige Forschung dar.

4.5.6 Zusammenfassung

In diesem Teilkapitel werden die verschiedenen Verfahren zur Berechnung des Fehlerwerts eines Modells vorgestellt und beleuchtet. Ausgehend vom linearen Fehlerwert, der die Distanz zwischen den Mess- und Modellwerten als Basis für die Fehlerwertberechnung ansetzt, über den quadratischen Fehlerwert, bei dem die geometrische Distanz zwischen Mess- und Modellwert quadratisch in den Fehlerwert einfließt, bis hin zum komplexen Fehlerwert werden Verfahren sowie Vor- und Nachteile der Fehlerwertberechnungen erläutert. Anschließend erfolgt eine vergleichende Bewertung der Fehlerberechnungs-Verfahren.

Die vergleichende Bewertung führt zu dem Ergebnis, dass die quadratische Fehlerwertberechnung am besten geeignet ist, um den Modellierungsfehler des evolutionären Algorithmus bei der Modellierung der Komponentenmodelle abzubilden. Die quadratische Fehlerwertberechnung reduziert das Problem nicht-äquidistanter Messwerte (welches bei der linearen Fehlerwertberechnung zu großen Problemen führen kann), ohne die Komplexität der Berechnung einer komplexen, klassifizierenden Fehlerwertberechnung einzuführen.

Die in diesem Teilkapitel eingeführten Definitionen der Fehlerwertberechnungsverfahren dienen als Grundlage für die Modellierung mittels evolutionärer Algorithmen in der folgenden Arbeit.

4.6 Zusammenfassung

Kapitel 4 beschreibt die Implementierung des evolutionären Algorithmus *Mendel*, die Integration der Ergebnisse von *Mendel* in die PCM-Simulationsumgebung, sowie die Konfiguration von *Mendel* und die Fehlerwertberechnung der generierten Modelle. All diese Punkte zusammen bilden die Experimentvorbereitung des kontrollierten Softwareexperiments, welches im nächsten Kapitel durchgeführt wird.

Der Prototyp *Mendel* bildet die Basis für die in dieser Arbeit durchgeführten Untersuchungen. Er ist in Java implementiert und speziell für diese Arbeit entwickelt und optimiert. Durch die Verwendung von Multithreading wird eine optimale Nutzung der zu Verfügung stehenden x86-Mehrkernmaschinen erreicht. Zudem können auf diese Weise sämtliche Zwischenstände untersucht und die Ausgaben des Algorithmus für diese Arbeit optimiert werden. Erste Modellierungsläufe mittels *Mendel* zeigen seine Eignung für die Erstellung mathematischer Modelle auf gemessenen Antwortzeitdaten. Die generierten Modelle weichen nur wenige Prozent von den gemessenen Werten ab. Diese Erkenntnis ist ausreichend, um Annahme 2.1 als bestätigt anzusehen. Eine weiterführende Analyse der Modellierungsmöglichkeiten von *Mendel* erfolgt in den weiteren Teilkapiteln und bestätigt dieses Ergebnis.

Die Einbindung der durch *Mendel* generierten Komponentenmodelle in die PCM-Simulation erfolgt über die im Rahmen dieser Arbeit entwickelten *Formula*-Implementierung der Performance Curve Integration (PCI). PCI erlaubt das Einbinden des Antwortzeitverhaltens von Komponenten in die Simulation. Die in diesem Kapitel vorgestellte *Formula*-Implementierung ermöglicht die Darstellung der Antwortzeit-Komponentenmodelle in Form mathematischen Formeln. In Teilkapitel 4.3 wird dargestellt, dass hierdurch eine deutliche Effizienzsteigerung gegenüber der ursprünglichen *DataTable*-Implementierung erreicht wird. Zudem ist durch die *Formula*-Implementierung eine direkte Einbindung der Ergebnisse des evolutionären Algorithmus möglich. Annahme 2.2 ist damit ebenfalls als korrekt anzusehen.

Die optimale Konfiguration des evolutionären Algorithmus zur Modellierung von Komponentenmodellen wird auf zwei Arten ermittelt. Zu einen erfolgt eine statistische Analyse, basierend auf den gemessenen Modellierungsergebnissen von 2.904 Modellierungsläufen, die eine komplette Abdeckung aller Permutationen der ausgewählten Konfigurationsparameter des evolutionären Algorithmus bildet. Diese Analyse resultiert in eindeutigen Konfigurationsempfehlungen für die Konfigurationsparameter Populations-Größe und Genome-Länge, liefert jedoch ein breites Feld an möglichen Konfigurationswerten für die drei Parameter Mutations- und Crossover-Wahrscheinlichkeit sowie Parental Population

Quota. Zur Bestimmung von optimalen Konfigurationswerten für die letztgenannten Konfigurationsparameter erfolgt der Einsatz eines evolutionären Meta-Algorithmus (Meta-GA) zur Konfiguration des evolutionären Modellierungs-Algorithmus. Der Meta-GA ermittelt nicht nur eine optimale Gesamtkonfiguration, sondern liefert zudem noch Informationen über die Bandbreite möglicher Konfigurationswerte und Alternativwerte. Aus mehr als fünf Millionen Zwischenergebnissen erfolgt die Identifikation optimaler Konfigurationswerte für alle fünf Konfigurationsparameter des Modellierungs-Algorithmus. Damit ist auch die Annahme 2.3 bestätigt.

Ein weiterer, die Modellierungseigenschaft des evolutionären Algorithmus stark beeinflussender Faktor ist die Wahl der Fehlerwertberechnung. Anhand der Fehlerwertberechnung wird die Güte eines generierten Modells bestimmt. In Teilkapitel 4.5 erfolgt eine Betrachtung möglicher Fehlerwertberechnungen sowie die Abwägung der Vor- und Nachteile. Neben der linearen, quadratischen und exponentiellen Fehlerwertberechnung werden mögliche komplexe Fehlerwertberechnungen vorgestellt. Im Anschluss erfolgen ein Vergleich der vorgestellten Verfahren und die Begründung, weshalb in dieser Arbeit die quadratische Fehlerwertberechnung eingesetzt wird.

Auf diesem Wege ist die zweite Forschungsfrage beantwortet.

Forschungsfrage 2: Wie kann ein bestehender evolutionärer Algorithmus in die Simulation der Performance einer Unternehmensanwendung eingebunden werden und wie muss der evolutionäre Algorithmus konfiguriert sein?

Antwort 2: Der evolutionäre Algorithmus generiert aus Performance-Messwerten der Komponenten der Unternehmensanwendung Komponentenmodelle. Diese Komponentenmodelle bilden das Antwortzeitverhalten der Komponenten ab und liegen in Form von mathematischen Formeln vor. Über die *Formula*-PCI-Implementierung können diese mathematischen Formeln in die Simulation als Beschreibung der Anwendungskomponenten eingebunden werden. Eine optimale Konfiguration des evolutionären Algorithmus lässt sich auf zwei Arten analysieren. Ein Ansatz ist die statistische Analyse vieler Modellierungsläufe. Diese Analyse ergibt eindeutige Konfigurationsempfehlungen für die Populationsgröße und die Genome-Länge, nicht jedoch für die Mutations- und Crossover-Wahrscheinlichkeit. Eindeutigere Ergebnisse für diese werden durch den Einsatz eines evolutionären Algorithmus zur Konfiguration des evolutionären Modellierungs-Algorithmus erzielt. Mittels dieses Meta-GA

ist eine sehr genaue Konfiguration des Modellierungs-Algorithmus möglich. Als optimale Konfiguration des Modellierungs-Algorithmus für den in dieser Arbeit betrachteten Anwendungsfall wurden folgende Konfigurationswerte ermittelt:

- Populationsgröße: 5.000
- Genom-Länge: 41
- Mutations-Wahrscheinlichkeit: 90%
- Crossover-Wahrscheinlichkeit: 75%
- Parental Population Quota: 30%

5 Performance-Simulation

Die in den vorherigen Kapiteln beschriebenen Aktionen dienen der Vorbereitung des kontrollierten Softwareexperiments zur Überprüfung der in Teilkapitel 1.2 aufgestellten Annahmen, insbesondere Annahme 1, sowie der Beantwortung der Forschungsfragen. Durch die Auswahl eines Simulationsframeworks in Kapitel 3, sowie die Implementierung des genetischen Algorithmus *Mendel*, dessen Konfiguration und die Einbindung der generierten Modelle in das Simulationsframework PCM, beschrieben in Kapitel 4, ist das Rahmenwerk für die Durchführung des Experiments aufgestellt. Die Durchführung des kontrollierten Softwareexperiments wird nun in diesem Kapitel beschrieben.

Das Experiment erfolgt anhand eines der Praxis entnommenen Anwendungsfalls. Bei dem Anwendungsfall handelt es sich um einen Ausschnitt aus einer Integrationsanwendung mit Service-orientierter Architektur, mittels derer existierende Unternehmensanwendungen unter einer einheitlichen, prozessorientierten Oberfläche zusammengefasst werden. Da die einzelnen Anwendungen ursprünglich nicht für die Integration in eine SOA konzipiert wurden sind Performance-Probleme nicht ausgeschlossen.

Die Experimentdurchführung zur Überprüfung der Annahme umfasst sechs Schritte. Im ersten Schritt wird der Anwendungsfall auf konventionelle Weise, also unter der Verwendung von Verteilungsfunktionen zur Darstellung des Antwortzeitverhaltens der einzelnen von den integrierten Anwendungen angebotenen Services, simuliert. Im zweiten Schritt erfolgt eine erneute Simulation, diesmal jedoch unter Verwendung der durch den evolutionären Algorithmus generierten Komponentenmodelle. Im dritten Schritt wird das tatsächliche Antwortzeitverhalten des Anwendungsfalls gemessen. Der Anwendungsfall befindet sich zum Zeitpunkt dieser Arbeit im Entwicklungsstadium, die Messung des Antwortzeitverhaltens erfolgt im Rahmen des zum Entwicklungsprozess gehörenden Lasttests. Der vierte und fünfte Schritt umfasst jeweils den Vergleich der Simulationsergebnisse mit dem gemessenen Antwortzeitverhalten. Im sechsten Schritt erfolgt dann letztendlich der Vergleich der Ergebnisse aus Schritt vier und fünf, mit dem Ziel der Überprüfung, ob durch die Verwendung der generierten Komponentenmodelle eine Verbesserung der Simulationsergebnisse erreicht werden konnte. Abbildung 5.1 stellt den Ablauf der Experimentdurchführung grafisch dar.

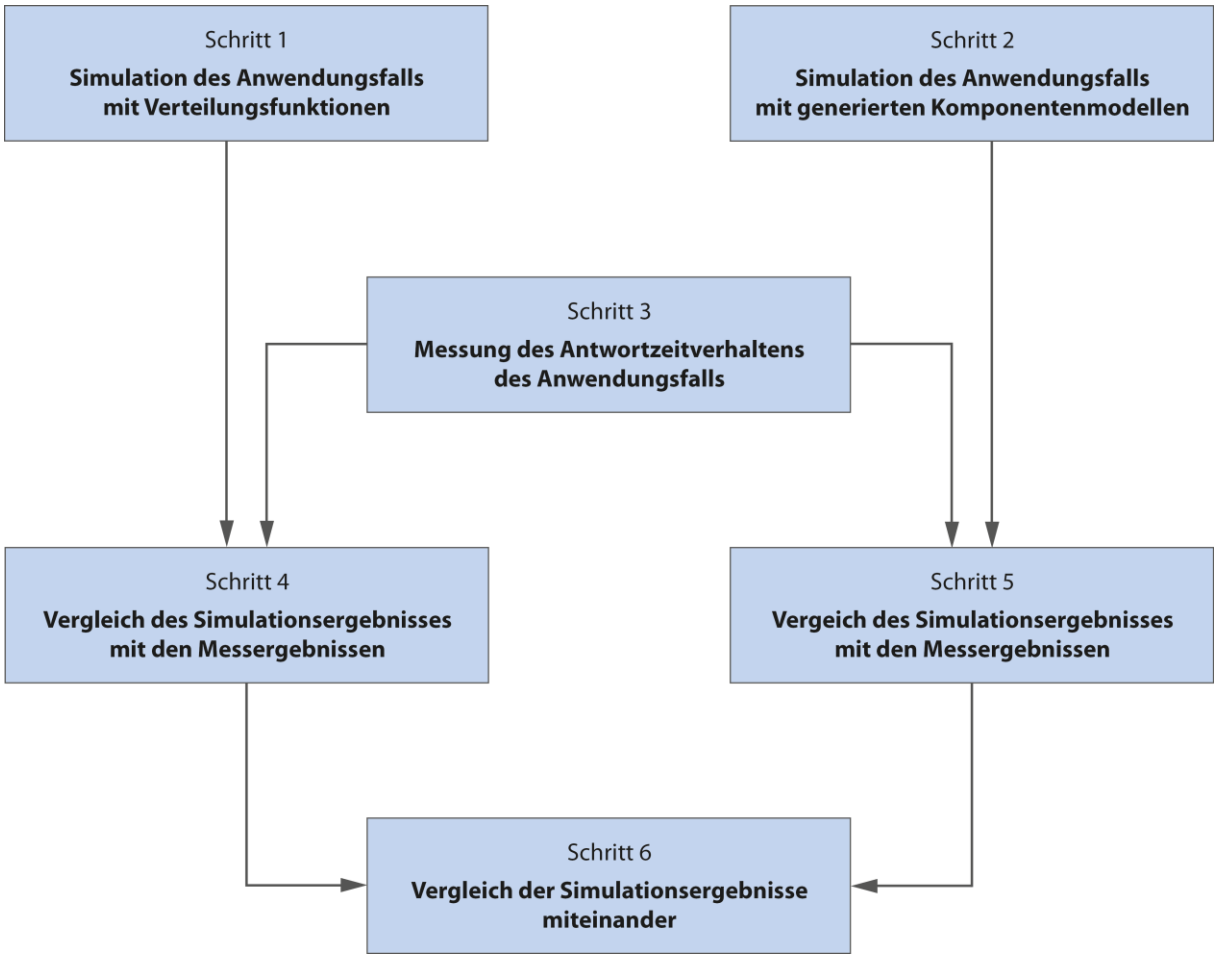


Abbildung 5.1. Ablauf der Experimentdurchführung. Quelle: Eigene Darstellung.

Die folgenden Abschnitte gehen auf die Details des Anwendungsfalls, der Messung dessen Performance-Verhaltens, die Anwendung des evolutionären Algorithmus zur Erzeugung der Komponentenmodelle und den Aufbau und die Durchführung der Simulationsdurchläufe ein. Anschließend erfolgen die Darstellung der Ergebnisse der Experimentdurchführung und eine Analyse, inwieweit die Annahme 1 zutrifft.

5.1 Anwendungsfall

Die Bundesagentur für Arbeit (BA) integriert im Rahmen des Projekts „Rollenbasierte Oberfläche“ (ROBASO) zahlreiche existierende und von der BA betriebene Unternehmensanwendungen (sogenannte *Verfahren*), um den Mitarbeitern und Kunden ein einheitliches Oberflächenkonzept zu präsentieren. Die Integration erfolgt als SOA über einen ESB, wobei die Operationen der einzelnen Verfahren (zum Teil über Facades) als Services angeboten werden (siehe Kapitel 2.6.2 für die Definition der SOA-Elemente). Auf den Operationen eines Verfahrens erfolgt eine logische Gruppierung, so dass es für jedes Verfahren einen oder mehrere Services gibt, welche wiederum mehrere Operationen anbieten.

Im Rahmen des Projekts erfolgt zudem die Implementierung der ROBASO-Anwendung, einer neuimplementierten Oberflächenanwendung zur Orchestrierung der Verfahrensservices, basierend auf ADF Task Flow (Oracle 2011). Die in ROBASO implementierten Oberflächenfolgen bilden die von den Anwendern durchzuführenden Geschäftsprozesse ab und ermöglichen damit eine Prozessausführung ohne Anwendungswechsel.

Das Projekt ROBASO hat eine Laufzeit von fünf Jahren und umfasst die Umsetzung von 14 Geschäftsprozessen. Der iterative Projektplan sieht die Umsetzung von drei Programmversionen pro Jahr vor. Als Untersuchungsobjekt für die vorliegende Arbeit wurde aus den Geschäftsprozessen der Prozess „Terminierung“ in der Version P12.03 ausgewählt. Der Terminierungsprozess umfasst sämtliche Terminfindungsabläufe zwischen Mitarbeitern der BA und Kunden, zumeist Arbeitssuchenden und Interessenten an einer Berufsberatung. Der Geschäftsprozess „Terminierung“ setzt sich aus zwei Hauptprozessen zusammen, welche sich wiederum in mehrere Teilprozesse aufgliedern. Die Auswahl des Terminierungsprozess erfolgte aufgrund der folgenden Vorteile:

- Der Terminierungsprozess ist zum Zeitpunkt des Beginns dieser Arbeit bereits nahezu vollständig spezifiziert.

- Er umfasst die meisten der für die Performance der ROBASO-Anwendung als kritisch eingestuften Services.

- Er ist, in Hinblick auf die anderen Geschäftsprozesse, von durchschnittlicher Komplexität.

- Die Implementierung des Terminierungsprozesses als ROBASO-Anwendung erfolgt parallel zur Erstellung dieser Arbeit, so dass die gewonnenen Simulationsergebnisse mit realen Messdaten des Testsystems verglichen werden können.

Im Folgenden werden die Bestandteile des Terminierungsprozesses beschrieben.

5.1.1 Bestandteile des Terminierungsprozesses

Zur Beantwortung der dritten Forschungsfrage sind drei Sichten auf den Anwendungsfall notwendig. Zum ersten ist die Prozessansicht von Bedeutung. Diese Sicht betrachtet den Terminierungsprozess als Geschäftsprozess mit all seinen Haupt- und Teilprozessen, ohne die technische Umsetzung zu berücksichtigen. Über die Prozessansicht werden die Folgen

von Serviceaufrufen ersichtlich, die notwendig sind, damit ein Nutzer sein Ziel, die Erfüllung einer geschäftlichen Aufgabe, erreicht. Die zweite Sicht berücksichtigt die verwendeten Verfahren und die von ihnen angebotenen Services. Da es sich bei der ROBASO-Implementierung des Terminierungsprozesses um eine SOA handelt erfolgt eine ausschließliche Betrachtung der Services, die Implementierungsdetails der Verfahren bleiben bis auf wenige grundlegende und für die Performance-Analyse notwendige Architekturbetrachtungen unberücksichtigt. Die technische Architektur und Implementierungsdetails des ROBASO-Terminierungsprozesses wiederum finden Betrachtung in der dritten Sicht. Hier werden technische Details hinsichtlich der Umsetzung sowie die verwendete Middleware erläutert, soweit sie für die Performance-Analyse relevant sind.

5.1.1.1 Haupt- und Teilprozesse

Der Terminierungsprozess ist ein Geschäftsprozess und setzt sich aus den folgenden Hauptprozessen zusammen:

- Terminierung vornehmen
- Terminverschiebung/ -absage vornehmen

Jeder dieser Hauptprozesse entspricht einer Fachaufgabe eines Anwenders, im Allgemeinen eines Mitarbeiters der Eingangszone (in einer Arbeitsagentur) oder des Service Centers (Call Center). Ein Prozess umfasst sowohl manuelle als auch automatisierte Schritte, wie im Folgenden beschrieben, und hat im Allgemeinen einen Arbeitsaufwand von wenigen Minuten. Der Terminierungsprozess umfasst keine langlaufenden Prozesse, diese sind auch in den weiteren, später umzusetzenden (und in dieser Arbeit nicht berücksichtigten) anderen Geschäftsprozessen nicht geplant.

Im Folgenden werden die Bestandteile des Terminierungsprozesses im Detail betrachtet. Hierbei wird insbesondere auf die verwendeten Verfahren und Services eingegangen. Die Darstellung der betrachteten Prozessabläufe erfolgt im Anhang.

5.1.1.2 Verwendete Verfahren und Services

Als (IT-)Verfahren werden im Kontext der BA Softwareanwendungen bezeichnet, die eigenständig betrieben werden und eine abgeschlossene Gruppe von Geschäftsvorfällen und Aufgaben unterstützen. So bietet beispielsweise die Allgemeine Terminverwaltung (ATV) Funktionalitäten zum Anlegen, Ändern und Löschen eines Termins.

Die Mengen der durch die einzelnen Verfahren implementierten Funktionalitäten sind disjunkt. Kein anderes Verfahren neben der ATV implementiert damit beispielsweise Operationen auf Terminen, und Personendaten werden singulär in der Zentralen Personendatenverwaltung (ZPDV) abgelegt. Die notwendige Kommunikation zwischen den Verfahren erfolgt bislang entweder über einen CORBA-Bus (Object Management Group 2012), über proprietäre Integrationsmechanismen, oder schlicht über das manuelle Kopieren von Daten über die unabhängigen Oberflächenanwendungen der Verfahren.

Im Rahmen von ROBASO werden zur Implementierung der Oberflächenanwendung des Terminierungsprozesses Funktionalitäten aus acht Verfahren integriert. Tabelle 5 führt diese Verfahren mit einer kurzen Beschreibung alphabetisch sortiert auf.

Tabelle 5. Verwendete Verfahren im ROBASO-Terminierungsprozess.

Verfahrensname	**Abkürzung**	**Beschreibung**
Allgemeine Aufgabenverwaltung	AAV	In ROBASO werden anstehende Teilprozesse als Aufgaben an Rollen zugewiesen. Jeder Mitarbeiter im Service Center oder in der Eingangszone bekleidet eine oder mehrere Rollen. Die Verwaltung offener Aufgaben, die Überprüfung, ob ein Mitarbeiter eine Aufgabe bearbeiten darf, sowie die Zuweisung einer Aufgabe an einen Mitarbeiter, erfolgt über die AAV.
Aufgabenempfängerkreisermittlung	AEKE	Bei der Erstellung einer Aufgabe, die in einer spezifischen Geschäftsstelle der BA erledigt werden muss, wird die Angabe eines Empfängerkreises für diese Aufgabe benötigt. Dieser Empfängerkreis wird über AEKE ermittelt.
Allgemeine Terminverwaltung	ATV	Die ATV dient zum Anlegen, Ändern und Löschen von Terminen. Zudem können über die ATV Terminvorschläge und Einladungen generiert werden
Basisdienst Dienststellen- / Träger- und Standortverzeichnis	BADIV	Das Verfahren BADIV dient zur Verwaltung von Dienststellen- und Standort-informationen. BADIV bietet Services an, um die Dienststelle eines Mitarbeiters zu ermitteln sowie Organisationseinheiten von Arbeitsämtern aufzulisten. Diese Informationen werden unter anderem bei der Terminvereinbarung benötigt.

Computerunterstütztes Leistungs-berechnungs- und Informationssystem für das Arbeitslosengeld	COLIBRI	COLIBRI ist ein Bearbeitungssystem zur elektronischen Unterstützung bei der Abwicklung des Arbeitslosengeld I. Das Verfahren enthält ein umfangreiches Auskunftsmodul zur Unterstützung der Sachbearbeiter sowie die Berechnungsroutinen für die zu erbringenden Entgeltersatzleistungen.
Personenrollenservice	PEROS	Das Verfahren PEROS weist Personen (im Allgemeinen Mitarbeitern) Rollen zu. Das Berechtigungskonzept in ROBASO ist rollenbasiert, so dass durch PEROS eine Zuordnung von Berechtigungen an Mitarbeiter stattfindet. Wann immer eine fachliche Aufgabe angelegt, übernommen oder geschlossen wird erfolgt eine Prüfung der Berechtigung über PEROS.
Virtueller Arbeitsmarkt / Vermittlungs-, Beratungs- und Informationssystem	VAM/VERBIS	VAM/VERBIS ist das zentrale Verfahren zur Vermittlung Arbeitssuchender. Über VAM/VERBIS werden offene Stellen verwaltet und mit den Profilen Arbeitssuchender abgeglichen. VAM ist dabei der öffentlich zugängliche Bereich, während VERBIS ausschließlich den Sachbearbeitern der BA zur Verfügung steht.
Zentrale Personendatenverwaltung	ZPDV	Sämtliche personenbezogenen Daten werden von der ZPDV verwaltet. Die ZPDV bildet den zentralen Speicherpunkt für Informationen wie Adressen und Telefonnummern, aber auch Bank-verbindungen und Verweise auf Leistungsdaten. Die ZPDV wird von zahlreichen anderen Verfahren verwendet.

Die von diesen Verfahren für ROBASO benötigten Funktionalitäten werden als Services (siehe Abschnitt 2.6.2.2) angeboten. Die Funktionalitäten werden dabei entweder als elementare Service-Operationen oder als Composed Services (CS) bereitgestellt. In Tabelle 6 sind die vom Terminierungsprozess verwendeten Service-Operationen und Composed Services beschrieben. Details zu den Composed Services folgen im Abschnitt 5.1.1.3.

Tabelle 6. Vom ROBASO-Terminierungsprozess verwendete Service-Operationen.

Operation	**Service**	**Verfah-ren**	**Beschreibung**
createAufgabe	AAV_ROBASO_ UCS	AAV	Legt eine Aufgabe in der AAV an.
findAufgabeByFilter	AAV_ROBASO_ UCS	AAV	Sucht alle Aufgaben, die einem Filterkriterium entsprechen.
findAufgabeByKundenID	AAV_ROBASO_ UCS	AAV	Sucht alle Aufgaben, die einen Kunden betreffen.
findZustaendigeBenutzer ByArbeitsschrittAnd_ Geschaeftsstelle	AEKE_ROBASO_ UCS	AEKE	Findet alle zuständigen Nutzer (Empfängerkreis) für einen Arbeitsschritt und eine Geschäftsstelle.
createEinladung	EinladungService	ATV	Erstellt eine Einladung an einen Kunden.
findEinladung	EinladungService	ATV	Findet eine bereits erstellte Einladung anhand der Kunden-ID, dem Zeitpunkt der Erstellung und anderer Kriterien.
getEinladungs_ voreinstellungen	EinladungService	ATV	Liefert die Voreinstellungen zu einer Einladung.
getDienststelle	Organisations_ strukturenService	ATV	Liefert die für einen Kunden zuständige Dienststelle.
createTermin	TerminService	ATV	Legt einen Termin für einen Kunden an.
deleteTermin	TerminService	ATV	Löscht einen Termin.
findFreieZeitraeume	TerminService	ATV	Sucht freie Zeiträume bei einem Mitarbeiter für die Erstellung eines Termins.
findKundenTermineBy_ Zeitraum	TerminService	ATV	Sucht alle Kundentermine innerhalb eines Zeitraums.
findMitarbeiterOrTeam_ Vorschlaege	TerminService	ATV	Sucht einen passenden Mitarbeiter oder Team für einen Kundentermin.
findTerminOrt_ VorschlaegeByBA_ Mitarbeiter	TerminService	ATV	Sucht Termin- und Ortsvorschläge für einen Mitarbeiter.
findTerminOrtVorschlaeg eByTeam	TerminService	ATV	Sucht Termin- und Ortsvorschläge für ein Team.
getTermin	TerminService	ATV	Liefert Informationen zu einem spezifischen Termin.
getTermin_ voreinstellungen	TerminService	ATV	Liefert die Voreinstellungen zu einem Termin.

findOrgEinheiten	Organisations_ strukturenService	BADIV	Sucht Organisationseinheiten anhand verschiedener Suchkriterien.
findOrgEinheitenByAA	Organisations_ strukturenService	BADIV	Sucht Organisationseinheiten einer Arbeitsagentur (AA).
getDienststelle	Dienststellen_ Service_V3	BADIV	Liefert die Dienststelle zu einem Mitarbeiter.
getLeistungsdaten	ColibriDatenService	COLIBRI	Liefert die Leistungsdaten zu einem Kunden.
createAufgabeVermerk	AufgabeVermerk ServiceCS	CS	Erzeugt eine Aufgabe sowie einen Vermerk zu dieser Aufgabe.
getBetreuerUndVertreter ByKundennummer	Betreuerdaten ServiceCS	CS	Liefert den Betreuer und dessen Vertreter für einen Kunden.
createEinladungVermerk	Terminierung_ ServiceCS	CS	Legt eine Einladung sowie den dazu gehörenden Vermerk an.
findTerminverantwortliche UndOrtsVorschlaegeBy_ KundennummerAnd_ Terminart	TerminVerwaltung_ ServiceCS	CS	Sucht Terminverantwortliche und Ortsvorschläge für einen Kunden und eine Terminart.
findMitarbeiter	Benutzerdaten_ Service	PEROS	Liefert die Daten eines Mitarbeiters anhand verschiedener Suchkriterien.
getMitarbeiterbyUserID	Benutzerdaten_ Service	PEROS	Liefert die Daten eines Mitarbeiters anhand dessen User ID.
findUserByID	BenutzerService	PEROS	Liefert die Nutzerdaten anhand der User ID.
getBetreuerBy_ Kundennummer	Betreuerdaten_ Service	VERBIS	Liefert die Daten des Betreuers eines Kunden.
getMitarbeiterfunktionen_ ByUserID	Betreuerdaten_ Service	VERBIS	Liefert die Funktionen eines Mitarbeiters anhand dessen User ID.
getVertreterByUserID	Betreuerdaten_ Service	VERBIS	Liefert den Vertreter eines Mitarbeiters anhand dessen User ID.
betreuerAuflisten	PersonenBetreuer_ Service	VERBIS	Listet alle Betreuer eines Kunden auf.
mitarbeiterFunktionen_ Auflisten	PersonenBetreuer_ Service	VERBIS	Listet alle Funktionen eines Mitarbeiters auf.
vertreterAuflisten	PersonenBetreuer_ Service	VERBIS	Listet alle Vertreter eines Mitarbeiters auf.
vermerkErstellen	VermerkeService	VERBIS	Legt einen Vermerk an.
vermerkeAuflisten	VermerkeService	VERBIS	Listet alle Vermerke zu einem Anliegen auf.
vermerkLesen	VermerkeService	VERBIS	Gibt den Inhalt eines Vermerks zurück.

getAktuelleAdresseById	PersonenService	ZPDV	Liefert die aktuelle Adresse einer Person zurück.
getDritterById	PersonenService	ZPDV	Liefert weitere Beteiligte zu den Anliegen einer Person.
getPersonById	PersonenService	ZPDV	Liefert die Personendaten zu einer ID.
getVertretungenKunde_ ByKundennummer	PersonenService	ZPDV	Liefert alle gesetzlichen Vertreter eines Kunden anhand dessen Kundennummer.

Alle in Tabelle 6 beschriebenen Operationen sind Bestandteil des Terminierungsprozesses. Sämtliche Operationen sind elementare Serviceoperationen, bis auf die unter Verfahren als CS (Composed Service) bezeichneten. Auf die Composed Services wird im folgenden Abschnitt im Detail eingegangen.

5.1.1.3 Composed Services

Composed Services (CS) sind Service-Operationen, welche wiederum andere Service-Operationen initiieren. CS bilden damit Implementierungen von Teilprozessen auf der Workflow Engine (in diesem Fall der Web Logic Application Server, siehe Abschnitt 5.1.3), welche wiederum als Service-Operationen ROBASO (und anderen Anwendern) bereitgestellt werden. Im Rahmen des ROBASO-Terminierungsprozesses finden vier Composed Services Anwendung, welche im Folgenden beschrieben werden. Composed Services erzeugen intensive Kommunikation auf dem Enterprise Service Bus (siehe hierzu Abschnitt 2.6.2.4) und bilden damit eine potentielle Performance-Schwachstelle. Zudem erfolgt durch die Composed Services eine intensive und „verdeckte“ Nutzung der Verfahrens-Services. Aus diesem Grund werden sie in diesem Teilkapitel im Detail vorgestellt.

Die grafische Darstellung der Composed Services ist sehr umfangreich und wurde aus diesem Grund in Anhang B verlagert.

5.1.1.3.1 AAV_ROBASO_UCS

Wann immer in ROBASO eine Aufgabe angelegt wird erfolgt das Erstellen eines Vermerks über den *VermerkeService* von VERBIS. Im Composed Service *AAV_ROBASO_UCS* wird diese immer wiederkehrende Kombination zusammengefasst. Der ROBASO-Terminierungsprozess nutzt von diesem CS die einzige Operation *createAufgabeVermerk*. Abbildung B.1 stellt den Aufbau des CS grafisch dar.

5.1.1.3.2 BetreuerdatenServiceCS

Für das Erstellen eines Termins mit einem Kunden werden die Detailinformationen wie beispielsweise der Einsatzort und der Kalender des Betreuers und oft auch dessen Vertreters benötigt. Um diese in einem einzigen Prozessschritt zu erlangen bietet der *BetreuerdatenServiceCS* die Operation *getBetreuerUndVertreterByKundennummer* an. Die Operation kontaktiert sowohl VERBIS, zur Identifikation des Betreuers und dessen Vertreter anhand der Kundennummer, als auch PEROS und BADIV bezüglich der Detailinformationen des Betreuers und des Vertreters. Abbildung B.2 bildet den Aufbau des CS ab.

5.1.1.3.3 TerminierungServiceCS

Zu jeder über den ROBASO-Terminierungsprozess angelegten Einladung wird ein Vermerk in der Akte des Kunden angelegt. Für diesen obligatorischen Schritt bietet der *TerminierungServiceCS* die Operation *createEinladungVermerk*. *createEinladungVermerk* führt das Anlegen des Termins in Kombination mit dem Anlegen des Vermerks in einer einzigen Serviceoperation durch und kontaktiert hierzu sowohl den TerminService der ATV als auch den VermerkeService von VERBIS. Abbildung B.3 stellt den Aufbau des CS dar.

5.1.1.3.4 TerminVerwaltungServiceCS

Der Composed Service *TerminVerwaltungsServiceCS* bietet zwei Operationen: *findTerminverantwortlicheUndOrtsVorschlaegeByKundennummerAndTerminart* und *findTermineMitEinladungByKundennummer*. Diese Operationen werden zur Verwaltung der Termine eines Kunden verwendet. Die erste Operation schlägt für einen Kunden und eine Terminart (z.B. Arbeitslosengeldantrag oder Berufsberatung) einen Terminverantwortlichen und einen Ort vor. Die zweite Operation findet alle Termine eines Kunden, für die Einladungen verschickt wurden. Der *TerminVerwaltungServiceCS* verwendet zahlreiche andere Services, wie den *TerminService* der ATV, den *PersonenBetreuerService* von VERBIS und den *DienststellenServiceV3* von BADIV. Abbildung B.4 sowie Abbildung B.5 stellen die Abläufe der im Terminierungsprozess verwendeten Serviceoperationen des *TerminVerwaltungServiceCS* grafisch dar.

5.1.2 Betrachteter Hauptprozess

Als Anwendungsfall dient in dem in dieser Arbeit durchgeführten Experiment der Hauptprozess *Terminierung vornehmen* aus dem Geschäftsprozess *Terminierung*. Der Hauptprozess *Terminierung vornehmen* umfasst die Terminfindung eines Sachbearbeiters mit einem Kunden zum Zweck der Berufsberatung, der Bearbeitung von Anliegen im Rahmen des Arbeitslosengeldes oder zu sonstigen Anliegen. Er gliedert sich damit in die

Teilprozesse *Terminierung Berufsberater*, *Terminierung Arbeitslosengeld* und *Terminierung sonstige Anliegen* auf.

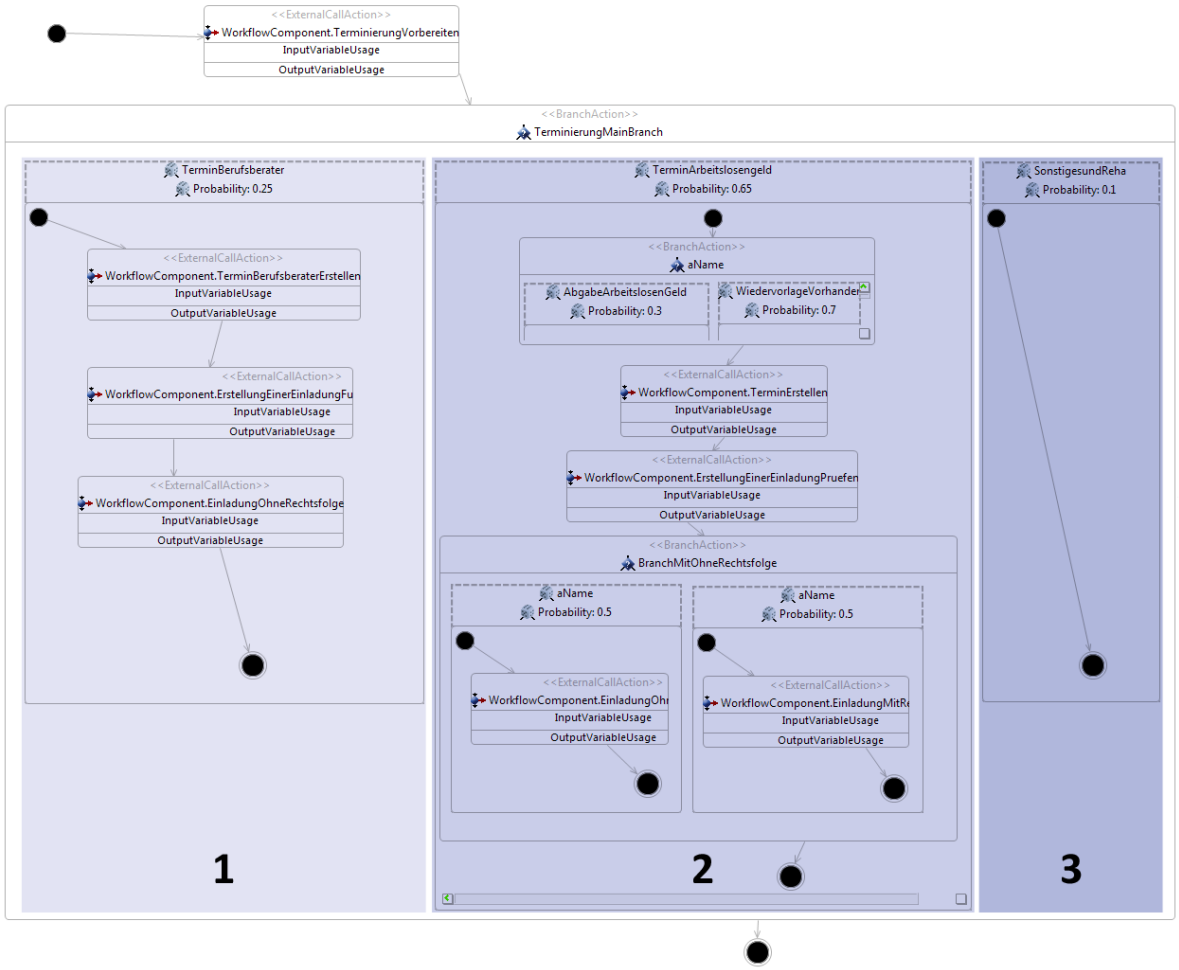


Abbildung 5.2. PCM-Abbildung des Hauptprozesses *Terminierung vornehmen*. Quelle: Eigene Darstellung.

Abbildung 5.2 zeigt die oberste Ebene des Hauptprozesses *Terminierung vornehmen*. Deutlich erkennbar sind die drei Teilprozesse *Terminierung Berufsberater* (Teilprozess 1), *Terminierung Arbeitslosengeld* (Teilprozess 2) und *Terminierung sonstige Anliegen* (Teilprozess 3).

Vor Ausführung jedes Teilprozesses erfolgt die Aktion *Terminierung vorbereiten*. In *Terminierung vorbereiten* werden die Informationen zum Kunden, wie Name und Kundennummer, aufgenommen. Anschließend erfolgt die Verzweigung in einen der Teilprozesse.

Je Kundeninteraktion wird genau ein Teilprozess betreten. Die Verzweigung ist nicht gleichverteilt – eine Analyse der Log-Informationen sowie mithilfe der Fachabteilung ergab

eine Wahrscheinlichkeit von 65 Prozent, dass die *Terminierung Arbeitslosengeld* betreten wird, 25 Prozent für die *Terminierung Berufsberatung* und 10 Prozent *Terminierung sonstige Anliegen*. Die sonstigen Anliegen werden in einem gesonderten Hauptprozess behandelt, so dass sie hier nicht weiter betrachtet werden.

Jede der in den Teilprozessen ausgeführten Workflow-Aktivitäten besteht wiederum aus einem Teilprozessabschnitt. Insgesamt setzt sich das Modell des Hauptprozesses *Terminierung vornehmen* aus über 90 Einzeldiagrammen zusammen. Diese an dieser Stelle aufzuführen und zu illustrieren ist nicht zielführend, die relevanten Diagramme (primär die später vorgestellten Composed Services) sind in Anhang B angefügt. Soweit für die vorliegende Arbeit relevant werden einzelne Modellabschnitte im Folgenden erläutert.

Zum Zeitpunkt der Erstellung dieser Arbeit liegen aus dem Lasttest der ROBASO-Anwendung ausschließlich belastbare Messwerte für eine Terminierung im Rahmen der regelmäßigen Meldung vor. Zur Gewährleistung der Vergleichbarkeit wird auch nur dieser Prozessablauf simuliert.

5.1.3 Software- und Hardwarearchitektur

Die Software- und Hardwarearchitektur der ROBASO-Anwendung hat einen großen Einfluss auf das Performance-Verhalten der Anwendung und damit auch auf das Performance-Simulationsmodell. In den folgenden Abschnitten werden die Architekturen im Detail beschrieben. Zudem erfolgt eine Betrachtung der verwendeten Middleware, die das Architekturbild vervollständigt.

5.1.3.1 Softwarearchitektur der ROBASO-Anwendung

Die ROBASO-Anwendung ist ein mehrschichtiges Softwaresystem auf Basis einer SOA. Als Integrationsanwendung baut ROBASO sowohl auf neuen SOA-Services (wie beispielsweise den AAV-Services) als auch auf Operationen von Bestandsverfahren, die durch Fassaden (vgl. Abschnitt 2.6.2.5) als Service zu Verfügung gestellt werden, auf. Der Zugriff auf die Services erfolgt in beiden Fällen über den ESB. Abbildung 5.3 illustriert dies.

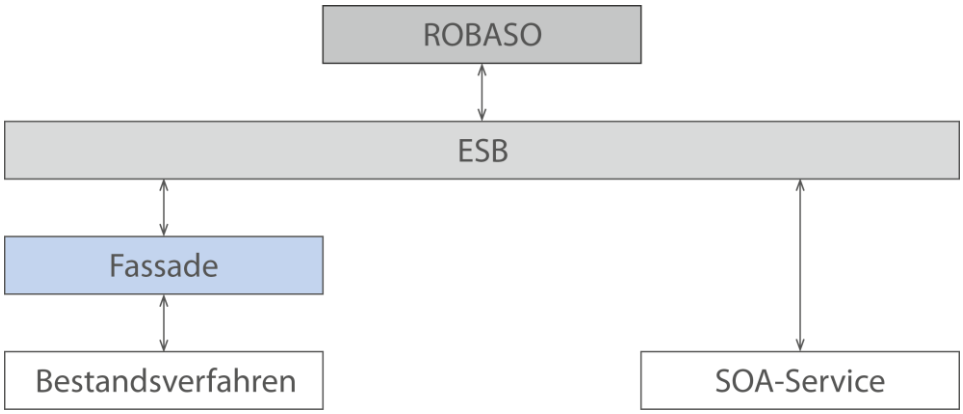


Abbildung 5.3. Zugriff auf neue SOA-Services sowie auf Bestandsverfahren durch ROBASO. Quelle: Eigene Darstellung.

ROBASO in seiner Gesamtheit lässt sich damit in fünf logische Architekturebenen unterteilen: das GUI, die Geschäftsprozesse, die Dienste (implementiert durch die Services), die Komponenten sowie die Betriebsinfrastruktur und die Datenhaltung. Die Architekturebenen sind in Abbildung 5.4 dargestellt.

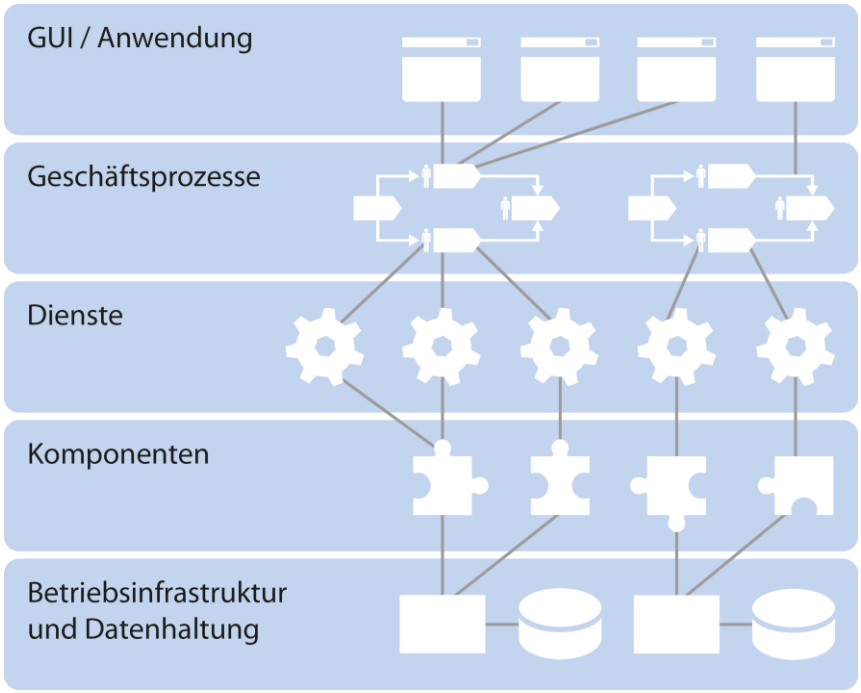


Abbildung 5.4. Logische Architektur der ROBASO-Anwendung. Quelle: Eigene Darstellung, nach Projektdokumentation.

- **GUI/Anwendung** – Die Ebene *GUI/Anwendung* dient der Darstellung der ROBASO-Oberflächenanwendung beim Nutzer. Weiterhin enthält sie auf Darstellungsseite ausgeführte Logik, wie die Vorselektion von Listenelementen und die Eingabevalidierung.

- **Geschäftsprozesse** – Die Ebene *Geschäftsprozesse* bildet die in ROBASO umgesetzten Geschäftsabläufe ab. In dieser Ebene findet die eigentliche Anwendungslogik statt. Die Geschäftsprozesse bestimmen, wann welcher Dienst verwendet wird.

- **Dienste** – *Dienste* sind die von den Komponenten, im Allgemeinen die vorhandenen und implementierten Anwendungen, angebotenen Funktionen. Im Rahmen von ROBASO sind Dienste immer Services. Die Ebene *Dienste* stellt alle Funktionalitäten bereit, die von ROBASO verwendet werden können. ROBASO greift nicht direkt auf die Komponenten durch.

- **Komponenten** – Die *Komponenten* sind die Bestandsanwendungen sowie die neu implementierten Anwendungen, die Dienste in Form von Services bereitstellen. Die *Komponenten* werden nicht alleine von ROBASO genutzt, sondern bieten ihre Funktionalitäten auch anderweitig, meist in Form einer eigenen Oberfläche, an.

- **Betriebsinfrastruktur und Datenhaltung** – Die Ebene *Betriebsinfrastruktur und Datenhaltung* enthält die Middle- und Hardware, auf der die Komponenten betrieben werden. Weiterhin enthält sie das Datenbank-Cluster, welches zur zentralen Datenhaltung dient.

Diese mehrschichtige logische Architektur bringt den Vorteil der losen Kopplung mit sich. Einzelne Ebenen-Elemente können verändert werden (beispielsweise können Verfahren weiterentwickelt werden), ohne dass die anderen Ebenen davon betroffen sind. Zudem ermöglicht die Verwendung von Services eine einfache Wiederverwendbarkeit.

Betrachtet man die eigentliche ROBASO-Anwendung, so ergibt sich ein Schichtenmodell wie in Abbildung 5.5 dargestellt.

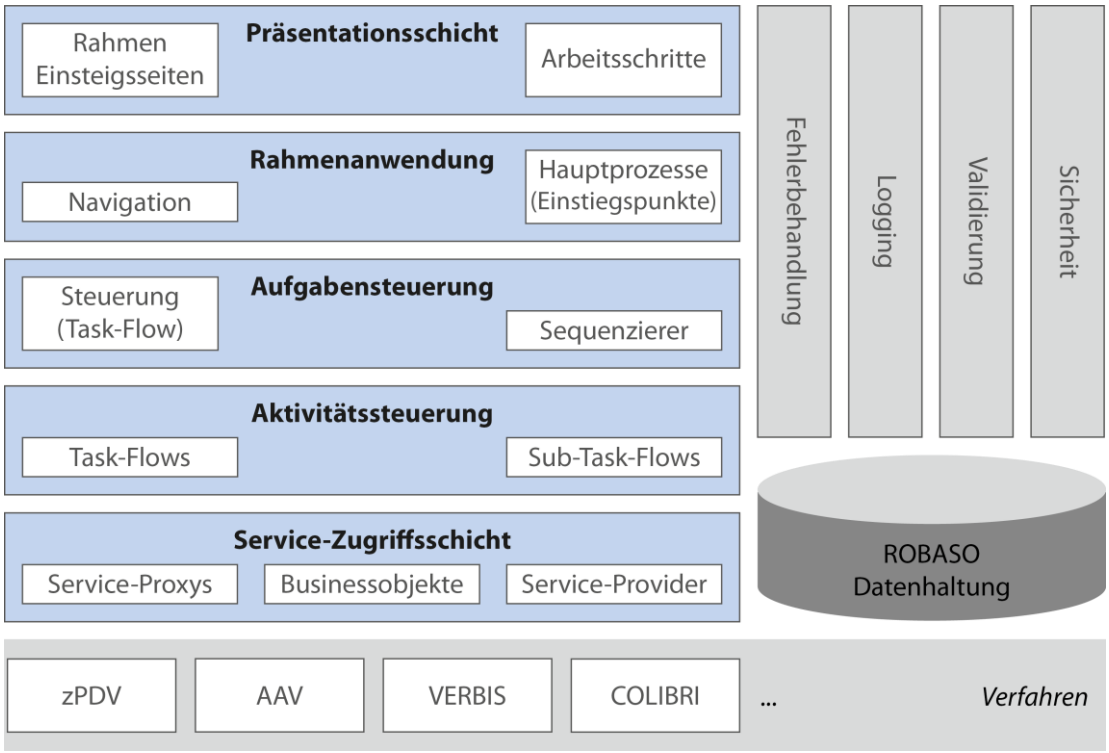


Abbildung 5.5. Schichtenmodell der ROBASO-Anwendung. Quelle: Eigene Darstellung, nach Projektdokumentation.

All diese Schichten finden sich in der vorab vorgestellten logischen Architektur auf den Ebenen *GUI/Anwendung* und *Geschäftsprozesse* wieder. Im Folgenden werden sie im Detail vorgestellt.

- **Präsentationsschicht** – Die Präsentationsschicht dient der grafischen Darstellung der ROBASO-Anwendung beim Anwender. Die Nutzeroberfläche ist in ROBASO als Web-Frontend in ADF implementiert.

- **Rahmenanwendung** – Die Rahmenanwendung handhabt die Breadcrumb-Navigation durch die Anwendung und initiiert die Einstiege in die Hauptprozesse. Die Rahmenanwendungsschicht ist in ROBASO durch ADF Taskflow implementiert.

- **Aufgabensteuerung** – Die Aufgabensteuerung übernimmt die Navigation durch die Hauptprozesse in Form sogenannter Taskflows, also Prozessabläufe. Diese Taskflows werden vom Sequenzierer in Teilprozesse aufgespalten. Jeder Teilprozess entspricht einer Aufgabe. Die Aufgabensteuerung ist ebenfalls als ADF Taskflow implementiert.

- **Aktivitätssteuerung** – Eine Aktivität ist ein Schritt in einem Teilprozess. Die Aktivitätssteuerung initiiert einzelne Aktivitäten. Handelt es sich bei den Aktivitäten um einen Service, so nutzt die Aktivitätssteuerung die Service-Zugriffsschicht, um den Service aufzurufen. Handelt es sich bei der Aktivität hingegen um eine Nutzeraktion, so initiiert die Aktivitätssteuerung die Bereitstellung der entsprechenden Maske durch die Präsentationsschicht. Auch die Aktivitätssteuerung ist als ADF Taskflow implementiert.

- **Service-Zugriffsschicht** – Die Service-Zugriffsschicht führt die Anfragen an die von den Verfahren angebotenen Services aus. Hierzu nutzt die Service-Zugriffsschicht den ESB. Die Service-Zugriffsschicht übernimmt das Marshalling der Anfrage und das Demarshalling der Antwort sowie die Authentifizierung. Implementiert werden die Service-Aufrufe in Form von Web Service Calls.

Parallel zu den Schichten liegen schichtenübergreifende Funktionalitäten wie die Fehlerbehandlung und das Logging. ROBASO verfügt zudem über eine Datenhaltung, in welcher Konfigurationsdateien und fachliche Texte abgelegt und verwaltet werden.

5.1.3.2 Middleware-Architektur

Die Service-Architektur der Bundesagentur für Arbeit umfasst in der finalen Ausbaustufe mehr als 13.400 Anwender. Eine monolithische Architektur mit einem zentralen ESB ist in dieser Größenordnung weder zuverlässig implementierbar noch wartbar. Die Entscheidung in der Middleware-Architektur fiel deshalb auf sogenannte Middleware-Zellen.

Eine Middleware-Zelle ist eine in sich größtenteils autarke Umsetzung der gesamten Anwendungslandschaft der Bundesagentur für Arbeit. Sie enthält einen eigenen ESB, über den sämtliche Aufrufe der Anwendungen von außerhalb, sowie die gesamte Kommunikation innerhalb der Middleware-Zelle erfolgt. Eine Middleware-Zelle kann jede Anfrage an eine Anwendung der Bundesagentur beantworten, jedoch nur eine begrenzte Anzahl an parallelen Anfragen, mit einer Begrenzung von deutlich unter den geplanten 13.400.

Die Skalierung auf die geplante Nutzerzahl erfolgt durch die Parallelisierung mehrerer Middleware-Zellen. Ein Hardware-Loadbalancer verteilt die Nutzeranfragen auf die einzelnen Middleware-Zellen, wodurch eine Reduktion der Last auf den einzelnen Zellen erreicht wird. Alle Zellen sind technisch baugleich, wodurch eine gleichmäßige Auslastung sowie eine Standardisierung zur Verbesserung der Wartbarkeit erreicht werden.

Die Kommunikation zwischen den einzelnen Middleware-Zellen erfolgt über eine Kopplung der ESB-Instanzen der Zellen. Um die hohe Bandbreite und die geringe Latenz innerhalb einer Zelle auszunutzen werden zellenübergreifende Anfragen vermieden, eine Anwendung innerhalb einer Zelle nutzt bei Abhängigkeiten die anderen Anwendungen innerhalb derselben Zelle.

Die Datenhaltung in Form eines Datenbank-Clusters erfolgt außerhalb der Middleware-Zellen. Die Middleware-Zellen enthalten ausschließlich die Anwendungsschicht, ein Persistieren von fachlichen Daten innerhalb der Zelle geschieht nicht. Zur Datenhaltung greifen die Zellen auf das gemeinsame Datenbank-Cluster zu. Hierdurch ist es möglich, eine Nutzeranfrage an eine beliebige Zelle weiterzuleiten. Innerhalb einer User-Session ist der Load Balancer *sticky*, das heißt, er leitet alle zu der Session gehörende Kommunikation an denselben Server weiter. Dies geschieht durch das verwendete T3-Protokoll.

In dieser Arbeit erfolgt die Performance-Analyse von ROBASO auf einer Middleware-Zelle. Dies entspricht dem Lasttest-Vorgehen bei ROBASO und ermöglicht so vergleichende Analysen.

5.1.3.3 Hardware-Architektur

Das Konzept der Middleware-Zellen erlaubt den Verzicht auf große monolithische Serversysteme. Die Cluster-Fähigkeit der WebLogic Application Server über mehrere Blades hinweg ermöglicht den Einsatz von x86-Servern der Standardklasse, die Entscheidung fiel hier auf eine Realisierung mit Blade-Technologie (Chou 2003; Yang et al. 2002). Eine Middleware-Zelle wird repräsentiert durch ein Blade-Enclosure mit 10 Höheneinheiten, wodurch der Einbau von drei Zellen pro Rack möglich ist.

Eine Middleware-Zelle besitzt in der Summe 128 CPU sowie 1,5 Terabyte Hauptspeicher. Die interne Kommunikation erfolgt über ein Netzwerk mit 20 Gbit/s. Gekühlt wird das Blade-Enclosure mit einer in die Rückwand eingelassenen Wasserkühlung.

5.1.4 Zugriffsverhalten

Die ROBASO-Anwendung wird in der Endausbaustufe von ca. 13.400 Anwendern genutzt. Diese Anwender verteilen sich jedoch nicht gleichverteilt über die 14 Geschäftsprozesse, die von ROBASO implementiert werden. Für die Betrachtung des Zugriffsverhaltens auf den Terminierungsprozess ist damit ein anderer Ansatzpunkt notwendig.

Es ist davon auszugehen, dass sich durch die Implementierung von ROBASO das Kundenverhalten (Kunden sind in diesem Fall Arbeitssuchende und Interessenten an einer Berufsberatung) nicht ändert. Analysen der ATV-Nutzung ergeben, dass von etwa 250.000 Terminen pro Woche ausgegangen werden kann. Wird von einer Gleichverteilung über die Wochentage ausgegangen, so ergibt dies einen Wert von 50.000 Terminen pro Tag.

Über den Tag hinweg ergeben sich Stoßzeiten, zu denen die Kundeninteraktion ihren Höhepunkt erreicht. Die Hauptzeit für Kundeninteraktionen ist zwischen 8 und 12 Uhr, mit der Spitze zwischen 10 und 11 Uhr. Zu dieser Zeit geschehen 20 Prozent der gesamten Kundeninteraktionen eines Tages. Abbildung 5.6 stellt die Zugriffsverteilung innerhalb eines Tages grafisch dar.

Zur Betrachtung des Performance-Verhaltens ist insbesondere die maximale Anzahl paralleler Nutzer von Bedeutung. Führt man die oben begonnene Rechnung fort, so ergibt sich zu den Hochlastzeiten eine Spitzenlast von 10.000 Terminen pro Stunde. Nimmt man nun wieder eine Gleichverteilung über die Stunde an, so resultiert dies in 167 Terminen pro Minute. Ein durchschnittlicher Terminierungsprozess dauert in etwa 180 Sekunden, wodurch sich eine minimale Spitzenlast von 500 parallelen Anwendern ergibt. Diese minimale Spitzenlast setzt jedoch die genannten Gleichverteilungen voraus. In der Realität ist diese Gleichverteilung sehr unwahrscheinlich, so dass ein Sicherheitsaufschlag auf 600 parallele Anwender zu einem realistischen Ergebnis führt. Verteilt man diese 600 parallelen Anwender auf die vier Middleware-Zellen ergibt sich ein Zugriffsverhalten von 150 parallelen Anwendern pro Zelle.

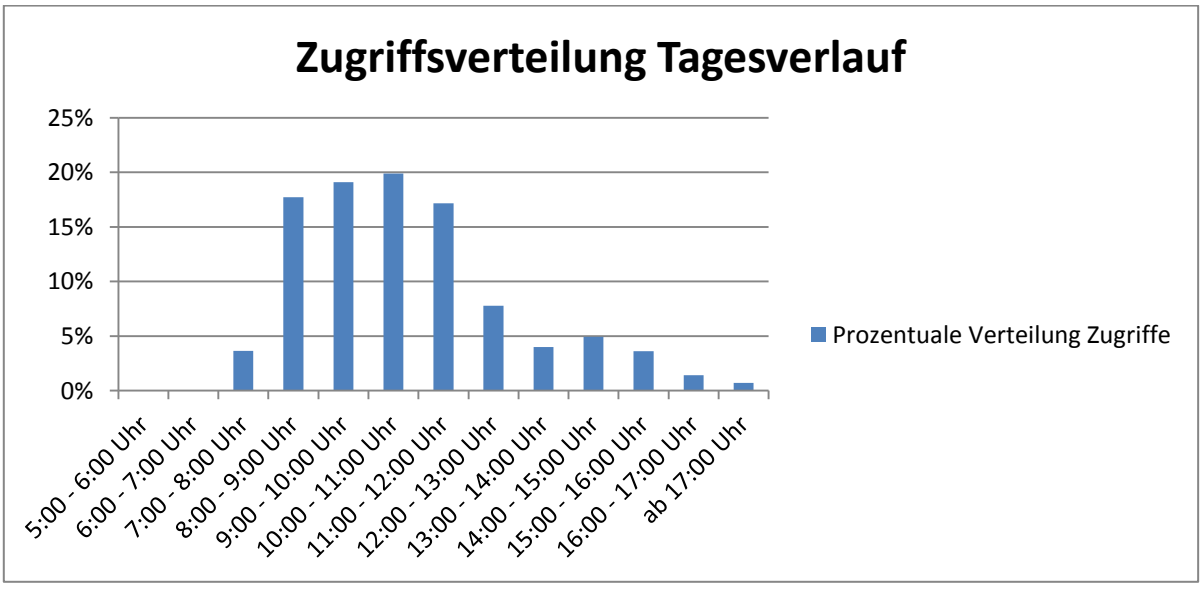


Abbildung 5.6. Zugriffsverteilung auf ROBASO im Tagesverlauf. Quelle: Eigene Darstellung.

5.1.5 Zusammenfassung

Dieses Teilkapitel stellt den Anwendungsfall vor, der die Basis der Experimentdurchführung bildet. Der Terminierungsprozess des Projekts ROBASO, durchgeführt von der Bundesagentur für Arbeit, eignet sich gut für die Experimentdurchführung, da er auf einer Service-orientierten Architektur basiert und damit stark komponentenorientiert ist. Weiterhin befindet sich der Terminierungsprozess zum Zeitpunkt dieser Arbeit in der Umsetzung und im Test, so dass alle für die Experimentdurchführung erforderlichen Messwerte erhoben werden können.

Um ein umfassendes Verständnis des Anwendungsfalls zu ermöglichen beschreibt das Teilkapitel den Aufbau des Terminierungsprozesses, die verwendeten Verfahren und Services, inklusive der umgesetzten Composed Services, und geht anschließend auf die Software- und Hardware-Architektur der ROBASO-Anwendung ein. Abgeschlossen wird das Teilkapitel mit einer Betrachtung des zu erwartenden Zugriffsverhaltens auf den Terminierungsprozess. Dieses Zugriffsverhalten bildet die Grundlage für das Usage Model der Simulation.

5.2 Messung des Performance-Verhaltens

Die komplexe Software- und Hardwarearchitektur der ROBASO-Anwendung sowie die Verteilung der verwendeten Services über mehrere operative Bereiche der Bundesagentur für Arbeit hinweg macht eine Aufteilung der Messungen des Performance-Verhaltens in drei Bereiche notwendig.

Die besten Messmöglichkeiten bietet die ROBASO-Oberflächenanwendung. Die Oberflächenanwendung ist eine Neuimplementierung basierend auf dem ADF-Framework und befindet sich zum Zeitpunkt dieser Arbeit in der Entwicklungs- und Testphase. Dies hat zur Folge, dass Zugriff sowohl auf die Architektur und den Programmcode der Oberflächenanwendung als auch organisatorisch auf die Architekten und die Entwickler besteht. Eine Einsicht in die inneren Strukturen der ROBASO-Oberflächenanwendung ist damit möglich und damit auch eine Messung auf detaillierter Weise.

Deutlich weniger bis gar keine Einsicht ist in die inneren Strukturen der verwendeten Services möglich. Die Services werden von organisatorischen Einheiten in der Bundesagentur für Arbeit entwickelt und betreut, die größtenteils von der ROBASO-Entwicklung getrennt operieren. Zudem bestehen einige der die Services bereitstellenden Anwendungen bereits seit vielen Jahren als Legacy-System und wurden durch Service-

Fassaden der SOA verfügbar gemacht. Eine Messung des Performance-Verhaltens der Services ist nur an deren Schnittstellen möglich.

Zuletzt erfolgt zudem eine Messung des Performance-Verhaltens der Middleware-Komponenten. Hier wird speziell der Oracle Service Bus (OSB, (Oracle 2012d)) als verwendete ESB-Implementierung vermessen. Der OSB kommt bei jedem Service-Aufruf mindestens einmal zum Einsatz, so dass ein starker Einfluss auf die Gesamt-Performance der Anwendung zu vermuten ist. Bei der Messung des Performance-Verhaltens des OSB wird primär auf die von Oracle mitgelieferten Werkzeuge zurückgegriffen.

5.2.1 Messung des Performance-Verhaltens der ROBASO-Oberflächenanwendung

Die ROBASO-Oberflächenanwendung liegt sowohl in Form von Programmcode als auch auf einer Testumgebung ausführbar zur Analyse vor und erlaubt damit die Messung von detaillierten Performance-Informationen. Aufgrund der Struktur der Anwendung, der Verwendung des ADF-Taskflow-Frameworks und der Ausführung auf dem Application Server ist es mit den im Projekt verfügbaren Mitteln nicht möglich, Aussagen über den Ressourcenverbrauch der Einzelkomponenten auf CPU-Basis zu erheben, wie dies beispielsweise in (Kounev/Buchmann 2003) und (Kounev 2006) beschrieben wird. Im folgenden Abschnitt 5.4.2 wird auf diese Schwierigkeit näher eingegangen. Eine Darstellung der ROBASO-Oberflächenanwendung dem Grundkonzept von PCM folgend (Becker et al. 2007; Brosig et al. 2009) ist damit nicht möglich.

Im Rahmen des Lasttests konnten jedoch Antwortzeit- und Laufzeitinformationen zu den einzelnen Komponenten der ROBASO-Anwendung ermittelt werden. Diese Informationen dienen als Basis für die Abbildung des Performance-Verhaltens im Simulationsmodell.

5.2.2 Messung des Performance-Verhaltens der Services

Im Gegensatz zur ROBASO-Oberflächenanwendung stehen bei den verwendeten Services weder Architektur- oder Implementierungsdetails noch Instrumentierungsmöglichkeiten zu Verfügung. Einzig über den Einsatz von Lasttests sowie die Analyse der Logfiles einiger Service-anbietender Verfahren lassen sich Informationen zum Performance-Verhalten der Services erlangen.

Da weder bekannt ist, welches Ressourcennutzungsverhalten die Verfahren hinter den Services besitzen, noch auf welcher Hardware die Verfahren betrieben werden, ist eine Ressourcennutzungsbetrachtung der Services hinfällig. Die Messung des Performance-

Verhaltens muss an der Service-Schnittstelle erfolgen, denn diese bildet den einzigen Ansatzpunkt für eine Messung.

An der Service-Schnittstelle können zwei Performance-Messwerte erhoben werden: der Durchsatz und die Antwortzeit. Beide Performance-Messwerte können mittels einfacher Aufrufe (beispielsweise in Form eines Last- oder Stresstests) gemessen werden und bedürfen keiner Instrumentierung oder sonstiger Eingriffe in die Anwendung selber. Da sich die vorliegende Arbeit mit der Antwortzeit des Gesamtsystems befasst fällt die Wahl des zu erhebenden Performance-Messwerts der Services auf die Antwortzeit.

Die Antwortzeit eines Services wird erhoben, indem die Zeit vom Versenden einer Anfrage bis zum Empfangen der Antwort auf die Anfrage gemessen wird (siehe Abschnitt 2.5.2.1). Im vorliegenden Fall erfolgt die Messung nicht vom Client aus, sondern vom OSB, da die Laufzeit des OSB separat abgebildet wird (siehe hierzu den folgenden Abschnitt 5.2.3).

Auf die Antwortzeit eines Services haben verschiedene Faktoren Einfluss:

- Die *Funktionalität*, welche die aufgerufene Service-Operation implementiert.
- Die *Parameter*, mit welchen die Service-Operation aufgerufen wird.
- Die Anzahl der *parallelen Aufrufe* auf dem Service.
- Die *Last*, die zum Zeitpunkt des Service-Operationsaufrufs auf den *zugrundeliegenden Ressourcen* (Hardware, Datenbank, etc.) besteht.

Es ist ersichtlich, dass der letzte Punkt in der Messung auf Antwortzeitbasis zu einer statistischen Varianz führt. Da an der Service-Schnittstelle keine Informationen über den zugrundeliegenden Systemzustand vorliegen variieren die Antwortzeiten bei gleichen Übergabeparametern und gleicher Anzahl paralleler Aufrufe, verursacht durch den (nicht sichtbaren) internen Zustand des dem Service zugrundeliegenden Systems. In der diesem Experiment zugrundeliegenden produktivnahen Testumgebung ist es jedoch möglich, die äußeren Umstände konstant zu halten, so dass bei dem in dieser Arbeit beschriebenen Experiment der Systemzustand vernachlässigt werden kann.

Nachdem die von den Service-Operationen implementierte Funktionalität im Experiment konstant bleibt sind die Übergabeparameter sowie die Anzahl der parallelen Aufrufe auf einem Service die Variablen, welche die Messung der Antwortzeit beeinflussen.

Anders als bei der Betrachtung der Middleware (im folgenden Abschnitt) ist eine Betrachtung der durchschnittlichen Antwortzeit der Service-Operationen nicht zielführend. Die vorab aufgeführten Einflussfaktoren *Parameterwerte* und *Anzahl paralleler Aufrufe auf dem Service* verursachen eine maßgebliche Änderung der Antwortzeit einer Service-Operation. Da die betrachteten Service-Operationen (bis auf die Mengen-Suchoperationen) recht simple Übergabeparameter definieren ist es dabei insbesondere die Anzahl der parallelen Zugriffe, die das Antwortzeitverhalten beeinflusst. Dies wird im Folgenden weiter ausgeführt.

Um das Antwortzeitverhalten eines Services korrekt abbilden zu können werden also Messungen jeder Service-Operation benötigt. Zur korrekten Abbildung des Antwortzeitverhaltens der jeweiligen Service-Operationen wiederum erfolgen Antwortzeitmessungen für alle Kombinationen von Parameterwerten und Anzahl paralleler Aufrufe. Da die potentiellen Parameterwerte oftmals unbegrenzt viele sind (beispielsweise schon bei einem ganzzahligen, unbegrenzten Übergabewert) erfolgt eine Bildung von Äquivalenzklassen, ähnlich dem Programmtest (vgl. (Ludewig/Lichter 2007b)). Die Anzahl der parallelen Aufrufe erfolgt vergleichbar in sinnvollen Schritten.

Oftmals ist aufgrund der Verfügbarkeit des Services oder aufgrund anderer Einschränkungen ein solch systematisches Vermessen nicht möglich. In einem solchen Fall bedarf es einer Analyse bestehender Messwerte, beispielsweise aus Logfiles und Monitoring-Daten oder aus Lasttests. Diese sind generell weitaus weniger strukturiert als die Messdaten aus expliziten Antwortzeitmessungen, dienen jedoch auch als Basis für die Modellierung. Im vorgestellten Anwendungsfall sind solche Einschränkungen gegeben, wie später weiter ausgeführt wird.

5.2.3 Messung des Performance-Verhaltens der Middleware

Für die Messung des Performance-Verhaltens der Middleware wird auf die von Oracle bereitgestellten proprietären Analysewerkzeuge zurückgegriffen. Primär werden die Laufzeiten über den OSB mittels des Enterprise Managers (Oracle 2012a) ermittelt. Der Enterprise Manager ist eine Monitoring-Lösung, die im Hintergrund kontinuierliche Laufzeitüberwachung auf dem OSB durchführt. Mittels vordefinierter Abfragen lassen sich für einen beliebigen Zeitraum Analysen über die Laufzeiten einzelner OSB-Strecken abfragen.

Lasttests von Seiten der Bundesagentur für Arbeit haben ergeben, dass der OSB auf der eingesetzten Hardware für alle zu erwarteten Last-Szenarien im Niedriglastbereich arbeitet. Analysen bisheriger Enterprise Manager-Informationen zeigen ebenfalls eine nahezu konstante Laufzeit des OSB. Die Modellierung der OSB-Laufzeiten in diesem Experiment erfolgt damit, basierend auf den gegebenen Laufzeitinformationen, als konstanter Durchschnittswert.

Auswertungen der OSB-Lasttestinformationen sowie der Enterprise Manager-Monitorinformationen von bereits über den OSB laufenden Anwendungen haben eine durchschnittliche Laufzeit von 13 Millisekunden pro OSB-Strecke ergeben, mit wenigen Prozent Varianz. Diese Streckenlaufzeit enthält die Authentifizierung mittels des Moduls WESPE 2 (Web Service Security and Policy Enforcement), einem auf dem OSB mittels des Oracle Web Services Manager (OWSM, (Oracle 2012e)) implementierten Regelsatzes zur Überprüfung der Zugriffsberechtigungen auf Web Services.

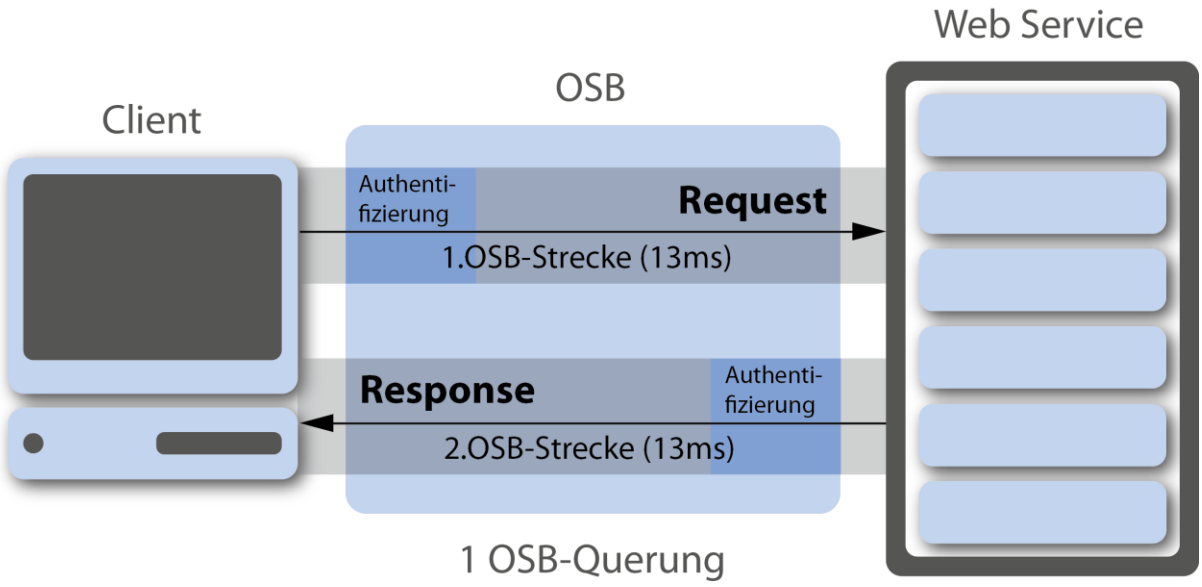


Abbildung 5.7. Grafische Darstellung von OSB-Strecke und OSB-Querung. Quelle: Eigene Darstellung.

Die Gesamtlaufzeit einer Web Service-Anfrage mit Rückantwort über den OSB beträgt damit 26 Millisekunden (13 Millisekunden für die Anfrage sowie 13 Millisekunden für die Rückantwort). Abbildung 5.7 illustriert dies. Dieser OSB-Overhead kommt bei jedem Web Service-Aufruf zum Tragen. Insbesondere bei der Modellierung der Composed Services (siehe Abschnitt 5.1.1.3) ist zu beachten, dass hier mehrere OSB-Querungen mit je 26 Millisekunden erfolgen.

5.2.4 Zusammenfassung

In den vorherigen Abschnitten wurden die Ansätze zur Messung des Performance-Verhaltens der drei Hauptkomponenten der ROBASO-Anwendung vorgestellt. Die drei Komponenten *ROBASO-Oberflächenanwendung*, *Services* und *Middleware* bieten unterschiedliche Möglichkeiten und Ansätze für die Messung der für die Simulation benötigten Performance-Informationen. Während die sich im Entwicklungsstadium befindende ROBASO-Oberflächenanwendung eine nahezu vollständige Instrumentierung und Analyse erlaubt, stehen die verwendeten Services einzig in Form von aufrufbaren Schnittstellen bereit. Zur Analyse der Performance des verwendeten OSB als Middleware-Komponente wiederum werden proprietäre Analysewerkzeuge des Anbieters eingesetzt.

Abhängig von dem verwendeten Vorgehen zur Messung werden von den drei Komponenten unterschiedliche Performance-Messwerte erhoben. Während die ROBASO-Anwendung Performance-Messwerte für die einzelnen Anwendungskomponenten erlaubt, beschränken sich die Performance-Informationen der Services auf Antwortzeiten in Abhängigkeit von Parameterwerten und der Anzahl paralleler Anfragen. Die Middleware wiederum bildet einen Sonderfall. Lasttests und Performance-Untersuchungen haben hier ergeben, dass sich die Laufzeit in allen gemessenen Fällen konstant mit wenigen Prozent Abweichung um einen festen Wert, 13 Millisekunden pro Strecke, bewegt. Die Laufzeit der Middleware wird damit, zur Vereinfachung des nachfolgend beschriebenen Modells, als konstant angenommen.

Die durchgeführten Messungen bilden die Grundlage für die im Folgenden vorgestellte Modellierung und Simulation.

5.3 Anwendung des evolutionären Algorithmus zur Modellierung

Die im vorherigen Teilkapitel vorgestellten Methoden zur Messung des Performance-Verhaltens der drei Hauptkomponenten von ROBASO sowie derer Teilkomponenten resultieren in drei verschiedenen Mengen an Performance-Messdaten. Die erste Menge, die für die ROBASO-Oberflächenanwendung gemessenen Antwortzeitinformationen, sowie die dritte Menge, die konstante Antwortzeit des OSB, lassen sich ohne Adaption im PCM-Framework abbilden (vgl. Kapitel 4.3 sowie (Reussner et al. 2007)). Die tatsächliche Herausforderung bildet die Modellierung und Einbindung der zweiten Menge an Messdaten, die Antwortzeitmessungen der Services. Diese können zwar durch die *DataTable*-PCI-Implementierung als Menge an Messdaten in die Simulation mit einbezogen werden, diese Integration bringt jedoch, gerade bei großen Systemen, gravierende Probleme mit sich (siehe Teilkapitel 4.3.1). In diesem Teilkapitel wird nun gezeigt, wie der in Kapitel 4.2 beschriebene evolutionäre Algorithmus *Mendel* verwendet wird, um das Performance-

Verhalten der von ROBASO verwendeten Services zu modellieren. Die auf diese Weise generierten Komponentenmodelle werden über die *Formula*-PCI-Implementierung in die Simulation eingebunden (vgl. hierzu Abschnitt 4.3.2).

5.3.1 Ziel

Ziel der Modellierung durch den evolutionären Algorithmus ist die Erzeugung mathematischer Formeln für die im Anwendungsfall ausgeführten Service-Operationen, die das Antwortzeitverhalten der Service-Operationen in Abhängigkeit von der Anzahl paralleler Aufrufe sowie gegebenenfalls von Übergabeparametern beschreiben.

Die Wahl des akzeptierten Modellfehlers ist im Fall der Modellierung des Performance-Verhaltens der Service-Operationen eine Herausforderung. Da die Performance-Messwerte der Services aus verschiedenen Quellen stammen (Lasttest-Messwerte sowie die verschiedenen Log- und Monitoring-Verfahren einzelner Service-Implementierungen) ist von unterschiedlicher Datenqualität auszugehen. Je nach Datenqualität ist ein Modellfehler von fünf Prozent akzeptabel (Tertilt/Krcmar 2012), aber auch von zehn bis fünfzehn Prozent (Tertilt/Krcmar 2011). Für das in dieser Arbeit präsentierte Experiment wird mit einem akzeptierten Modellfehler von sieben Prozent ein Mittelwert gewählt. Durch die Wahl dieses Modellfehlerwerts werden nur Komponentenmodelle akzeptiert, die nah an den gemessenen Werten liegen. Dennoch erfolgt eine Konvergenz (und damit ein Ende des Modellierungslaufes) in akzeptabler Zeit, selbst wenn die Messdatensätze Messfehler enthalten.

5.3.2 Konfiguration

Die Konfiguration des evolutionären Algorithmus wirkt sich maßgeblich auf das Modellierungs- und Konvergenzverhalten des Algorithmus aus (vgl. Kapitel 4.4). Eine nicht optimal gewählte Konfiguration wirkt sich entweder dadurch aus, dass der evolutionäre Algorithmus kein brauchbares Modell für die gegebenen Daten findet (beispielsweise, weil er in einem lokalen Optimum verharrt), oder dass er nicht konvergiert (beispielsweise, weil er von einem möglichen Optimums-Ansatz zum nächsten springt, ohne zum Optimum zu konvergieren). In beiden Fällen liefert der evolutionäre Algorithmus keine brauchbare Lösung.

Um diese Probleme zu vermeiden wurde in Kapitel 4.4 eine optimale Konfiguration ermittelt. Diese wird für das in dieser Arbeit durchgeführte Experiment verwendet. Hierdurch ergibt sich die folgende Konfiguration für *Mendel* (Tabelle 7):

Tabelle 7. Konfiguration des evolutionären Algorithmus zur Modellierung der Services.

Konfigurationsparameter	**Wert**
Populationsgröße	5.000
Genome-Länge	41
Mutationswahrscheinlichkeit in %	90
Crossover-Wahrscheinlichkeit in %	75
Parental Population Quota	30

Diese statistisch und mittels des Meta-GA ermittelte optimalen Konfiguration für den evolutionären Algorithmus bildet damit die Konfiguration, die mit maximaler Wahrscheinlichkeit zu guten Modellierungsergebnissen führt. Um die Umgebungsvariablen konstant zu halten erfolgt die Ausführung des evolutionären Algorithmus zur Modellierung in diesem Anwendungsfall auf der identischen Hardware (vgl. Abschnitt 4.4.2.4).

5.3.3 Datensätze

In Teilkapitel 5.2 wurde beschrieben, dass die ROBASO-Oberflächenanwendung auf Ressourcenebene vermessen wird, die verwendeten Services auf Antwortzeitebene, und die Abbildung der Middleware in der Simulation als Konstante erfolgen kann. Die Modellierung der Komponentenmodelle durch den evolutionären Algorithmus erfolgt damit ausschließlich auf den verwendeten Services. Die ROBASO-Oberflächenanwendung sowie die Middleware werden auf konventionelle Weise im PCM-Simulationsframework modelliert und bilden damit einen Teil des stabilen Experimentaufbaus.

Für die Modellierung des Antwortzeitverhaltens der Services kommen Messdaten aus den Lasttests der Services sowie aus dem ROBASO-Lasttest zum Einsatz. Für die einzelnen Services stehen folgende Messdatensätze zu Verfügung (Tabelle 8):

Tabelle 8. Übersicht über die zur Modellierung zu Verfügung stehenden Messdatensätze der Services.

Service	**Operation**	**Anzahl Messdaten**
PersonenBetreuerService	PersonenBetreuerAuflisten	4092
VermerkeService	VermerkLesen	6120
VermerkeService	VermerkeAuflisten	6120
VermerkeService	VermerkErstellen	8184

Aus Tabelle 8 ist ersichtlich, dass der Umfang der Messdatensätze für die einzelnen Services unterschiedlich ausgeprägt ist. Dies ist darauf zurückzuführen, dass jedes Verfahren die Lasttests eigenständig und in Eigenregie durchführt. Der in diesem Kapitel betrachtete Anwendungsfall ist ein Beispiel aus der Praxis, so dass dies nicht als Nachteil, sondern als korrekte Abbildung der realen Projektsituation zu betrachten ist.

Weiterhin liegen nicht für alle Service-Operationen verwendbare Antwortzeitinformationen zur Modellierung vor. Von den 47 aufgerufenen Service-Operationen stehen gerade einmal für die vier in Tabelle 8 genannten Operationen ausreichend Antwortzeitinformationen zu Verfügung. Dies ist auf die Messdatenaggregation zurückzuführen, die während und nach den Lasttests durchgeführt werden, und im Zuge derer die gemessenen Daten auf abstrakte Kennzahlen wie die durchschnittliche und maximale Antwortzeit über einen definierten Zeitraum komprimiert werden. Auf diesen aggregierten Daten ist eine Modellierung nicht möglich, die entsprechenden Service-Operationen werden in der Simulation konventionell über eine Normalverteilung abgebildet.

5.3.4 Ergebnisse

Auf den vorab vorgestellten Messdatensätzen der Service-Operationen wird der evolutionäre Algorithmus *Mendel* mit der in Teilkapitel 4.4 ermittelten optimalen Konfiguration ausgeführt. Bereits nach einer Modellierungszeit von unter einer halben Stunde ergeben sich Komponentenmodelle mit einem Fehlerwert unter 6 Prozent.

Abbildung 5.8 zeigt exemplarisch das Antwortzeit-Komponentenmodell der Operation *VermerkLesen* des VERBIS-Services *VermerkeService*. Die zugrundeliegenden Antwortzeitdaten stammen aus einem von VERBIS initiierten Stresstest mit bis zu 1.300 parallelen Anwendern. In den Lasttests sind für die simulierten Anwender jedoch Think Times definiert, sie greifen nicht kontinuierlich auf das zu testende System zu. Unter Anwendung von Little’s Law (Menasce et al. 2004) ergibt sich hierdurch eine Antwortzeitkurve bis zu 300 parallelen Zugriffen mit einem Fehlerwert von 5,03 Prozent. Die nahezu lineare Antwortzeitkurve ergibt sich aus dem Processor-Sharing-Verhalten der zugrundeliegenden Implementierung. Die Randwertbetrachtung zeigt, dass die zugrundeliegende Hardware mit den gemessenen parallelen Anfragen nicht ausgelastet ist. Anderenfalls würde die Kurve im Hochlastbereich (am rechten Ende) stark ansteigen.

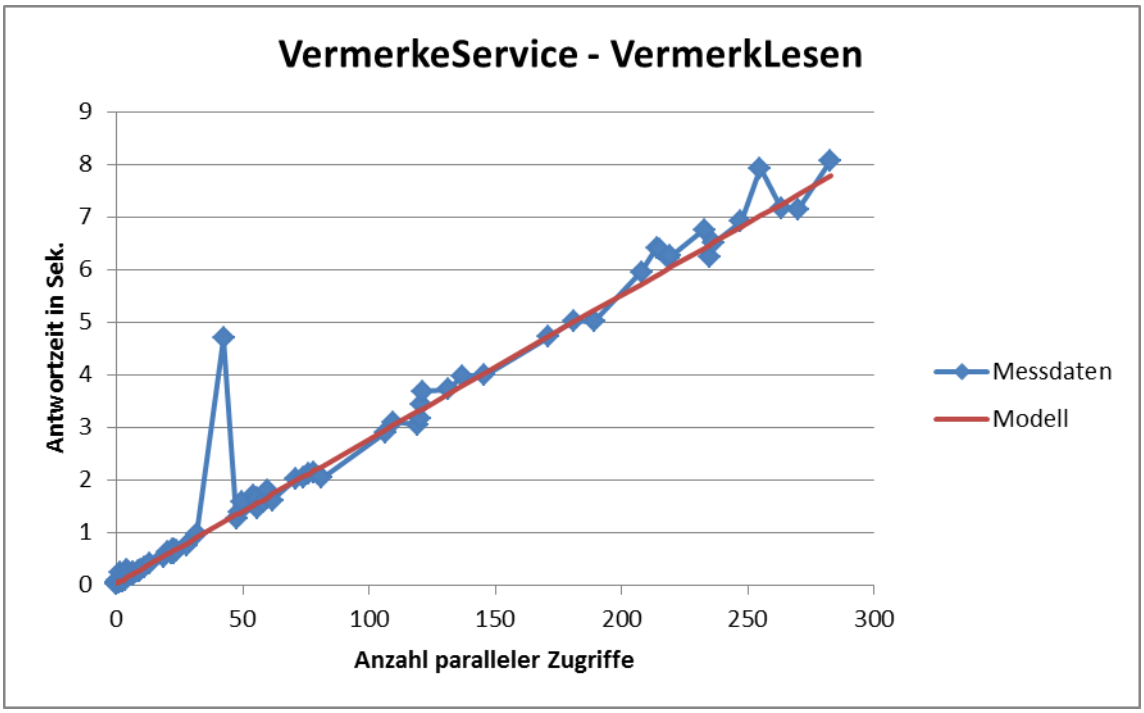


Abbildung 5.8. Antwortzeit-Komponentenmodell der Service-Operation VermerkeService.VermerkLesen. Quelle: Eigene Darstellung.

Formel 14 stellt das generierte Antwortzeit-Komponentenmodell in Form einer mathematischen Formel exemplarisch für alle weiteren generierten Komponentenmodelle dar.

Formel 14. Formel-Darstellung des Antwortzeit-Komponentenmodells der Operation VermerkLesen des VermerkeService.

$$-0{,}033 * (0{,}3749 + X0 + 1{,}2434 * \sin(1{,}8051 - \sin(1{,}4729 - X0 - \frac{-0{,}0578}{\sin\left(X0 - \sin\left(1{,}0322 + \sin\left(\frac{0{,}9684}{\left(1{,}0322 + \sin(|-0{,}9905|)^{\frac{-0{,}3908}{\sin(X0 + \sin(2{,}6159 * |X0|^{-1{,}4996}))}}\right)}\right)\right)\right)^{-1{,}4919}})))$$

$$-1{,}2345$$

Für die anderen in Tabelle 8 aufgeführten Service-Operationen ergeben sich ebenfalls verwendbare Komponentenmodelle in dem Modellierungszeitraum. Diese sind in Anhang D dargestellt.

Auch in der Praxis erweist sich die evolutionären Algorithmen zur Modellierung der Komponentenmodelle damit als effektiv. Die Konfiguration nach Teilkapitel 4.4 sorgt für

schnelle Resultate. Dass sich das Performance-Verhalten der untersuchten Service-Operationen durch eine einfachere Funktion darstellen ließe ist dabei nicht von Belang, da hierzu Vorwissen nötig wäre. In einem Ansatz, der den manuellen Aufwand minimiert, kann dieses Vorwissen jedoch nicht vorausgesetzt werden. Bei der Betrachtung einer großen Unternehmensanwendung würde der Einsatz von Vorwissen eine Vorab-Analyse aller Messdaten der Service-Operationen verlangen. Der dafür notwendige manuelle Aufwand rechtfertigt die potentielle Vereinfachung der generierten Modelle in keiner Weise. Und auch auf die Simulationszeit wirkt sich die Komplexität der generierten Formel nicht merklich aus, wie Abschnitt 4.3.4 zeigt.

5.3.5 Zusammenfassung

Die in diesem Teilkapitel beschriebene Anwendung des evolutionären Algorithmus auf den Antwortzeit-Messdaten der Service-Operationen zeigt, dass dieser auch auf realen Messdaten effektiv modelliert. Innerhalb von 30 Minuten generiert der Algorithmus mit der identifizierten optimalen Konfiguration Antwortzeit-Komponentenmodelle für die Service-Operationen mit einem Fehlerwert von 5 Prozent. Die generierten mathematischen Modelle sind dabei nicht intuitiv, die grafische Darstellung zeigt jedoch, dass die Messwerte sehr gut angenähert werden. Weiterhin werden klar erkennbare Messfehler nivelliert. Die generierten Komponentenmodelle eignen sich damit gut für den Einsatz in der Performance-Simulation.

5.4 Performance-Simulation

Die Experimentausführung des kontrollierten Softwareexperiments zur Überprüfung der in Kapitel 1.2 aufgestellten Annahme 1 umfasst zwei Simulationsdurchläufe. Die veränderliche Variable des Experiments ist dabei die Verwendung der durch den evolutionären Algorithmus generierten Komponentenmodelle.

Als Basis für das Experiment dient der vorab vorgestellte Anwendungsfall *Terminierung vornehmen*. Der Anwendungsfall ist komplett in PCM abgebildet und bleibt über beide Simulationsdurchläufe stabil. Einzig die Modellierung des Antwortzeitverhaltens der verwendeten Service-Operationen erfährt eine Änderung – die Einbindung der Komponentenmodelle.

Im ersten Fall (ohne Komponentenmodelle) erfolgt die Modellierung des Antwortzeitverhaltens der Services traditionell in Form einer Normalverteilung um die durchschnittliche Antwortzeit. Dies ist die gängige Art der Modellierung einer Komponente, die nicht über ihre Ressourcennutzung abgebildet werden kann, und dient als Vergleichswert zur Ermittlung der Veränderung beim Einsatz der Komponentenmodelle.

Im zweiten Fall (mit Komponentenmodellen) werden die Service-Operationen durch Komponentenmodelle abgebildet (vgl. Kapitel 2.3). Die vom evolutionären Algorithmus generierten Komponentenmodelle bilden das Antwortzeitverhalten der Service-Operationen in Abhängigkeit der Übergabeparameter und der parallelen Aufrufe auf dem Service ab. Ihre Generierung erfolgt aus Antwortzeitmessungen durch Lasttests und die Auswertung von Monitoring-Informationen.

5.4.1 Aufbau

Das PCM-Simulationsmodell des Anwendungsfalls umfasst, wie jedes PCM-Simulationsmodell, fünf Modelltypen (vgl. (Reussner et al. 2007)):

- **Usage Model** – Das Usage Model beschreibt das Nutzerverhalten der auf das simulierte System zugreifenden Anwender.

- **System Model** – Die Schnittstellen des Systems nach außen sowie die Aufrufbeziehungen innerhalb des betrachteten Systems auf Komponentenebene werden im System Model definiert.

- **Repository Model** – Das Repository Model enthält die Definition der Komponenten des betrachteten Systems sowie deren Methoden. Zudem wird im Repository Model definiert, welche Komponente welches Interface implementiert und nutzt.

- **Resource Environment Model** – Die Hardware-Umgebung, auf welcher das betrachtete System betrieben wird, ist im Resource Environment Model definiert. Das Modell umfasst die verfügbaren Server mit CPU- und HDD-Definition sowie das Netzwerk, welches die Server verbindet.

- **Allocation Model** – Das Allocation Model definiert, welche Komponente auf welchem Server des Resource Environment Model betrieben wird.

Neben diesen Hauptmodellen umfasst das Simulationsmodell des Anwendungsfalls zudem für jede Komponenten-Operation eine Service Effect Specification (SEFF), die das Verhalten der Komponente bei Aufruf dieser Operation in Bezug auf Ressourcennutzung oder Antwortzeitverhalten beschreibt.

5.4.2 Herausforderungen bei der Modellierung der ROBASO-Anwendung

Bei der Erstellung des Simulationsmodells der ROBASO-Anwendung trifft man auf vier große Herausforderungen:

- Die Komplexität des Systems, mit fünf Schichten allein in der ROBASO-Anwendung, den 47 Service-Operationen bereits im Terminierungsprozess sowie der Middleware führt zu einem im Editor kaum noch zu handhabbaren Simulationsmodell.

- Die Struktur der ROBASO-Anwendung als zentrale UI und Orchestrierungslösung für die verschiedenen Service-Aufrufe lässt sich in PCM nur schwer abbilden.

- Die für die Abbildung des Nutzerverhaltens notwendigen Pfadwahrscheinlichkeiten sind sehr schwer zu erlangen.

- Die Messung des Ressourcenverbrauchs der ROBASO-Komponenten auf dem Application Server gestaltet sich problematisch.

Die ersten beiden Herausforderungen werden durch Limitierungen des PCM-Frameworks sichtbar. Die letzten beiden Herausforderungen bilden hingegen ein Problem der Datenerhebung. Im Folgenden werden die genannten Herausforderungen genauer beschrieben und Lösungsansätze zu ihrer Bewältigung skizziert.

5.4.2.1 Modellierung eines komplexen Systems in PCM

Die Erstellung der Simulationsmodelle erfolgt in PCM im grafischen Editor per Hand. Das Anlegen und Pflegen der großen Menge an Einzelkomponenten sowie ihre Beziehungen zueinander über diesen Editor ist sehr zeitaufwändig und fehleranfällig. Insbesondere das Repository und System Model nehmen manuell kaum mehr zu handhabende Dimensionen an, obwohl nur ein einziger Prozess abgebildet wurde. Zudem wird für jede Service-Operation und jede Aktion der ROBASO-Anwendung eine SEFF angelegt, wodurch im Anwendungsfall 89 SEFF-Dateien erstellt und gepflegt werden müssen.

5.4.2.2 Orchestrierung mittels einer zentralen Komponente

Die zweite Herausforderung ergibt sich aus der Struktur der ROBASO-Anwendung selber. Anders als in anderen komponentenbasierten Systemen existieren im Anwendungsfall keine gleichberechtigten Komponenten, die in Kooperation miteinander stehen. Vielmehr bildet die ROBASO-UI die zentrale Orchestrierung zahlreicher Service-Aufrufe. Bildet man diese Struktur in PCM ab, so resultiert dies in einer zentralen Komponente, die alle anderen

Komponenten nutzt. Abbildung C.1, aufgrund ihrer Größe in den Anhang verlagert, zeigt das Repository Model des Anwendungsfalls in der Gesamtansicht. Die zentrale ROBASO-UI-Komponente als Interface (oben) und Implementierung (unten) ist in der Mitte deutlich zu erkennen.

Die Bildung der zentralen Komponente ist architektonisch korrekt, bringt jedoch zwei Nachteile mit sich.

Zum einen erschwert sie die Verständlichkeit und Wartbarkeit des Modells, da in ihren Funktionen faktisch die gesamte Geschäftslogik abgebildet wird. Die eigentlichen Abläufe verbergen sich damit in den zahlreichen SEFFs der zentralen Komponente und sind aus den System und Repository Modellen nicht zu erkennen. Die Ablauflogik verteilt sich damit auf mehrere Dateien und ist nur schwer nachzuvollziehen.

Zum anderen führt die Bildung der zentralen Komponente zur Verschiebung eines Teils des Nutzerverhaltens aus dem Usage Model in die SEFFs der zentralen Komponente. Wie in Teilkapitel 5.1 beschrieben setzt sich der im Anwendungsfall betrachtete Hauptprozess „Terminierung vornehmen“ aus mehreren Teilprozessen zusammen. Das PCM Usage Model erlaubt jedoch keine hierarchische Definition der Nutzerverhaltensabläufe. Hierdurch werden im Usage Model nur die Schritte des Hauptprozesses abgebildet, während die Arbeitsschritte und Verzweigungen der einzelnen Teilprozesse in je einer Funktion der zentralen Komponente und der zugehörigen SEFF definiert werden. Die ROBASO-Struktur lässt sich auf diese Weise in PCM abbilden, die Komplexität des Modells erhöht sich jedoch weiter.

5.4.2.3 Ermittlung der Pfadwahrscheinlichkeiten

An Verzweigungspfaden verlangt PCM die Definition eine Wahrscheinlichkeit, mit welcher jeder Pfad vom Anwender eingeschlagen wird. Die Annahme einer Gleichverteilung über alle Pfade (oder eine beliebige andere Standard-Verteilung) ist unbegründet – für ein korrektes Simulationsmodell ist eine Ermittlung der Pfadwahrscheinlichkeiten für jede Verzweigung unerlässlich.

In der Praxis des Anwendungsfalls zeigt sich jedoch, dass diese Pfadwahrscheinlichkeiten zumeist nicht bekannt sind. Wie in Teilkapitel 5.1 ausgeführt werden die in ROBASO umgesetzten Hauptprozesse zum aktuellen Zeitpunkt vom Mitarbeiter der Eingangszone oder des Service Centers manuell über mehrere Einzelanwendungen hinweg durchgeführt. Für die einzelnen Prozesse gibt es Verfahrensanweisungen, die beschreiben, wie die

Einzelaktivitäten durchzuführen sind. Eine Analyse, welcher Durchlauf wie oft geschieht, existiert hingegen nicht.

Das Wissen über die Pfadwahrscheinlichkeiten existiert damit ausschließlich in den Köpfen der Mitarbeiter Eingangszone und Service Center. Doch selbst bei den Mitarbeitern existiert nur partielles Wissen, fokussiert auf den jeweiligen Einsatzbereich. Je nach Einsatzgebiet des Mitarbeiters ist durchaus von einer unterschiedlichen Erfahrung auszugehen: so wird in Regionen mit niedriger Arbeitslosenquote die Terminierung zum Arbeitslosengeld seltener geschehen als in Regionen mit hoher Arbeitslosigkeit. Die Erfassung korrekter Pfadwahrscheinlichkeiten ist damit eine ausgesprochen komplexe und aufwändige Aufgabe.

5.4.2.4 Ermittlung des Ressourcenverbrauchs der Anwendungs-Komponenten

Die ROBASO-Anwendung wird als Java Enterprise Edition Anwendung auf einem Application Server deployed. Neben der eigentlichen ROBASO-Anwendung umfasst das deployte Paket zudem verschiedene Bibliotheken. Die im Lasttest ermittelbaren CPU-Nutzungsinformationen liefern jedoch nur den CPU- und Speicher-Verbrauch der Java Virtual Machine im Ganzen – eine Differenzierung der gemessenen Informationen auf die einzelnen Bestandteile der Anwendung ist nicht möglich. Das automatische Garbage Collection der Java Virtual Machine erschwert zudem die Messung des Speicherverbrauchs: zu einem Messzeitpunkt ist nur schwer zu sagen, wie viel der gemessenen Speichernutzung auf tatsächlich benutzte Objekte zurück zu führen ist, und wie viel Speicher durch Garbage-Objekte belegt ist.

5.4.2.5 Herangehensweise zur Lösung der Herausforderungen

Die vorgestellten Herausforderungen sind grundsätzlicher Natur und damit schwer zu lösen. Im Rahmen dieser Arbeit wurden Wege gefunden, mit den Herausforderungen umzugehen. Diese werden im folgenden Abschnitt vorgestellt. Für den produktiven Praxiseinsatz eignen sich die dort vorgestellten Lösungswege jedoch nur bedingt. Aus diesem Grund werden im zweiten Absatz Lösungsskizzen für eine allgemeinere Lösung der vorgestellten Herausforderungen vorgestellt.

5.4.2.5.1 Lösungswege im Rahmen dieser Arbeit

Für die Bearbeitung des Anwendungsfalls im Rahmen der vorliegenden Arbeit wurden die beschriebenen Herausforderungen durch zwei Ansätze überwunden: Vereinfachung und manueller Aufwand.

Die erste Herausforderung – die Modellierung eines komplexen Systems – wurde im Rahmen des Projekts durch einen großen Einsatz manueller Arbeit am PCM-Editor überwunden.

Bei der zweiten Herausforderung – die Orchestrierung mittels einer zentralen Komponente – wurde so weit wie möglich vereinfacht. So wurde auf die Abbildung der einzelnen Schichten der ROBASO-Anwendung verzichtet und diese zu einer Komponente zusammengefasst. Diese Vereinfachung wirkt sich auf beide der unter Kapitel 5 beschriebenen Simulationsdurchläufe gleichermaßen aus, so dass es zu keiner Verfälschung der Ergebnisse kommt. Die Verschlechterung der Verständlichkeit und Wartbarkeit des Simulationsmodells wird für diesen Anwendungsfall hingenommen.

Vereinfacht wurde auch die Ermittlung der Pfadwahrscheinlichkeiten. Um die Ergebnisse vergleichbar zu halten wurden diese im Simulationsmodell an die für den Vergleich verwendeten Lasttestfälle angepasst. Hierdurch durchlaufen die Simulationsdurchläufe statistisch gleich oft die einzelnen Pfade wie die Lasttestfälle. Die Vergleichbarkeit der Ergebnisse ist damit gewährleistet.

Das Verhalten der ROBASO-Anwendung wurde im Anwendungsfall durch das gemessene Antwortzeitverhalten der einzelnen Komponenten in Form eines Delays abgebildet. Die Messung von Ressourcennutzungsinformationen wurde dadurch komplett vermieden. Da die ROBASO-Anwendung für beide Experimentdurchläufe identisch bleibt wird das Ergebnis hierdurch nicht verfälscht.

5.4.2.5.2 Lösungsansätze für den Praxiseinsatz

Diese Lösungswege eignen sich für den Anwendungsfall, nicht jedoch für den Einsatz in der Fläche. Da dies den Rahmen dieser Arbeit überschreitet und eine Validierung nicht durchgeführt werden kann werden mögliche Lösungsansätze im Folgenden nur skizziert.

Die Erstellung des Simulationsmodells komplexer Anwendungen in der Praxis erfordert eine automatische Generierung auf Basis vorhandener Informationen und gemessener Daten. Eine manuelle Erzeugung der Modelle komplexer Anwendungen ist, zumindest auf Basis des existierenden Editors, aufgrund des hohen manuellen Aufwands kaum zu bewerkstelligen. Weiterhin reduziert eine Generierung das Problem der Modell-Aktualität: Änderungen am simulierten System müssen im Simulationsmodell nicht mehr nachgezogen werden. Vielmehr wird vor jedem Simulationslauf das Simulationsmodell (oder Teile dessen) komplett neu generiert. Die Aktualität ist damit gewährleistet, das Modell kann nicht veralten.

Für die Abbildung der Orchestrierungsschicht bedarf es einer Erweiterung des PCM-Metamodells um eine Workflow-Komponente oder der Erweiterung des Usage Models um die Fähigkeit der Verschachtelung mehrerer Ebenen. Auf diese Weise wird eine Darstellung von Prozessen verschiedener Ebenen, wie im Anwendungsfall vorhanden, ermöglicht. Eine solche Erweiterung unterstützt zudem die Wiederverwendbarkeit einzelner Teilprozesse. Aktuell ist die Mehrfachverwendung von Teilen des Usage Models in PCM nicht vorgesehen. Werden im Anwendungsfall Teilprozesse mehrfach verwendet, so müssen diese aktuell mehrfach abgebildet werden. Eine Wiederverwendung auf Prozessebene macht damit durchaus Sinn und vereinfacht die Modellerstellung und –pflege.

Die Ermittlung der Pfadwahrscheinlichkeiten ist eine primär organisatorische Aufgabe. Die Möglichkeit der Ermittlung der Wahrscheinlichkeiten aus Logfiles der einzelnen Verfahren auf der Produktivumgebung ist gering, da die Logfiles Verfahrens-spezifisch aufgebaut sind und keine eindeutige Erkennung von Prozessdurchläufen ermöglichen. Der einzige Ansatz, um über den gesamten Nutzerkreis gültige Pfadwahrscheinlichkeiten zu erlangen ist eine großflächige Analyse. Diese kann entweder in Form von Workshops ausgewählter Mitarbeiter geschehen, oder über eine Umfrage. Für letztere bietet sich die Verwendung eines gemeinsam genutzten Wikis an. Dieses Vorgehen wird zum Zeitpunkt der Erstellung dieser Arbeit bereits in Kooperation mit dem Projekt WISMO geprüft (Universität Osnabrück 2013).

Die Ermittlung der Ressourcennutzungsinformationen kann über Schnittstellen des Application Servers erfolgen. Hierzu stellt der WebLogic Application Server eine Java Management Extension (JMX)-Schnittstelle (Oracle 2012b) bereit. Diese kann beispielsweise über Servlet Filter und EJB Interceptoren (Perry 2003) angesprochen werden.

5.4.3 Durchführung

Die Simulation der PCM-Modelle erfolgt durch die Simulations-Engine *SimuCom* (Becker et al. 2009), welche in das PCM-Framework integriert ist. Die Ausführung der Simulation erfolgt durch die PCM-eigenen Startroutinen.

Zur Durchführung der Simulationsdurchläufe ohne und mit den Komponentenmodellen werden die Simulationsmodelle erstellt, anschließend wird das Projekt kopiert. Die Anpassungen für die vergleichenden Simulationsdurchläufe geschehen ausschließlich an den SEFFs der Komponenten, die Haupt-Simulationsmodelle werden nach dem Kopieren nicht mehr verändert.

In der Variante ohne Komponentenmodelle werden die Laufzeiten der Services als Normalverteilung (Georgii 2009) um den durchschnittlichen Antwortzeitwert abgebildet. In PCM geschieht dies durch die Darstellung als Delay, dessen Wartezeit in Form einer *Norm*-Funktion definiert ist (vgl. (Reussner et al. 2007)). In der Variante mit Komponentenmodellen hingegen erfolgt die Darstellung der SEFFs wie in Teilkapitel 4.3.2 beschrieben als *Formula*-PCI.

Die Simulationsdurchführung ohne und mit Komponentenmodellen erfolgt auf derselben Maschine, um Einflüsse der Hardware auszuschließen. Zwischen den beiden Simulationsdurchläufen erfolgt eine Pause von über einer Stunde, in der das Betriebssystem der verwendeten Hardware neu gestartet wird, um eine gegenseitige Beeinflussung der Simulationsläufe auszuschließen. Für die Simulation wird dieselbe PCM-Instanz verwendet. Beide Varianten werden mit 150 parallelen Anwendern simuliert. Hierdurch wird das in Abschnitt 5.1.4 beschriebene erwartete Zugriffsverhalten abgebildet. Je Simulationslauf wurden 30.000 Zugriffe simuliert. Die folgenden Abschnitte präsentieren die durch die Simulation gewonnenen Ergebnisse.

5.4.4 Ergebnisse ohne Komponentenmodelle

Ohne Komponentenmodelle erfolgt die Simulation des Anwendungsfalls auf konventionelle Weise. Da für die Service-Operationen weiterhin nur Antwortzeitinformationen aus dem Lasttest vorliegen, erfolgt die Abbildung des Komponentenverhaltens durch eine Normalverteilung um den durchschnittlichen Antwortzeitwert. Die Lasttest-Ergebnisse liefern zu den einzelnen Service-Operationen sowohl diesen durchschnittlichen Antwortzeitwert als auch die Standardabweichung. Mittels der beiden Werte und der *Norm*-Funktion von PCM wird das Antwortzeitverhalten der Service-Operationen als *Delay* abgebildet.

Die Simulation des Anwendungsfalls *Terminierung vornehmen* ergibt die in Abbildung 5.9 dargestellte Antwortzeitverteilung.

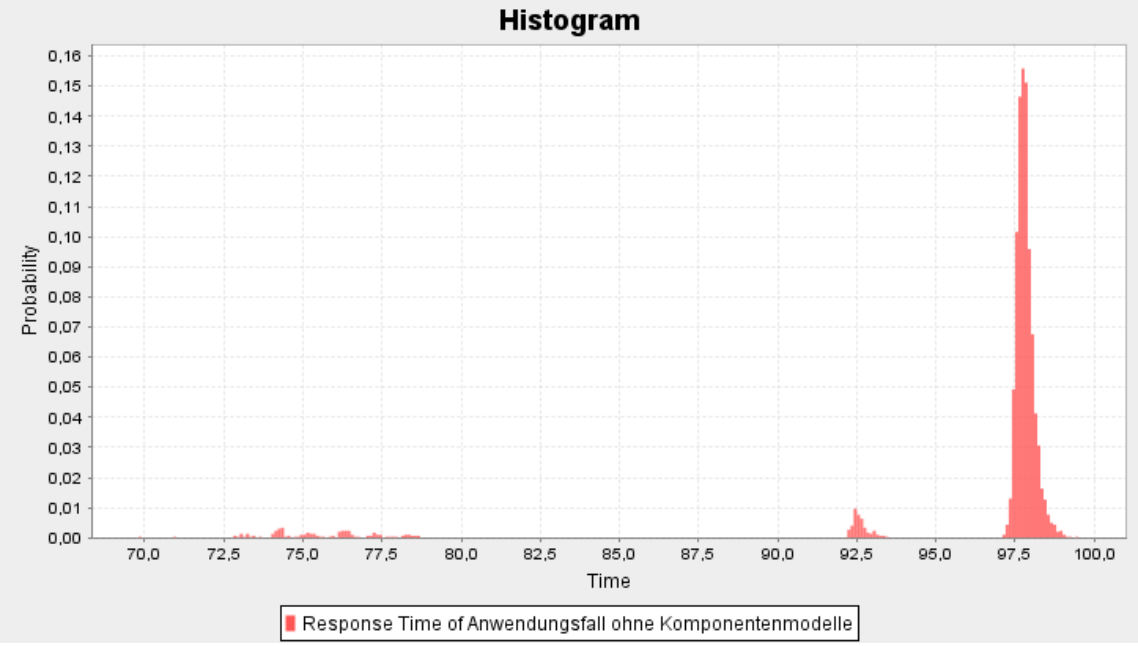


Abbildung 5.9. Simulierte Antwortzeitverteilung des Anwendungsfalls *Terminierung vornehmen* ohne Komponentenmodelle. Quelle: Eigene Darstellung.

Aus der Abbildung wird ersichtlich, dass die Antwortzeit des Anwendungsfalls von 72 Sekunden bis etwa 100 Sekunden erstreckt. Ein Großteil der Anfragen wird in dem Zeitrahmen zwischen 97 und 99 Sekunden bearbeitet. In den Bereichen 72 bis 79 Sekunden und 92 bis 93,5 Sekunden gibt es kleinere Häufungen.

5.4.5 Ergebnisse mit Komponentenmodellen

Ersetzt man die Normalverteilungen zur Beschreibung des Antwortzeitverhaltens der in Tabelle 8 aufgeführten Service-Operationen durch Komponentenmodelle, so liefert die Simulation eine Antwortzeitverteilung wie in Abbildung 5.10 dargestellt.

Auf den ersten Blick wirkt die Antwortzeitverteilung ähnlich der Simulation ohne Komponentenmodelle. Klar erkennbar ist jedoch die deutlich breitere Streuung der simulierten Antwortzeiten im Bereich von 90 bis 101 Sekunden. Auch bei der Simulation mit der Verwendung der Komponentenmodelle entsteht eine geringe Häufung der Antwortzeiten im Bereich 72 bis 79 Sekunden.

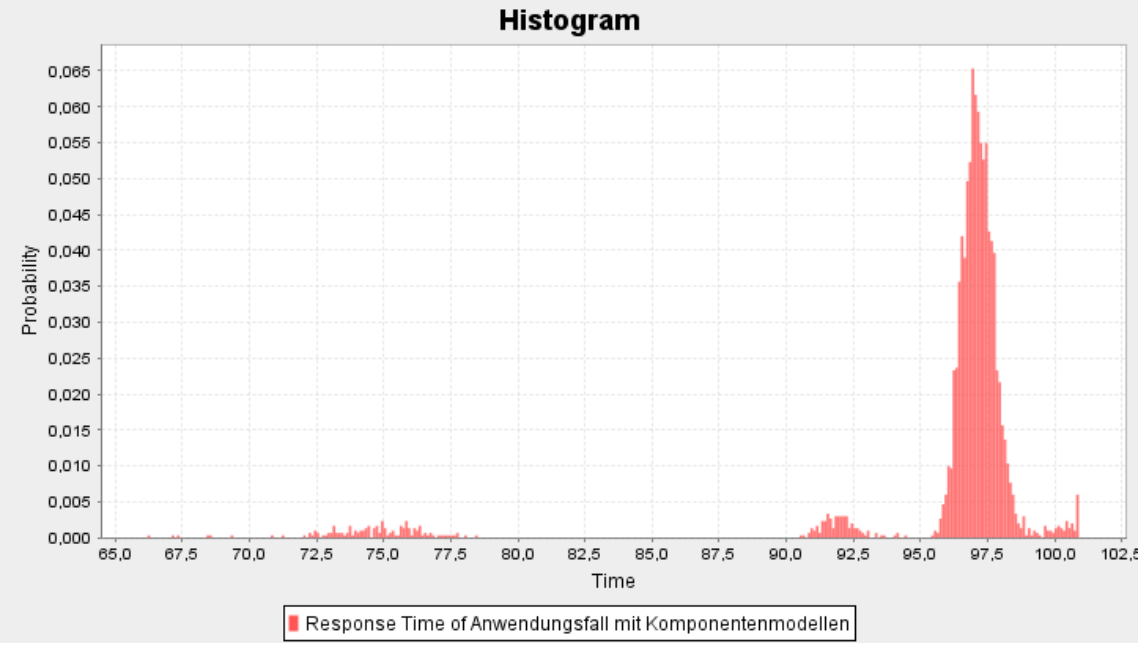


Abbildung 5.10. Simulierte Antwortzeitverteilung des Anwendungsfalls mit Komponentenmodellen. Quelle: Eigene Darstellung.

5.4.6 Zusammenfassung

Der Aufbau des Simulationsmodells bildet das Rahmenwerk des Experiments. Nur wenn dieses Rahmenwerk stabil bleibt und sich ausschließlich die bewusst angepassten Variablen verändern lässt sich eine Änderung im Ergebnis auf diese Variablenanpassung zurückführen.

In den vorhergehenden Abschnitten ist beschrieben, wie das PCM-Simulationsmodell des Experimentaufbaus strukturiert ist, welche Hauptmodelle es enthält, und wie die Variablenanpassung, das Hinzufügen der Komponentenmodelle, vonstattengeht, ohne den restlichen Experimentaufbau zu verändern. Hierzu wird das Simulationsmodell erstellt und anschließend kopiert. Die beiden Kopien des Simulationsmodells werden vollständig konsistent gehalten, einzig die SEFFs der Service-Operationen werden durch die Einbindung der Komponentenmodelle angepasst. Die Stabilität des Experimentaufbaus ist damit gewährleistet.

Weiterhin ist beschrieben, wie die Experimentdurchführung, also das Ausführen der beiden Simulationsdurchläufe, durchgeführt wird. Hierzu werden die beiden Simulationsmodelle (mit und ohne die Einbindung der Komponentenmodelle) mit ausreichend zeitlicher Distanz und einem zwischenzeitlichen Neustart der Hardware (um eine Beeinflussung auszuschließen) auf derselben Hardware und mit derselben PCM-Instanz simuliert. Eine Verfälschung der Ergebnisse durch Hardware- und Softwareunterschiede ist damit ebenfalls ausgeschlossen.

Letztendlich erfolgt die Präsentation der Simulationsergebnisse, noch ohne Bewertung. Diese folgt im nächsten Kapitel. Es ist jedoch bereits bei der Betrachtung der simulierten Antwortzeitverteilungen deutlich, dass die Verwendung der Komponentenmodelle zu einem im Großen ähnlichen, im Kleinen jedoch unterschiedlichen Simulationsergebnis führt.

5.5 Bewertung der Ergebnisse

Ein Vergleich der vorab vorgestellten Simulationsergebnisse lässt auf den ersten Blick nur geringe Unterschiede erkennen. Vertieft man die Betrachtung, wie im Folgenden beschrieben, so werden doch einige Veränderungen deutlich. Dies ist insbesondere vor dem Hintergrund relevant, dass für die vorliegende Betrachtung nur vier der insgesamt 47 Service-Operationen durch Antwortzeit-Komponentenmodelle abgebildet wurden (vgl. Abschnitt 5.3.3).

Neben dem Vergleich der Simulationsergebnisse erfolgt in diesem Teilkapitel auch ein Vergleich der Simulationsergebnisse mit dem gemessenen Antwortzeitverhalten. Erst durch diese Vergleiche sind Schlüsse hinsichtlich der Verbesserung des Simulationsergebnisses zu ziehen.

5.5.1 Vergleich der Simulationsergebnisse

Die im vorherigen Teilkapitel präsentierten simulierten Antwortzeitverteilungen ohne und mit Komponentenmodellen wirken auf den ersten Blick ähnlich. Geht man ins Detail, und berücksichtigt man, dass nur vier der insgesamt 47 verwendeten Service-Operationen durch Komponentenmodelle abgebildet wurden, so ergibt sich doch ein deutlicher Unterscheid.

Bei der Betrachtung der beiden Simulationsergebnisse wird sofort sichtbar, dass die Verwendung der Komponentenmodelle zu einer deutlich stärkeren Verteilung der Antwortzeiten um den Mittelwert der Hauptspitze bei 97 Sekunden führt. Die kleineren Häufungspunkte bei 72 bis 77 Sekunden und bei 91 bis 94 Sekunden sind in beiden Simulationsergebnissen ausgeprägt, bei den Ergebnissen mit Komponentenmodellen jedoch deutlicher und verteilter.

Eine Betrachtung der simulierten kumulierten Antwortzeitverteilung des Anwendungsfalls mit und ohne die Komponentenmodelle (dargestellt in Abbildung 5.11) zeigt, dass die Verwendung der Komponentenmodelle zu einer leichten aber dennoch sichtbaren Reduktion der durchschnittlichen Antwortzeit führt.

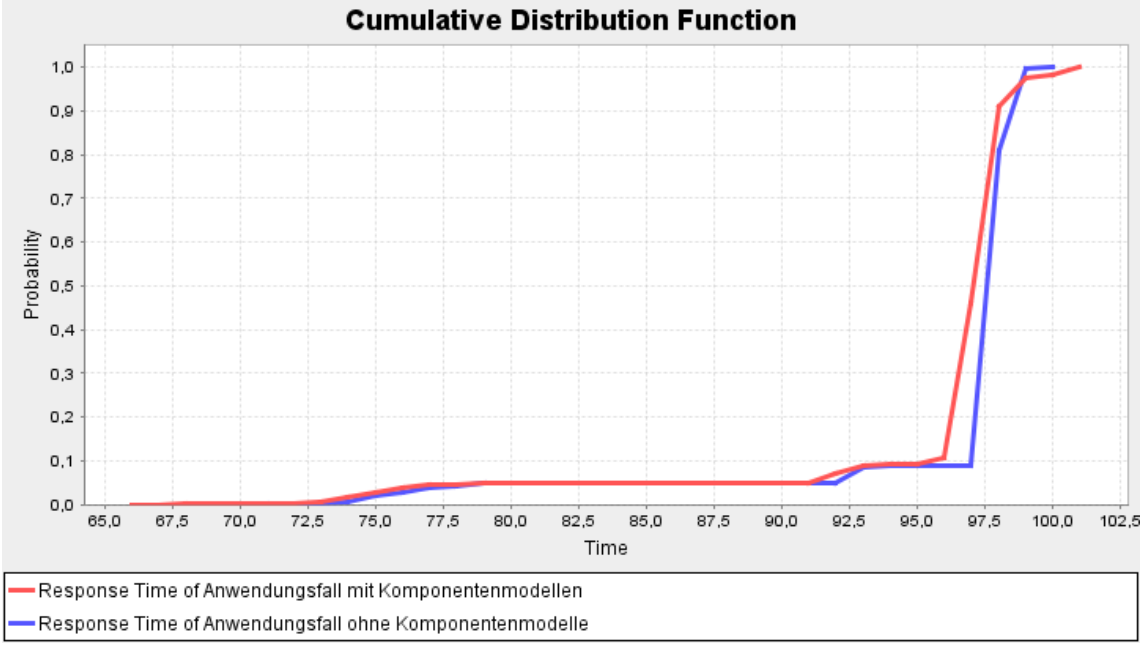


Abbildung 5.11. Simulierte kumulative Verteilung der Antwortzeiten des Anwendungsfalls ohne und mit Komponentenmodellen. Quelle: Eigene Darstellung.

Ebenfalls ergibt sich aus der kumulierten Antwortzeitverteilung, wie auch bereits in Abbildung 5.10 erkennbar, eine Verschiebung des Maximalwerts über die Grenze von 100 Sekunden hinaus.

Rein durch den Vergleich der simulierten Antwortzeitverteilungen lässt sich jedoch nicht sagen, ob die Verwendung der durch den evolutionären Algorithmus generierten Komponentenmodelle tatsächlich eine Verbesserung des Simulationsergebnisses gebracht hat. Hierzu erfolgt in den folgenden Abschnitten der Vergleich der beiden Simulationsergebnisse mit dem gemessenen Antwortzeitverhalten des betrachteten Systems. Hierdurch ergibt sich, ob die Verwendung der Komponentenmodelle eine Annäherung des Simulationsergebnisses an die gemessene Antwortzeitverteilung geliefert hat.

5.5.2 Gemessene Prozessdurchlaufzeiten

Zur Bewertung der Veränderung der simulierten Antwortzeitverteilung ist ein Vergleich mit Messwerten des betrachteten Anwendungsfalls notwendig. Hierzu wird auf die im regulären Lasttest erhobenen Prozessdurchlaufzeiten zurückgegriffen. Der Lasttest wurde mit 150 parallelen Nutzern durchgeführt, jeder Nutzer folgte dem vorab beschriebenen Anwendungsfall. Abbildung 5.12 stellt die Häufigkeitsverteilung der gemessenen Prozessdurchlaufzeiten grafisch dar.

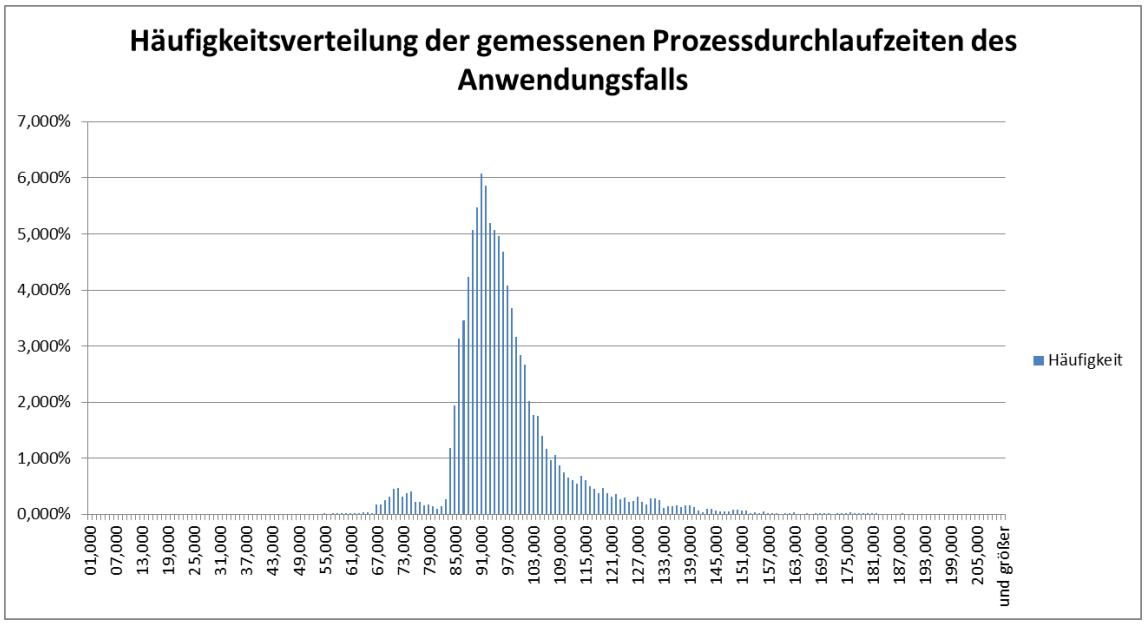


Abbildung 5.12. Häufigkeitsverteilung der gemessenen Prozessdurchlaufzeiten des Anwendungsfalls aus dem Lasttest mit 150 Anwendern. Quelle: Eigene Darstellung.

Auffällig ist die breite Streuung der Antwortzeiten von 67 bis zu etwa 150 Sekunden. Auch die erste Häufung um etwa 72 Sekunden fällt ins Auge. Die Hardware-Ressourcen waren bei diesem Lasttest unter 50 Prozent ausgelastet, so dass keine Beeinflussung durch Hardware-Engpässe zu vermuten ist.

5.5.3 Bewertung der Simulationsergebnisse ohne Komponentenmodelle

Vergleicht man die Simulationsergebnisse ohne Komponentenmodelle mit den gemessenen Prozessdurchlaufzeiten, so fällt die deutlich schmalere Verteilung der simulierten Ergebnisse um die 97,5 Sekunden auf. In der Spitze der Verteilung stimmen die gemessenen Werte und die Simulationsergebnisse in etwa überein, die simulierten Antwortzeiten liegen leicht über den gemessenen. Auch die vorgelagerten Prozessdurchlaufzeiten um die 72 Sekunden bildet das Simulationsergebnis ab.

Eine deutliche Abweichung ergibt sich hinsichtlich der Streuung der Antwortzeiten um die Häufigkeitsspitze. Während das Simulationsergebnis neben dem breiten Häufigkeitsfeld zwischen 72 und 78 Sekunden zwei weitere Spitzen in den Bereichen 92 bis 93 Sekunden und 97 bis 99 Sekunden abbildet zeigt sich in den Messwerten eine breite Streuung von 82 Sekunden bis etwa 120 Sekunden, mit einem asymptotisch auslaufenden Langläufer-Bereich bis zu 151 Sekunden. Diesen Bereich der langen Antwortzeiten jenseits der 100 Sekunden bildet das Simulationsergebnis ohne Komponentenmodelle nicht ab.

5.5.4 Bewertung der Simulationsergebnisse mit Komponentenmodellen

In der grundlegenden Struktur ergeben sich für die Simulationsergebnisse mit Komponentenmodellen dieselben Ergebnisse wie für die Simulationsergebnisse ohne Komponentenmodelle. Die Spitze der höchsten Häufigkeitsverteilung trifft mit 97 Sekunden noch etwas präziser die gemessenen Werte. Auch die Häufung der Antwortzeiten zwischen 72 und 77 Sekunden stellt die gemessenen Werte etwas exakter dar. Im Großen und Ganzen unterscheiden sich die Simulationsergebnisse in dieser Hinsicht jedoch kaum.

Einen deutlichen Unterschied verursacht die Verwendung der Komponentenmodelle hingegen bei der Streuung der simulierten Antwortzeiten. Verglichen mit den gemessenen Werten ist die simulierte Streuung immer noch zu eng. Dennoch führt die Verwendung der Komponentenmodelle zur einer Kurvenform, die der gemessenen deutlich genauer entspricht. Auch die Streuung über die 100 Sekunden hinaus nähert, wenn auch noch deutlich zu gering, die simulierte Antwortzeitverteilung der gemessenen an.

5.5.5 Auswirkung auf die Praxis

Die Verwendung der mittels des evolutionären Algorithmus erzeugten Komponentenmodelle führen zu einer Annäherung des Simulationsergebnisses an das gemessene Antwortzeitverhalten. Der Vorteil der Verwendung des evolutionären Algorithmus liegt dabei in der Praxis insbesondere in der Tatsache, dass keine Vorgaben (wie beispielsweise bei der mathematischen Regression) und kein manueller Aufwand für den Einsatz notwendig sind. Performance-Verantwortlichen kann der evolutionäre Algorithmus in Form beispielsweise eines Services angeboten werden, dem ausschließlich gemessene Antwortzeitdaten bereitgestellt werden müssen. Die erzeugten Modelle können versioniert und zentral abgelegt werden und auf diese Weise von den Performance-Verantwortlichen anderer Nutzer des modellierten Services verwendet werden. Auch eine vergleichende Betrachtung des Performance-Verhaltens einer Service-Operation über die Zeit und über mehrere Versionsstände ist hierdurch möglich. Eine solche Betrachtung erlaubt beispielsweise eine Bewertung des Effekts von veränderten Datenbank-Füllständen oder von Refactoring-Maßnahmen auf das Antwortzeitverhalten des modellierten Services.

Dem „Performance Worker“ der Praxis bietet der in dieser Arbeit vorgestellte Ansatz damit die Möglichkeit, ohne manuellen Aufwand und ohne Wissen über die Funktionalität und den inneren Aufbau eines Services das Antwortzeitverhalten der Operationen dieses Services in der Simulation exakt abzubilden. Die gewählte Form der Ein- und Ausgabedaten des evolutionären Algorithmus erlauben ihm dabei den Einsatz etablierter Techniken des Software Engineerings, wie beispielsweise ein Repository, um die Modelle zentral zu

verwalten und zu versionieren. Der für die Modellierung notwendige Ressourceneinsatz kann auf diese Weise optimiert werden.

5.5.6 Zusammenfassung

Die Einbindung von Komponentenmodellen in das Simulationsmodell führt zu einer Annäherung des Simulationsergebnisses an das messbare Antwortzeitverhalten des Anwendungsfalls. Im Vergleich mit dem gemessenen Antwortzeitverhalten bilden die Simulationsergebnisse mit Komponentenmodellen dieses exakter dar als die Simulationsergebnisse ohne Komponentenmodelle.

Werden im betrachteten System existierende Zusammenhänge in der Simulation nicht abgebildet, so wird dies jedoch durch die Einbindung der Komponentenmodelle auch nicht behoben. Im vorgestellten Anwendungsfall tritt in den Messwerten eine unerwartet hohe Streuung der Antwortzeiten auf. Da die Auslastung der Hardware-Ressourcen unter 50 Prozent liegt ist diese Streuung auf eine oder mehrere im Simulationsmodell nicht abgebildete Korrelationen zurückzuführen. Diese können in der ROBASO-Anwendung nicht verortet werden, so dass sie zwischen den Services bestehen müssen.

Existieren solche unbekannten Zusammenhänge zwischen den als Black Box betrachteten Komponenten, so stößt die Simulation an ihre Grenzen. Die aus Lasttestdaten gewonnenen Komponentenmodelle nähern die simulierten Antwortzeiten insoweit dem tatsächlichen Verhalten an, als dieses in dem Lasttest, aus dem die für die Generierung genutzten Daten stammen, aufgetreten ist. Mit den Komponentenmodellen wird implizites Wissen in die Simulation eingebracht, das durch eine Abbildung mittels Normalverteilung nicht gegeben ist.

5.6 Zusammenfassung

Nachdem in den vorherigen Kapiteln das kontrollierte Softwareexperiment Schritt für Schritt vorbereitet wurde, erfolgt in diesem Kapitel nun die Beschreibung der Experimentdurchführung.

Im ersten Schritt wird der Anwendungsfall vorgestellt. Die ROBASO-Anwendung entstammt aus einem realen Anwendungskontext und eignet sich damit für die Überprüfung der in Kapitel 1.2 getroffenen Annahme. Die Auswahl des Hauptprozesses „Terminierung vornehmen“ des Terminierungsprozesses der ROBASO-Anwendung als Anwendungsfall reduziert den Anwendungsfall auf einen im Rahmen dieser Arbeit umsetzbaren Umfang. Der Terminierungsprozess dient auch im ROBASO-Projekt als *Proof of Concept*, von der Repräsentativität des Prozesses für die Anwendung kann damit ausgegangen werden.

Zudem befindet sich der Terminierungsprozess in einem Umsetzungsstadium, in welchem die benötigten Informationen für die in dieser Arbeit beschriebene Untersuchung erlangt werden können.

Auf den Anwendungsfall folgend wird beschrieben, wie die für die Modellierung und Simulation benötigten Performance-Messwerte des Anwendungsfalls erhoben werden. Das Teilkapitel umfasst alle drei Teile des Anwendungsfalls (ROBASO-Oberflächenanwendung, Services und Middleware), der Fokus liegt jedoch auf der Messung der Antwortzeiten der verwendeten Services. Diese bilden die Grundlage für die Modellierung der in dieser Arbeit betrachteten Komponentenmodelle, so dass die Messung umfangreich beschrieben ist.

Die Anwendung des evolutionären Algorithmus zur Generierung der Komponentenmodelle ist Inhalt des darauffolgenden Teilkapitels. Die in den vorangehenden Kapiteln detailliert beschriebenen theoretischen Grundlagen und die prototypische Implementierung wird hier um die konkrete Konfiguration des Prototyps *Mendel* und die Durchführung der Modellierung auf den Messdaten des Anwendungsfalls erweitert. Zudem erfolgt eine Erörterung der aus den beschriebenen Messdaten gewonnenen Komponentenmodelle. In diesem Teilkapitel wird gezeigt, dass sich der Ansatz der evolutionären Algorithmen auch in der Praxis bewährt. Auf den aus einem realen Lasttest stammenden Messdaten generiert der evolutionäre Algorithmus innerhalb einer halben Stunde Komponentenmodelle mit einem Fehlerwert von fünf Prozent. Als Hindernis für die Modellierung stellt sich die Aggregation der Lasttestergebnisse durch die konventionellen Lasttest-Werkzeuge dar. Diese komprimieren die gemessenen Ergebnisse oft zu Kennzahlen wie der durchschnittlichen Antwortzeit über einen definierten Zeitraum hinweg, wodurch eine Gewinnung der benötigten Einzelmesswerte erschwert oder vollständig unmöglich wird. Aus diesem Grund konnten nur für vier der insgesamt 47 im Anwendungsfall aufgerufenen Service-Operationen Antwortzeit-Komponentenmodelle generiert werden.

Die Experimentdurchführung in Form der Simulationsläufe liefert die simulierten Antwortzeitverteilungen des Anwendungsfalls ohne und mit dem Einsatz der Komponentenmodelle. Ein Vergleich der Simulationsergebnisse miteinander zeigt, dass die Ergebnisse im Grundlegenden zwar ähnlich, in der inneren Verteilung jedoch deutlich unterschiedlich sind. Während die Mittelwerte der Häufungen in ähnlichen Bereichen liegen unterscheidet sich die Streuung der Antwortzeiten stark. Die Simulationsergebnisse ohne Komponentenmodelle liefert sehr diskrete Werte, während die Simulationsergebnisse mit Komponentenmodellen eine deutlich breitere Streuung der Antwortzeiten enthält.

Der Vergleich der Simulationsergebnisse mit den gemessenen Antwortzeiten des Anwendungsfalls ergibt eine Verbesserung der simulierten Ergebnisse durch den Einsatz der Komponentenmodelle. Die breite Streuung der gemessenen Antwortzeiten wird durch die Simulation mit Komponentenmodellen deutlich besser abgebildet. Auch die gemessenen längeren Antwortzeiten jenseits der 100 Sekunden bildet die Simulation mit Komponentenmodellen ab.

Allerdings weicht auch die Antwortzeitverteilung der Simulation mit Komponentenmodellen von den gemessenen Werten ab. Die Streuung auch dieser Simulationsergebnisse ist deutlich enger als die gemessene. Zudem treten in den Messwerten am oberen Ende deutlich längere Antwortzeiten auf als in den Simulationsergebnissen. Dies ist ersten Analysen zufolge auf Korrelationen zwischen den aufgerufenen Services zurückzuführen, die in der Simulation nicht abgebildet wurden. Betrachtet man den geringen Aufwand, der für die Erstellung und Einbindung der Antwortzeit-Komponentenmodelle betrieben werden muss, so ist die gewonnene Verbesserung des Simulationsergebnisses bedeutend.

Mit Blick auf die vorherigen Ergebnisse lässt sich nun die dritte Forschungsfrage nach dem Einfluss der Komponentenmodelle im Anwendungsfall abschließend beantworten.

Forschungsfrage 3: Wie wirkt sich der Einsatz von generierten Komponentenmodellen auf die Performancesimulation von Unternehmensanwendungen im Fallbeispiel aus?

Antwort 3: Der Einsatz von generierten Komponentenmodellen führt im Fallbeispiel zu einer Verbesserung der Simulationsergebnisse. Sowohl die simulierten Einzelwerte als auch die Verteilung der Antwortzeiten liegen bei der Verwendung der Komponentenmodelle näher an den gemessen Werten.

Durch die Verwendung der Komponentenmodelle ist eine Annäherung der simulierten Antwortzeitverteilung an die gemessenen Antwortzeiten zu beobachten. Dies betrifft vor allem die Struktur der Antwortzeitverteilung: die Simulationsergebnisse mit Komponentenmodellen bilden die gemessene Streuung deutlich besser ab als die Verwendung von Normalverteilungen zur Abbildung des Antwortzeitverhaltens einer Service-Operation. Dennoch ist im Anwendungsfall eine Diskrepanz zwischen den Simulationsergebnissen und den gemessenen Antwortzeiten gegeben. Diese ist ersten Analysen zufolge auf nicht in der Simulation abgebildete Korrelationen zwischen den verwendeten Services zurückzuführen. Da diese Korrelationen auch in den Lasttests wirkten, aus denen die Daten stammen, die zur Generierung der Komponentenmodelle verwendet

werden gleichen die Komponentenmodelle dieses Fehlen zum Teil aus. Komplett ist dies jedoch nicht der Fall. Dennoch ist die Verbesserung deutlich, insbesondere unter der Berücksichtigung, dass im Anwendungsfall nur vier von insgesamt 47 Service-Operationen durch Komponentenmodelle abgebildet werden konnten.

Die Simulation des Anwendungsfalls zeigt zudem, dass bei einem einheitlichen Lasttestergebnis-Standard kaum manueller Aufwand für die Generierung der Komponentenmodelle durch den evolutionären Algorithmus und die Einbindung der Modelle mittels der *Formula*-PCI-Implementierung in die Simulation erforderlich ist. Der gewonnenen Verbesserung des Simulationsergebnisses steht damit kaum zusätzlicher manueller Aufwand gegenüber.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Ziel dieser Arbeit ist es, eine Brücke aufzuzeigen zwischen der Messung des Antwortzeitverhaltens von Komponenten einer Unternehmensanwendung und der Simulation des Performance-Verhaltens der Anwendung. Um diese Brücke in der Praxis einsetzen zu können ist ein minimaler manueller Aufwand zwingend notwendig. Evolutionäre Algorithmen als Instanz der Verfahren maschinellen Lernens bieten sich hierfür an. Die vorangehenden fünf Kapitel dieser Arbeit befassen sich mit der Überprüfung der Annahme, dass ein existierender Simulationsansatz für die Performance von Unternehmensanwendungen durch die Einbindung von durch einen evolutionären Algorithmus generierten Komponentenmodellen verbessert werden kann. In zahlreichen Einzelschritten wurde gezeigt, wie ein evolutionärer Algorithmus aufgebaut ist, der die Modellierung der Komponentenmodelle durchführen kann, welche Herausforderungen seine Konfiguration und die Einbindung der Ergebnisse in die Simulation mit sich bringen, und wie diese Herausforderungen bewältigt werden können. Letztendlich wurde anhand einer Fallstudie gezeigt, wie sich der Einsatz der Komponentenmodelle auf die Performance-Simulation der betrachteten Unternehmensanwendung auswirkt.

Zur Strukturierung der Arbeit wurde die oben genannte Annahme in zwei Annahmen aufgeteilt. Die Prüfung dieser beiden Annahmen erfolgt anhand der Beantwortung von drei Forschungsfragen. Die Ergebnisse der Arbeit sind wie folgt:

Annahme 1. Antwortzeitmodelle für Softwarekomponenten ermöglichen eine Performance-Analyse von Unternehmensanwendungen mittels eines existierenden Performance-Simulationsansatzes, die Skalierungsbetrachtungen beinhaltet.

Die Einbindung von Antwortzeit-Komponentenmodellen in den PCM-Simulationsansatz führt bei dem in Kapitel 5 betrachteten Anwendungsfall zu Simulationsergebnissen, die näher an dem gemessenen Performance-Verhalten des Anwendungsfalls liegen, als die Simulationsergebnisse unter Verwendung von Verteilungsfunktionen für das Komponentenverhalten, dem konventionellen Vorgehen. Dabei ist zu beobachten, dass sich die Antwortzeiten breiter über den Ergebnisbereich verteilen, ohne jedoch die grundlegende Struktur zu verlassen. Dies ist auf das Skalierungsverhalten zurückzuführen, welches die Antwortzeit-Komponentenmodelle abbilden. Nimmt die Anzahl der auf das betrachtete System parallel zugreifenden Anwender zu, so erhöht sich die Last auf allen Komponenten.

Da die Komponentenmodelle die Antwortzeit in Abhängigkeit der Anzahl paralleler Zugriffe berechnet wird diese Eigenschaft berücksichtigt – die simulierte Streuung der Antwortzeiten der Gesamtanwendung nimmt zu. Bei der Verwendung von Verteilungsfunktionen für die Beschreibung des Komponentenverhaltens hingegen erfolgt keine Berücksichtigung der aktuellen Anzahl paralleler Anfragen. Annahme 1 ist damit bestätigt: die Verwendung von Antwortzeit-Komponentenmodellen ermöglicht eine Performance-Analyse von Unternehmensanwendungen mittels des existierenden Performance-Simulationsansatz PCM, die eine Skalierungsbetrachtung beinhaltet.

Die zweite Annahme zielt auf die Erstellung der Antwortzeit-Komponentenmodelle ab. Um den vorgestellten Ansatz auch für große Unternehmensanwendungen mit zahlreichen Komponenten nutzbar zu machen ist es notwendig, dass die Erstellung der Komponentenmodelle vollautomatisch erfolgt. Hierfür wurde der Ansatz der evolutionären Algorithmen gewählt.

Annahme 2: Durch evolutionäre Algorithmen können Antwortzeitmodelle für Softwarekomponenten automatisch aus Messdaten generiert werden.

Zur Überprüfung dieser Annahme wurde ein evolutionärer Algorithmus prototypisch implementiert und zur Modellierung des Antwortzeit-Komponentenverhaltens des Anwendungsfalls eingesetzt. Voruntersuchungen in Form der Modellierung des Komponentenverhaltens einer Unternehmensanwendung ergaben zum einen, dass sich der evolutionäre Algorithmus für die Modellierung des Komponentenverhaltens auf Messdaten eignet, zum anderen, dass die Eignung des evolutionären Algorithmus stark von seiner Konfiguration abhängt. Anhand einer statistischen Untersuchung von 2.904 Modellierungsläufen des entwickelten Prototyps *Mendel,* sowie mittels der Implementierung eines evolutionären Algorithmus zur Ermittlung der optimalen Konfiguration für *Mendel* wurde eine optimale Konfiguration identifiziert. Bei der Performance-Simulation des Anwendungsfalls lieferte *Mendel* mit der identifizierten Konfiguration Komponentenmodelle mit einem Fehlerwert unter fünf Prozent. Bezüglich der zweiten Annahme lässt sich somit bestätigen, dass ein evolutionärer Algorithmus Antwortzeitmodelle für Softwarekomponenten automatisch aus Messdaten generieren kann.

Die drei Forschungsfragen dienten zur Strukturierung der Arbeit. Zum Abschluss lassen sich die Forschungsfragen wie folgt beantworten:

Forschungsfrage 1: Welcher in der Literatur beschriebene Simulationsansatz für das Performance-Verhalten von Unternehmensanwendungen eignet sich für die Performance-Analyse auf Basis des Antwortzeitverhaltens der Anwendungskomponenten?

Mittels einer strukturierten Literaturanalyse nach (Webster/Watson 2002) wurde die Literatur analysiert und die existierenden Simulationsansätze zur Simulation des Performance-Verhaltens von Unternehmensanwendungen identifiziert. Bei der Literaturanalyse stellte sich heraus, dass die existierenden Performance-Modellierungs- und -Simulationsansätze primär auf HPC konzentriert sind. Von den wenigen Performance-Simulationsansätzen für Unternehmensanwendungen wird rund die Hälfte nicht weiter entwickelt, so dass im Endeffekt nur zwei Ansätze, LQSIM von (Woodside 2002) und PCM von (Reussner et al. 2007) und (Becker et al. 2007), zur Auswahl standen.

Als Ergebnis der Beantwortung der Forschungsfrage 1 ergab sich die Auswahl eines Simulationsansatzes als Basis für das kontrollierte Softwareexperiment, welches zur Überprüfung der Annahme durchgeführt wurde. Die Wahl fiel auf das von (Reussner et al. 2007) vorgestellte Palladio Component Model (PCM). PCM eignet sich für die Durchführung der in dieser Arbeit beschriebenen Untersuchungen aus mehreren Gründen. PCM ist für die Performance-Analyse komponentenbasierter Softwaresysteme entworfen und eignet sich damit optimal für den in dieser Arbeit betrachteten Anwendungsfall (siehe Teilkapitel 5.1). Weiterhin befindet sich PCM in einem kontinuierlichen Weiterentwicklungsprozess durch eine etablierte Community, so dass die Nachhaltigkeit der vorgestellten Lösung gegeben ist. Zudem liegt PCM im Quellcode vor, so dass die in dieser Arbeit durchgeführten Adaptionen ohne Reengineering-Maßnahmen möglich waren.

Antwort 1: Der Performance-Simulationsansatz *Palladio Component Model* (PCM) von (Reussner et al. 2007) und (Becker et al., 2009) eignet sich für die Performance-Analyse von Unternehmensanwendungen auf Basis des Antwortzeitverhaltens der Anwendungskomponenten. Der Ansatz ist umfangreich in der Literatur beschrieben und wird kontinuierlich durch eine aktive und etablierte Community weiterentwickelt. PCM besitzt bereits eine erste Schnittstelle zum Einbinden von Antwortzeitkurven und liegt in Form des Quellcodes vor, wodurch die Entwicklung einer Schnittstelle zum Einbinden von Antwortzeit-Komponentenmodellen ermöglicht wird.

Als Ergebnis der Beantwortung der ersten Forschungsfrage wurde das Simulations-Framework PCM ausgewählt. Darauf aufbauend kann die zweite Forschungsfrage beantwortet werden.

Forschungsfrage 2: Wie kann ein evolutionärer Algorithmus in die Simulation der Performance einer Unternehmensanwendung eingebunden werden und wie muss der evolutionäre Algorithmus konfiguriert sein?

Die Beantwortung der zweiten Forschungsfrage bildet die Experimentvorbereitung des kontrollierten Softwareexperiments. Durch die Einbindung der durch den evolutionären Algorithmus generierten Modelle des Antwortzeitverhaltens der Softwarekomponenten in den ausgewählten PCM-Simulationsansatz wird die veränderliche Variable des Experiments definiert. Die Integration der Komponentenmodelle geschieht durch die Implementierung des von (Wert 2011) bereitgestellten Performance Curve Integration (PCI) Interfaces. Die prototypische *DataTable*-Implementierung von Wert eignet sich nicht für die Anforderungen des Experiments, so dass eine *Formula*-Implementierung umgesetzt wurde, welche die Definition der Antwortzeit-Komponentenmodelle in Form einer mathematischen Formel zulässt. Mithilfe dieser *Formula*-Implementierung ist es möglich, die Simulation auf demselben Modell, mit demselben Simulationsframework durchzuführen, und nur die Definition des Komponentenverhaltens anzupassen. Die Experimentumgebung ist damit sicher stabil, einzig die veränderliche Variable – die Darstellung des Komponentenverhaltens – ändert sich kontrolliert zwischen den Experimentdurchführungen.

Die Effektivität eines evolutionären Algorithmus zur Modellierung des Komponentenverhaltens ist stark abhängig von seiner Konfiguration. Um die Ergebnisse des Experiments nicht durch fehlerhafte Konfigurationsparameter zu beeinträchtigen wurde mittels statistischer Analyse sowie eines evolutionären Algorithmus eine optimale Konfiguration für den evolutionären Modellierungsalgorithmus entwickelt. Als optimale Konfiguration ergaben sich die in Tabelle 9 aufgeführten Konfigurationswerte. Als Fitnessfunktion kam bei der Modellierung die quadratische Fehlerwertberechnung zum Einsatz.

Tabelle 9. Optimale Konfiguration des Modellierungsalgorithmus.

Konfigurationsparameter	**Wert**
Populationsgröße	5.000
Genome-Länge	41
Mutations-Wahrscheinlichkeit	90%
Crossover-Wahrscheinlichkeit	75%
Parental Population Quota	30%

Die zweite Forschungsfrage kann damit wie folgt beantwortet werden:

Antwort 2: Der evolutionäre Algorithmus generiert aus Performance-Messwerten der Komponenten der Unternehmensanwendung Antwortzeit-Komponentenmodelle. Diese Komponentenmodelle bilden das Antwortzeitverhalten der Komponenten ab und liegen in Form von mathematischen Formeln vor. Über die *Formula*-PCI-Implementierung können diese mathematischen Formeln in die Simulation als Beschreibung der Anwendungskomponenten eingebunden werden.

Eine optimale Konfiguration des evolutionären Algorithmus lässt sich auf zwei Arten analysieren. Ein Ansatz ist die statistische Analyse vieler Modellierungsläufe. Diese Analyse ergibt eindeutige Konfigurationsempfehlungen für die Populationsgröße und die Genome-Länge, nicht jedoch für die Mutations- und Crossover-Wahrscheinlichkeit. Eindeutigere Ergebnisse werden durch den Einsatz eines evolutionären Algorithmus zur Konfiguration des evolutionären Modellierungs-Algorithmus erzielt. Mittels dieses Meta-GA ist eine sehr genaue Konfiguration des Modellierungs-Algorithmus möglich. Als optimale Konfiguration des Modellierungs-Algorithmus für den in dieser Arbeit betrachteten Anwendungsfall wurden folgende Konfigurationswerte ermittelt:

- Populationsgröße: 5.000
- Genom-Länge: 41
- Mutations-Wahrscheinlichkeit: 90%
- Crossover-Wahrscheinlichkeit: 75%
- Parental Population Quota: 30%

Durch die Beantwortung der ersten und zweiten Forschungsfrage sind alle Grundlagen für die Durchführung des kontrollierten Softwareexperiments geschaffen. Ein Simulationsframework wurde ausgewählt. Der evolutionäre Algorithmus wurde implementiert, konfiguriert, und die durch den Algorithmus generierten Modelle können mittels der entwickelten *Formula*-PCI-Implementierung in die Simulation eingebunden werden. Im nächsten Schritt wurde nun ein Anwendungsfall ausgewählt und in PCM abgebildet. Das Simulationsmodell des Anwendungsfalls gehört ebenfalls zum Experimentaufbau und wurde so entworfen, dass sich die veränderliche Variable, die Einbindung der Komponentenmodelle, ohne große Änderung am Simulationsmodell durchführen lässt. Unerwünschte Nebeneffekte durch die Anpassung der veränderlichen Variablen lassen sich dadurch ausschließen. Mit der Fertigstellung des Simulationsmodells war die

Experimentvorbereitung abgeschlossen, so dass mit der Experimentdurchführung begonnen werden konnte.

Die Experimentdurchführung des kontrollierten Softwareexperiments führte zur Beantwortung der dritten Forschungsfrage.

Forschungsfrage 3: Wie wirkt sich der Einsatz von generierten Komponentenmodellen auf die Performance-Simulation von Unternehmensanwendungen im Fallbeispiel aus?

Anhand eines Realwelt-Anwendungsfalls wurde der Effekt der Verwendung von automatisch generierten Komponentenmodellen auf die Performance-Simulation einer Unternehmensanwendung untersucht. Hierzu erfolgten zwei Simulationsdurchläufe – ein Durchlauf ohne die Verwendung von Antwortzeit-Komponentenmodellen und ein Durchlauf mit Komponentenmodellen. Sämtliche Elemente der Simulation blieben stabil, so dass aus den Unterschieden der Simulationsläufe auf den Effekt der Verwendung der Komponentenmodelle geschlossen werden kann.

Abbildung 6.1 stellt die Unterschiede der Simulationsergebnisse grafisch dar.

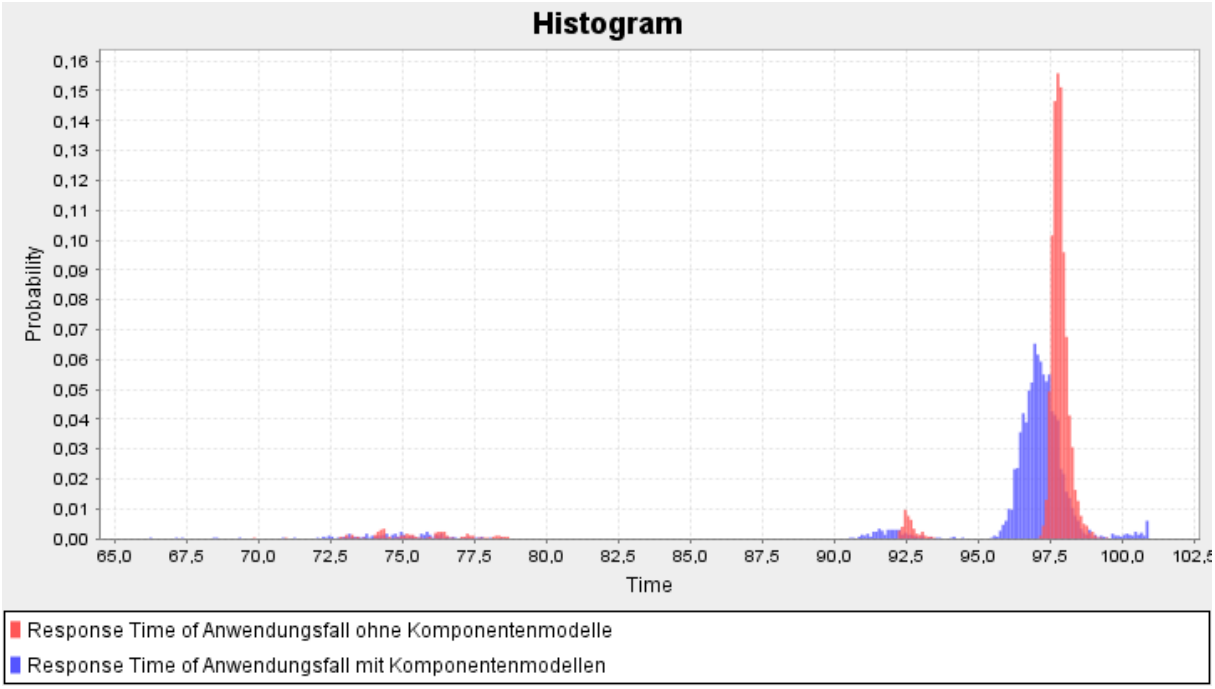


Abbildung 6.1. Vergleich der Antwortzeitverteilungen der Simulationsdurchläufe mit und ohne Komponentenmodelle. Quelle: Eigene Darstellung.

Vergleicht man die Simulationsergebnisse mit den gemessenen Lasttest-Ergebnissen des Anwendungsfalls, so erkennt man, dass beide Simulationsergebnisse eine Abweichung beinhalten. Eine genauere Untersuchung führt jedoch zu der Erkenntnis, dass die Abweichung der Simulationsergebnisse, die mit der Verwendung der Komponentenmodelle durchgeführt wurde, geringer ist als bei der Simulation ohne die Komponentenmodelle. Auffällig ist insbesondere, dass die Struktur der Antwortzeitverteilung bei der Verwendung von Komponentenmodellen stärker der real gemessenen Verteilung entspricht. Auch bei der durchschnittlichen und maximalen Abweichung des Simulationsergebnisses von den Messwerten liefert die Simulation mit Komponentenmodellen bessere Ergebnisse. Berücksichtigt man, dass im beschriebenen Anwendungsfall nur vier der insgesamt 47 Service-Operationen durch Antwortzeitkurven ersetzt werden konnten, so wird der Einfluss der Komponentenmodelle deutlich.

Die Beantwortung der dritten Forschungsfrage lautet damit:

Antwort 3: Der Einsatz von generierten Komponentenmodellen führt im Fallbeispiel zu einer Verbesserung der Simulationsergebnisse. Sowohl die simulierten Einzelwerte als auch die Verteilung der Antwortzeiten liegen bei der Verwendung der Komponentenmodelle näher an den gemessen Werten.

Durch die Verwendung der Komponentenmodelle ist eine Annäherung der simulierten Antwortzeitverteilung an die gemessenen Antwortzeiten zu beobachten. Dies betrifft vor allem die Struktur der Antwortzeitverteilung: die Simulationsergebnisse mit Komponentenmodellen bilden die gemessene Streuung deutlich besser ab als die Verwendung von Normalverteilungen zur Abbildung des Antwortzeitverhaltens einer Service-Operation. Dennoch ist im Anwendungsfall eine Diskrepanz zwischen den Simulationsergebnissen und den gemessenen Antwortzeiten gegeben. Diese ist ersten Analysen zufolge auf nicht in der Simulation abgebildete Korrelationen zwischen den verwendeten Services zurückzuführen. Da diese Korrelationen auch in den Lasttests wirkten, aus denen die Daten stammen, die zur Generierung der Komponentenmodelle verwendet werden gleichen die Komponentenmodelle dieses Fehlen zum Teil aus. Komplett ist dies jedoch nicht der Fall. Dennoch ist die Verbesserung deutlich, insbesondere unter der Berücksichtigung, dass im Anwendungsfall nur vier von insgesamt 47 Service-Operationen durch Komponentenmodelle abgebildet werden konnten.

Die Simulation des Anwendungsfalls zeigt zudem, dass bei einem einheitlichen Lasttestergebnis-Standard kaum manueller Aufwand für die Generierung der

Komponentenmodelle durch den evolutionären Algorithmus und die Einbindung der Modelle mittels der *Formula*-PCI-Implementierung in die Simulation erforderlich ist. Der gewonnenen Verbesserung des Simulationsergebnisses steht damit kaum zusätzlicher manueller Aufwand gegenüber.

6.2 Ausblick

Im Rahmen dieser Arbeit wurde gezeigt, dass mithilfe von Antwortzeit-Komponentenmodellen, die durch einen evolutionären Algorithmus generiert wurden, die Performance von Unternehmensanwendungen, deren Komponenten nicht komplett in ihrem Ressourcenverbrauch messbar sind, simuliert werden kann. Es erfolgte eine Implementierung des evolutionären Algorithmus sowie der Integration der durch den Algorithmus generierten Komponentenmodelle in das Simulations-Framework PCM. Die Anwendbarkeit wurde an einem Fallbeispiel gezeigt.

Evolutionäre Algorithmen bilden jedoch nur einen Ansatz, um Antwortzeit-Komponentenmodelle zu erzeugen. Regressionsverfahren wie die Methode der kleinsten Quadrate (Schwarz/Köckler 2004) bilden eine numerische Alternative, die in dieser Arbeit nicht betrachtet wurde. Auch andere Ansätze des maschinellen Lernens, wie beispielsweise der Artificial Bee Colony-Ansatz (Karaboga/Akay 2006) oder der Breeding Swarm (Settles et al. 2005) eignen sich zur Modellierung des Komponentenverhaltens (Chawki 2012). Zukünftige Arbeiten können betrachten, inwieweit sich mittels weiterer Modellierungsverfahren Komponentenmodelle effizient erzeugen lassen.

Für eine als Komponentenmodell abgebildete Software-Komponente liegen die Performance-Informationen gängiger Weise auf zwei Arten vor: entweder als Lasttest-Resultate oder als Monitoring- und Log-Daten. Liegen die Performance-Informationen in Form von Lasttest-Resultaten vor, so ist die Anzahl der gemessenen Faktoren, die das Performance-Verhalten beeinflussen können, begrenzt. Dies ist auch im vorgestellten Anwendungsfall der Fall. Dienen stattdessen Monitoring- und Log-Daten als Quelle, so liegen Informationen über zahlreiche theoretische Einflussfaktoren auf die gemessene Antwortzeit vor, wie beispielsweise die übergebenen Parameter, die Art des Anwenders (normaler Nutzer oder Administrator), aber auch Umgebungsfaktoren wie die Anzahl paralleler Zugriffe, die Füllstände von Datenbanken, Netzwerkbelastungen und andere. Die Auswahl der relevanten Faktoren zur Reduktion der Dimensionsanzahl für die Modellierung ist keinesfalls eine triviale Aufgabe. Während die Anzahl der parallelen Zugriffe in den meisten Fällen noch recht deutlich als Einflussfaktor auf die Performance einer Komponente

identifiziert werden kann, ist es bei anderen Faktoren, wie beispielsweise der Art und Größe der übergebenen Parameter, stark abhängig von der Funktion der Komponente. In der vorliegenden Arbeit erfolgte die Dimensionsauswahl manuell anhand von Erfahrungswerten und direkten Analysen der vorhandenen Messwerte. Eine wissenschaftliche Erörterung der Dimensionsauswahl würde diese Auswahl vereinfachen und den manuellen Aufwand reduzieren.

Letztendlich bleibt noch die weitere Überprüfung des in dieser Arbeit vorgestellten Ansatzes an weitere Unternehmensanwendungen. Der in Kapitel 5 vorgestellte Anwendungsfall besitzt eine hohe Komplexität und aufgrund der Service-orientierten Architektur auch hohe Relevanz. Unternehmensanwendungen existieren jedoch in einer Vielzahl von Architekturen, die in dieser Arbeit nicht mehr beleuchtet werden konnten. Die Anwendung des vorgestellten Ansatzes auf Unternehmensanwendungen anderer Architekturtypen wird interessante Einblicke in die Generalisierbarkeit des Vorgehens liefern.

A Literaturverzeichnis

Ackley, D.H. (1987): A connectionist machine for genetic hillclimbing, Kluwer Academic Publishers, Boston 1987.

Altshuler, E.E.; Linden, D.S. (1997): Wire-antenna designs using genetic algorithms. In: Antennas and Propagation Magazine, IEEE, Vol. 39 (1997) No. 2, pp. 33-43.

Bäck, T. (1996): Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, Oxford University Press, USA 1996.

Bagrodia, R.; Deeljman, E.; Docy, S.; Phan, T. (1999): Performance Prediction of Large Parallel Applications using Parallel Simulations. *7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (pp. 151-162). Atlanta, Georgia, United States: ACM.

Bailey, D.H.; Snavely, A. (2005): Performance Modeling: Understanding the Present and Predicting the Future. In: Euro-Par 2005 Parallel Processing (Vol. 3648/2005). Eds. Springer, Berlin / Heidelberg 2005, pp. 185-195.

Becker, S.; Koziolek, H.; Reussner, R. (2007): Model-based Performance Prediction with the Palladio Component Model. Paper presented at the WOSP '07: 6th International Workshop on Software and Performance, New York, NY, USA, pp. 54-65.

Becker, S.; Koziolek, H.; Reussner, R. (2009): The Palladio component model for model-driven performance prediction. In: Journal of Systems and Software, Vol. 82 (2009) No. 1, pp. 3 - 22.

Bhatia, K. (1994): Genetic algorithms and the traveling salesman problem. In: Computer Science and Engineering CSE292: New Age Algorithms, University of California at San Diego, Vol. 2 (1994).

Boel, R.; Mihaylova, L. (2006): A compositional stochastic model for real time freeway traffic simulation. In: Transportation Research Part B: Methodological, Vol. 40 (2006) No. 4, pp. 319-334.

Bögelsack, A. (2011): Performance und Skalierung von SAP ERP Systemen in virtualisierten Umgebungen. Ph.D. Thesis, Fakultät für Informatik, Technische Universität München, München, Germany 2011.

Bögelsack, A.; Jehle, H.; Wittges, H.; Schmidl, J.; Krcmar, H. (2008): An Approach to Simulate Enterprise Resource Planning Systems. In Ultes-Nitsche, U.; Moldt, D.; Augusto, J.C. (Eds.), *6th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2008, In conjunction with ICEIS 2008* (pp. 160-169). Barcelona, Spain: INSTICC PRESS.

Bögelsack, A.; Wittges, H.; Krcmar, H.; Kühnemund, H. (2011): Zachmanntest - A synthetic memory benchmark for SAP ERP systems. *13th International Conference on Enterprise Information Systems.* Beijing, China.

Bolch, G.; Greiner, S.; Meer, H.d.; Trivedi, K.S. (2006): Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. (2. ed.), Wiley 2006.

Bossel, H. (2004): Systeme, Dynamik, Simulation: Modellbildung, Analyse und Simulation komplexer Systeme, Books on Demand, Norderstedt 2004.

Bourgeois, J.; Spies, F. (1999): Performance prediction of distributed applications running on network of workstations. Paper presented at the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA, pp. 672–678.

Bourgeois, J.; Spies, F. (2000): Performance Prediction of an NAS Benchmark Program with ChronosMix Environment. *6th International Euro-Par Conference on Parallel Processing* (pp. 208-216).

Brebner, P. (2009): Service-Oriented Performance Modeling the MULE Enterprise Service Bus (ESB) Loan Broker Application. Paper presented at the 35th Euromicro Conference on Software Engineering and Advanced Applications, Patras, Greece, pp. 404-411.

Brebner, P.; O'Brien, L.; Gray, J. (2009): Performance Modeling Evolving Enterprise Service Oriented Architectures. *Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture 2009*. Cambridge, UK.

Brebner, P.C. (2008): Performance modeling for service oriented architectures. *Companion of the 30th international conference on Software engineering* (pp. 953-954). Leipzig, Germany: ACM.

Brosig, F.; Huber, N.; Kounev, S. (2011): Automated extraction of architecture-level performance models of distributed component-based systems. Paper presented at the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lawrence, Kansas, USA, pp. 183-192.

Brosig, F.; Kounev, S.; Krogmann, K. (2009): Automated extraction of palladio component models from running enterprise Java applications. Paper presented at the 4th International ICST Conference on Performance Evaluation Methodologies and Tools, Pisa, Italy.

Bryant, K. (2000): Genetic algorithms and the traveling salesman problem. Bachelor's Thesis, Department of Mathematics, Harvey Mudd College, Claremont, USA 2000.

Buck, B.; Hollingsworth, J.K. (2000): An API for Runtime Code Patching. In: International Journal on High Performance Computing Applications, Vol. 14 (2000) No. 4, pp. 317-329.

Bungartz, H.-J.; Zimmer, S.; Buchholz, M.; Pflüger, D. (2009): Die Simulationspipeline. In: Modellbildung und Simulation - Eine anwendungsorientierte Einführung. Eds. Springer-Verlag, Berlin Heidelberg 2009.

Carrington, L.; Snavely, A.; Wolter, N. (2006): A performance prediction framework for scientific applications. In: Future Generation Computer Systems, Vol. 22 (2006) No. 3, pp. 336-346.

Carrington, L.; Wolter, N.; Snavely, A. (2002): A Framework for Application Performance Prediction to Enable Scalability Understanding. In: Scaling to New Heights Workshop. Eds. Pittsburgh 2002.

Chawki, A. (2012): Machine Learning-Verfahren zur Modellierung des Performanceverhaltens von Softwarekomponenten. Bachelor's Thesis, Lehrstuhl für Wirtschaftsinformatik, Technische Universität München, München, Germany 2012.

Chen, S.; Liu, Y.; Gorton, I.; Liu, A. (2005): Performance prediction of component-based applications. In: Journal of Systems and Software, Vol. 74 (2005) No. 1, pp. 35-43.

Chou, C.W. (2003): Blade server module. Google Patents.

Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. (2001): Web services description language (WSDL) 1.1. http://www.w3.org/TR/wsdl, accessed at 2012-02-10.

Coello, C.A.; Lamont, G.B.; Van Veldhuizen, D.A. (2007): Evolutionary Algorithms for Solving Multi-Objective Problems. (2. ed.), Springer Verlag 2007.

Darwin, C. (1859): On the origin of species by means of natural selection, John Murray, London, United Kingdom 1859.

De Jong, K. (1988): Learning with genetic algorithms: An overview. In: Machine learning, Vol. 3 (1988) No. 2, pp. 121-138.

De Jong, K.A.; Spears, W.M. (1989): Using genetic algorithms to solve NP-complete problems. Paper presented at the 3rd International Conference on Genetic Algorithms, Washington DC, USA.

Deb, K. (2001): Multi-objective optimization using evolutionary algorithms, Wiley 2001.

DeRose, L.; Ekanadham, K.; Hollingsworth, J.K.; Sbaraglia, S. (2002): SIGMA: A Simulator Infrastructure to Guide Memory Analysis. Paper presented at the ACM/IEEE 2002 Conference on Supercomputing, Baltimore, USA.

Dongarra, J. (1988): The LINPACK Benchmark: An explanation In: Supercomputing (Vol. 297). Eds.: Houstis, E.; Papatheodorou, T.; Polychronopoulos, C. Springer Berlin / Heidelberg 1988, pp. 456-474.

Doran, G.T. (1981): There's a SMART way to write management's goals and objectives. In: Management Review, Vol. 70 (1981) No. 11, pp. 35-36.

Dustdar, S.; Gall, H.; Hauswirth, M. (2003): Komponentenmodelle. In: Software-Architekturen für Verteilte Systeme. Eds. Springer Verlag, Berlin / Heidelberg 2003, pp. 199-210.

Eiben, A.E.; Smith, J.E. (2003a): Evolutionary Computing: Why? In: Introduction to evolutionary computing. Eds.: Rozenberg, G. Springer Verlag, Berlin / Heidelberg 2003a, pp. 9-12.

Eiben, A.E.; Smith, J.E. (2003b): Introduction to evolutionary computing, Springer Verlag, Berlin / Heidelberg 2003b.

Erk, K.; Priese, L. (2009): NP-vollständige Probleme. In: Theoretische Informatik: Eine umfassende Einführung. Eds. Springer Verlag, Berlin / Heidelberg 2009, pp. 453-454.

Fettke, P. (2006): State-of-the-Art des State-of-the-Art. In: Wirtschaftsinformatik, Vol. 48 (2006) No. 4, pp. 257-266.

Fonseca, C.M.; Fleming, P.J. (1995): An Overview of Evolutionary Algorithms in Multiobjective Optimization. In: Evolutionary Computation, Vol. 3 (1995) No. 1, pp. 1-16.

Fonseca, C.M.; Fleming, P.J. (1998): Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms – Part I: A Unified Formulation. In: IEEE Transaction on Systems, Man, and Cybernetics, Part A: Systems and Humans, Vol. 28 (1998) No. 1, pp. 26-37.

Fowler, M. (2003a): Distribution Patterns. In: Patterns of Enterprise Application Architecture. Eds. Addison-Wesley, Amsterdam 2003a, pp. 388-400.

Fowler, M. (2003b): Introduction. In: Patterns of Enterprise Application Architecture. Eds. Addison-Wesley, Amsterdam 2003b, pp. 1-13.

Franks, G.; Al-Omari, T.; Woodside, M.; Das, O.; Derisavi, S. (2009): Enhanced Modeling and Solution of Layered Queueing Networks. In: IEEE Transactions on Software Engineering, Vol. 35 (2009) No. 2, pp. 148-161.

Franks, G.; Maly, P.; Woodside, M.; Petriu, D.C.; Hubbard, A. (2005): Layered Queueing Network Solver and Simulator User Manual. Ottawa, Canada: Department of Systems and Computer Engineering, Carleton University.

Gamma, E. (1995a): Creational Patterns. In: Design Patterns: Elements of Reusable Object-Oriented Software. Eds.: Kernighan, B.W. Addison-Wesley Professional 1995a, pp. 81-136.

Gamma, E. (1995b): Facade. In: Design Patterns: Elements of Reusable Object-Oriented Software. Eds.: Kernighan, B.W. Addison-Wesley Professional 1995b, pp. 185-193.

Gartner Inc. (2011): Gartner Says Worldwide Application Infrastructure and Middleware Market Revenue Increased 7.3 Percent in 2010. http://www.gartner.com/it/page.jsp?id=1632714, accessed at 2012-02-12.

Georgii, H.-O. (2009): Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik, de Gruyter Lehrbuch, Berlin 2009.

Goldberg, D.E. (1989): Genetic algorithms in search, optimization, and machine learning, Addison-Wesley Professional, Upper Saddle River, NJ, USA 1989.

Gomaa, H.; Menascé, D.A. (2000): Design and performance modeling of component interconnection patterns for distributed software architectures. Paper presented at the 2nd International Workshop on Software and Performance, Ottawa, Canada, pp. 117-126.

Gorton, I.; Liu, A.; Brebner, P. (2003): Rigorous evaluation of COTS middleware technology. In: Computer, Vol. 36 (2003) No. 3, pp. 50-55.

Gradl, S. (2012): Performance-Modellierung und Simulation eines SAP-ERP-Systems. Ph.D. Thesis, Fakultät für Informatik, Technische Universität München, München, Germany 2012.

Gradl, S.; Bögelsack, A.; Wittges, H.; Krcmar, H. (2009): Layered Queuing Networks for Simulating Enterprise Resource Planning Systems. Paper presented at the 7th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2009, In conjunction with ICEIS 2009, Milan, Italy, pp. 85-92.

Gradl, S.; Mayer, M.; Danciu, A.; Wittges, H.; Krcmar, H. (2011): Understanding the Performance Behavior of a SAP ERP System for the Use of Queuing Models. Paper presented at the 10th International Conference on Modeling and Applied Simulation, MAS 2011, Rende, Italy, pp. 361-370.

Griewank, A. (1981): Generalized descent for global optimization. In: Journal of optimization theory and applications, Vol. 34 (1981) No. 1, pp. 11-39.

Gwozdz, P.; Szlachcic, E. (2009): An Adaptive Selection Evolutionary Algorithm for the Capacitated Vehicle Routing Problem. Paper presented at the 2nd International Symposium on Logistics and Industrial Informatics, Linz, Austria, pp. 1-6.

Happe, J.; Becker, S.; Rathfelder, C.; Friedrich, H.; Reussner, R. (2010): Parametric performance completions for model-driven performance prediction. In: Performance Evaluation, Vol. 67 (2010) No. 8, pp. 694-716.

Heineman, G.T.; Councill, W.T. (2001): Component-based Software Engineering: Putting the Pieces Together (Vol. 67), Addison-Wesley Longman, Amsterdam, Netherlands 2001.

Hevner, A.R.; March, S.T.; Park, J.; Ram, S. (2004): Design Science in Information Systems Research. In: MIS Quarterly, Vol. 28 (2004) No. 1, pp. 77-105.

Hohpe, G.; Woolf, B. (2003a): Channel Adapter. In: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions Eds. Addison-Wesley Longman, Amsterdam 2003a, pp. 127-132.

Hohpe, G.; Woolf, B. (2003b): Integration Styles. In: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions Eds. Addison-Wesley Longman, Amsterdam 2003b, pp. 39-56.

Hohpe, G.; Woolf, B. (2003c): Message Bus. In: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions Eds. Addison-Wesley Longman, Amsterdam 2003c, pp. 137-141.

Hohpe, G.; Woolf, B. (2003d): Solving Integration Problems Using Patterns. In: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions Eds. Addison-Wesley Longman, Amsterdam 2003d, pp. 1-37.

Hollingsworth, J.K.; Snavely, A.; Sbaraglia, S.; Ekanadham, K. (2005): EMPS: An Environment for Memory Performance Studies. *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 10 - Volume 11* (pp. 223.2): IEEE Computer Society.

Hossain, E.; Babar, M.A.; Hye-Young, P. (2009): Using Scrum in Global Software Development: A Systematic Literature Review. Paper presented at the 4th IEEE International Conference on Global Software Engineering, Limerick, Ireland, pp. 175-184.

IBM (2012): IBM WebSphere - Software - Deutschland. http://www-01.ibm.com/software/de/websphere/, accessed at 2012-05-20.

Jacobson, P.A.; Lazowska, E.D. (1982): Analyzing queueing networks with simultaneous resource possession. In: Communications of the ACM, Vol. 25 (1982) No. 2, pp. 142-151.

Jain, R. (1991): Introduction to Simulation. In: The art of computer systems performance analysis. Eds. 2nd (ed.). John Wiley & Sons, Inc., New York 1991, pp. 393-422.

Jarvis, S.A.; Spooner, D.P.; Lim Choi Keung, H.N.; Cao, J.; Saini, S.; Nudd, G.R. (2006): Performance prediction and its use in parallel and distributed computing systems. In: Future Generation Computer Systems, Vol. 22 (2006) No. 7, pp. 745-754.

Jehle, H. (2010): Performance-Messung eines Portalsystems in virtualisierter Umgebung am Fallbeispiel SAP. Ph.D. Thesis, Fakultät für Informatik, Technische Universität München, München, Germany 2010.

Johnson, G.L.; Hanson, C.L.; Hardegree, S.P.; Ballard, E.B. (1996): Stochastic Weather Simulation: Overview and Analysis of Two Commonly Used Models. In: Journal of Applied Meteorology, Vol. 35 (1996) No. 10, pp. 1878-1896.

Justesen, P.D. (2009): Multi-objective Optimization using Evolutionary Algorithms. Department of Computer Science, University of Aarhus, 2009.

Kappler, T.; Koziolek, H.; Krogmann, K.; Reussner, R. (2008): Towards automatic construction of reusable prediction models for component-based performance engineering. In: Proceedings on Software Engineering, Vol. 121 (2008), pp. 140-154.

Karaboga, D.; Akay, B. (2006): Artificial bee colony (ABC), harmony search and bees algorithms on numerical optimization. Paper presented at the IEEE Swarm Intelligence Symposium, Indianapolis, Indiana, USA.

Kerbyson, D.J.; Jones, P.W. (2005): A Performance Model of the Parallel Ocean Program. In: International Journal of High Performance Computing Applications, Vol. 19 (2005) No. 3, pp. 261-276.

Kienzle, M.G.; Sevcik, K.C. (1979): Survey of analytic queueing network models of computer systems. In: SIGSIM Simulation Digest, Vol. 11 (1979) No. 1, pp. 113-129.

Kitchenham, B.; Charters, S. (2007): Guidelines for performing systematic literature reviews in software engineering. Software Engineering Group, School of Computer Science and Mathematics, Keele University, and Department of Computer Science, University of Durham, 2007.

Kounev, S. (2006): Performance modeling and evaluation of distributed component-based systems using queueing petri nets. In: IEEE Transactions on Software Engineering, Vol. 32 (2006) No. 7, pp. 486-502.

Kounev, S.; Buchmann, A. (2003): Performance modeling and evaluation of large-scale J2EE applications. Paper presented at the 29th International Conference of the Computer Measurement Group on Resource Management and Performance Evaluation of Enterprise Computing Systems, Dallas, Texas, USA, pp. 273-284.

Koza, J.R. (1992a): The Fitness Measure. In: Genetic Programming, On the Programming of Computers by Means of Natural Selection. Eds. MIT Press, Cambridge, MA, USA 1992a, pp. 624-628.

Koza, J.R. (1992b): Genetic Programming, On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, USA 1992b.

Koziolek, H.; Happe, J.; Becker, S.; Reussner, R. (2007): Evaluating Performance of Software Architecture Models with the Palladio Component Model. http://sdqweb.ipd.kit.edu/mediawiki-sdq-extern/images/0/03/Chapter-PCM.pdf, accessed at 2012-09-02.

Kraft, S.; Pacheco-Sanchez, S.; Casale, G.; Dawson, S. (2009): Estimating service resource consumption from response time measurements. *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools.* Pisa, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Krafzig, D.; Banke, K.; Slama, D. (2005a): Elements of a Service-Oriented Architecture. In: Enterprise SOA: Service-Oriented Architecture Best Practices. Eds. Prentice Hall PTR 2005a, pp. 59-65.

Krafzig, D.; Banke, K.; Slama, D. (2005b): Enterprise Software is a Different Animal. In: Enterprise SOA: Service-Oriented Architecture Best Practices. Eds. Prentice Hall PTR 2005b, pp. 3-4.

Krafzig, D.; Banke, K.; Slama, D. (2005c): Evolution of the Service Concept. In: Enterprise SOA: Service-Oriented Architecture Best Practices. Eds. Prentice Hall PTR 2005c, pp. 13-25.

Krafzig, D.; Banke, K.; Slama, D. (2005d): Software Buses and the Service Bus. In: Enterprise SOA: Service-Oriented Architecture Best Practices. Eds. Prentice Hall PTR 2005d, pp. 159-170.

Krafzig, D.; Banke, K.; Slama, D. (2005e): What is a Service-Oriented Architecture? In: Enterprise SOA: Service-Oriented Architecture Best Practices. Eds. Prentice Hall PTR 2005e, pp. 56-62.

L'Ecuyer, P.; Meliani, L.; Vaucher, J. (2002): SSJ: A Framework for Stochastic Simulation in Java. Paper presented at the 34th Conference on Winter Simulation, San Diego, USA, pp. 234-242.

Lämmel, U.; Cleve, J. (2008): Künstliche Intelligenz. (Third ed.), Hanser Verlag 2008.

Liu, T.-K.; Behroozi, A.; Kumaran, S. (2003): A performance model for a business process integration middleware. Paper presented at the IEEE International Conference on E-Commerce, Newport Beach, California, USA, pp. 191-198.

Liu, Y.; Fekete, A.; Gorton, I. (2004): Predicting the performance of middleware-based applications at the design level. In: ACM SIGSOFT Software Engineering Notes, Vol. 29 (2004) No. 1, pp. 166-170.

Ludewig, J.; Lichter, H. (2007a): Engineering und Ingenieur. In: Software Engineering - Grundlagen, Menschen, Prozesse, Techniken. Eds.: Zimpfer, U., (1. ed.). dpunkt.verlag GmbH, Heidelberg 2007a, pp. 31-34.

Ludewig, J.; Lichter, H. (2007b): Programmtest. In: Software Engineering - Grundlagen, Menschen, Prozesse, Techniken. Eds.: Zimpfer, U., (1. ed.). dpunkt.verlag GmbH, Heidelberg 2007b, pp. 445-504.

Mayer, M. (2013): Performance-Modellierung und Simulation eines SAP-Netweaver-Portal-Systems, Fakultät für Informatik, Technische Universität München, München, Germany 2013.

Menascé, D. (1997): A Framework for Software Performance Engineering of Client/Server Systems. Paper presented at the International Conference of the Computer Measurement Group on Resource Management and Performance Evaluation of Enterprise Computing Systems, Orlando, Florida, USA, pp. 460-469.

Menasce, D.A.; Almeida, V.A.F.; Dowdy, L. (2004): Basic Performance Results. In: Performance by Design: Computer Capacity Planning by Example. Eds. Prentice Hall 2004, pp. 61-74.

Menascé, D.A.; Gomaa, H. (1998): On a language based method for software performance engineering of client/server systems. *1st International Workshop on Software and Performance* (pp. 63-69). Santa Fe, New Mexico, United States: ACM.

Mühlenbein, H.; Schomisch, M.; Born, J. (1991): The parallel genetic algorithm as function optimizer. In: Parallel computing, Vol. 17 (1991) No. 6-7, pp. 619-632.

MuleSoft (2012): Mule ESB - Open Source ESB Community w. Documentation | Mule ESB Community. http://www.mulesoft.org/, accessed at 2012-06-03.

Munz, C.D.; Westermann, T. (2009): Numerische Behandlung gewöhnlicher und partieller Differenzialgleichungen, Springer Verlag Berlin / Heidelberg 2009.

Nudd, G.R.; Kerbyson, D.J.; Papaefstathiou, E.; Perry, S.C.; Harper, J.S.; Wilcox, D.V. (2000): Pace - A Toolset for the Performance Prediction of Parallel and Distributed Systems. In: International Journal of High Performance Computing Applications, Vol. 14 (2000) No. 3, pp. 228-251.

OASIS (2012): UDDI - Online community for the Universal Description, Discovery, and Integration. http://uddi.xml.org/, accessed at 2012-02-10.

Object Management Group (2012): OMG CORBA Website. http://www.corba.org/, accessed at 2012-04-05.

Ochi, L.S.; Vianna, D.S.; Drummond, L.; Victor, A.O. (1998): A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. In: Future Generation Computer Systems, Vol. 14 (1998) No. 5-6, pp. 285-292.

Oracle (2011): Task Flow Design Fundamentals. http://www.oracle.com/technetwork/developer-tools/jdev/adf-task-flow-design-132904.pdf, accessed at 2012-02-22.

Oracle (2012a): Enterprise Manager. http://www.oracle.com/de/products/enterprise-manager/index.html, accessed at 2012-09-14.

Oracle (2012b): Java Management Extensions (JMX). http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html, accessed at 2012-12-02.

Oracle (2012c): Middleware | Fusion Middleware 11g | Oracle. http://www.oracle.com/de/products/middleware/index.html, accessed at 2012-05-20.

Oracle (2012d): Oracle Service Bus. http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html, accessed at 2012-09-14.

Oracle (2012e): Oracle Web Services Manager. http://www.oracle.com/technetwork/middleware/webservices-manager/index.html, accessed at 2012-09-17.

Parpinelli, R.S.; Lopes, H.S. (2011): An eco-inspired evolutionary algorithm applied to numerical optimization. Paper presented at the Thrid World Congress on Nature and Biologically Inspired Computing (NaBIC), Salamanca, Spain, pp. 466-471.

Perry, B.W. (2003): Java Servlet and JSP Cookbook, O'Reilly & Associates, Inc., Sebastopol, CA, USA 2003.

Pllana, S.; Benkner, S.; Xhafa, F.; Barolli, L. (2008): Hybrid Performance Modeling and Prediction of Large-Scale Computing Systems. Paper presented at the International Conference on Complex, Intelligent and Software Intensive Systems, Barcelona, Spain, pp. 132-138.

Potter, M.; De Jong, K. (1994): A cooperative coevolutionary approach to function optimization. Paper presented at the 3rd Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, pp. 249-257.

Prechelt, L. (2001): Kontrollierte Experimente in der Softwaretechnik: Potenzial und Methodik, Springer Verlag, Heidelberg 2001.

Prins, C. (2004): A simple and effective evolutionary algorithm for the vehicle routing problem. In: Computers & Operations Research, Vol. 31 (2004) No. 12, pp. 1985-2002.

Puljic, K.; Manger, R. (2005): An evolutionary algorithm with repeated mutations for solving the vehicle routing problem. Paper presented at the 27th International Conference on Information Technology Interfaces, Cavtat, Croatia, pp. 478-483.

Red Hat (2012): Red Hat | JBoss Enterprise Middleware. Open source middleware. http://www.redhat.com/products/jbossenterprisemiddleware/, accessed at 2012-06-20.

Reussner, R.; Becker, S.; Happe, J.; Koziolek, H.; Krogmann, K.; Kuperberg, M. (2007): The palladio component model. In: Interner Bericht, Vol. 21 (2007).

Rolia, J.; Casale, G.; Krishnamurthy, D.; Dawson, S.; Kraft, S. (2009): Predictive modelling of SAP ERP applications: challenges and solutions. *Fourth International ICST Conference on Performance Evaluation Methodologies and Tools.* Pisa, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Rolia, J.A.; Sevcik, K.C. (1995): The Method of Layers. In: IEEE Transactions on Software Engineering, Vol. 21 (1995) No. 8, pp. 689-700.

Rosenbrock, H.H. (1960): An automatic method for finding the greatest or least value of a function. In: The Computer Journal, Vol. 3 (1960) No. 3, pp. 175-184.

Sbalzarini, I.F.; Müller, S.; Koumoutsakos, P. (2000): Multiobjective optimization using evolutionary algorithms. Paper presented at the Summer Program, Center for Turbulence Research, NASA, Stanford, CA, USA, pp. 63-74.

Scheffermann, R.; Bender, M.; Cardeneo, A. (2009): Robust solutions for vehicle routing problems via evolutionary multiobjective optimization. Paper presented at the IEEE Congress on Evolutionary Computation, Trondheim, Norway, pp. 1605-1612.

Schwarz, H.R.; Köckler, N. (2004): Numerische Mathematik. (5. ed.), Teubner, Stuttgart 2004.

Schwetman, H.D. (1978): Hybrid simulation models of computer systems. In: Communications of the ACM, Vol. 21 (1978) No. 9, pp. 718-723.

Settles, M.; Nathan, P.; Soule, T. (2005): Breeding swarms: a new approach to recurrent neural network training. Paper presented at the Conference on Genetic and Evolutionary Computation Washington DC, USA, pp. 185-192.

Simon, H.A. (1996): The Sciences of the Artificial, The MIT Press, Cambridge, MA, USA 1996.

Snavely, A.; Carrington, L.; Wolter, N.; Labarta, J.; Badia, R.; Purkayastha, A. (2002): A framework for performance modeling and prediction. Paper presented at the 2002 ACM/IEEE Conference on Supercomputing, Baltimore, Maryland, pp. 1-17.

Software Design and Quality Group (2012): Model Solvers: Palladio Software Architecture Simulator. http://www.palladio-simulator.com/science/palladio_component_model/model_solvers/, accessed at 2012-11-18.

SourceForge (2012): Jep Java - Math Expression Parser. http://sourceforge.net/projects/jep/, accessed at 2012-03-11.

Sun, X.-H.; Fahringer, T.; Pantano, M. (2002): SCALA: A Performance System For Scalable Computing. In: International Journal of High Performance Computing Applications, Vol. 16 (2002) No. 4, pp. 357-370.

Szyperski, C.; Gruntz, D.; Murer, S. (2002): Component software: beyond object-oriented programming, Addison-Wesley Professional 2002.

Tertilt, D.; Boegelsack, A.; Krcmar, H. (2012): Modeling the Performance and Scalability of a SAP ERP System using an Evolutionary Algorithm. Paper presented at the 14th International Conference on Enterprise Information Systems, Wroclaw, Poland.

Tertilt, D.; Krcmar, H. (2011): Generic Performance Prediction for ERP and SOA Applications. Paper presented at the 19th European Conference on Information Systems, Helsinki, Finland.

Tertilt, D.; Krcmar, H. (2012): Easy-to-Use SAP Sizing Based on Evolutionary Generated Scalability Models. Paper presented at the Conference on Enterprise Information Systems, Algarve, Portugal.

Tertilt, D.; Leimeister, S.; Gradl, S.; Mayer, M.; Krcmar, H. (2010): Towards an Evolutionary Model Generation for ERP Performance Simulation. Paper presented at the IADIS International Conference on Intelligent Systems and Agents 2010, Freiburg im Breisgau, Germany, pp. 124-128.

Thangiah, S.R.; Nygard, K.E.; Juell, P.L. (1991): Gideon: A genetic algorithm system for vehicle routing with time windows. Paper presented at the 7th IEEE Conference on Artificial Intelligence Applications, Washington DC, USA, pp. 322-328.

The Eclipse Foundation (2011): Eclipse - The Eclipse Foundation open source community website. http://eclipse.org/, accessed at 2011-10-14.

Tikir, M.M.; Carrington, L.; Strohmaier, E.; Snavely, A. (2007): A genetic algorithms approach to modeling the performance of memory-bound computations. 2007 *ACM/IEEE conference on Supercomputing*. Reno, Nevada: ACM.

Törn, A.; Zilinskas, A. (1989): Global Optimization. Lecture Notes in Computer Science, Nº 350. Berlin: Springer-Verlag.

Torraco, R.J. (2005): Writing integrative literature reviews: Guidelines and examples. In: Human Resource Development Review, Vol. 4 (2005) No. 3, pp. 356-367.

Universität Osnabrück (2013): WISMO - Wissensbasierte semiformale Modellierung. http://www.imwi.uni-osnabrueck.de/wismo.htm, accessed at 2013-09-08.

Webster, J.; Watson, R. (2002): Analyzing the Past to Prepare for the Future: Writing a Literature Review. In: MIS Quarterly, Vol. 26 (2002) No. 2, pp. 13-23.

Wert, A. (2011): Integration of Software Performance Curves in PCM, Faculty of Computer Science, Institute for Program Structures and Data Organization (IPD), Karlsruhe Institute of Technology, Karlsruhe, Germany 2011.

Wilhelm, K. (2003): Messung und Modellierung von SAP R/3- und Storage-Systemen für die Kapazitätsplanung. Ph.D. Thesis, Fakultät für Mathematik, Universität Duisburg-Essen, Duisburg-Essen, Germany 2003.

WineHQ.org (2012): Wine HQ. http://www.winehq.org, accessed at 2012-09-03.

Woodside, M. (2002): Tutorial Introduction to Layered Modeling of Software Performance.

Wu, X.; Woodside, M. (2004): Performance modeling from software components. In: SIGSOFT Softw. Eng. Notes, Vol. 29 (2004) No. 1, pp. 290-301.

Yang, S.Y.; Shih, C.C.; Yen, C.T.; Chen, Y.C. (2002): Blade server management system. Google Patents.

Zhong, W.; Liu, J.; Xue, M.; Jiao, L. (2004): A multiagent genetic algorithm for global numerical optimization. In: IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, Vol. 34 (2004) No. 2, pp. 1128-1141.

Zitzler, E.; Thiele, L. (1999): Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. In: IEEE Transactions on Evolutionary Computation, Vol. 3 (1999) No. 4, pp. 257 - 271.

B Composed Services

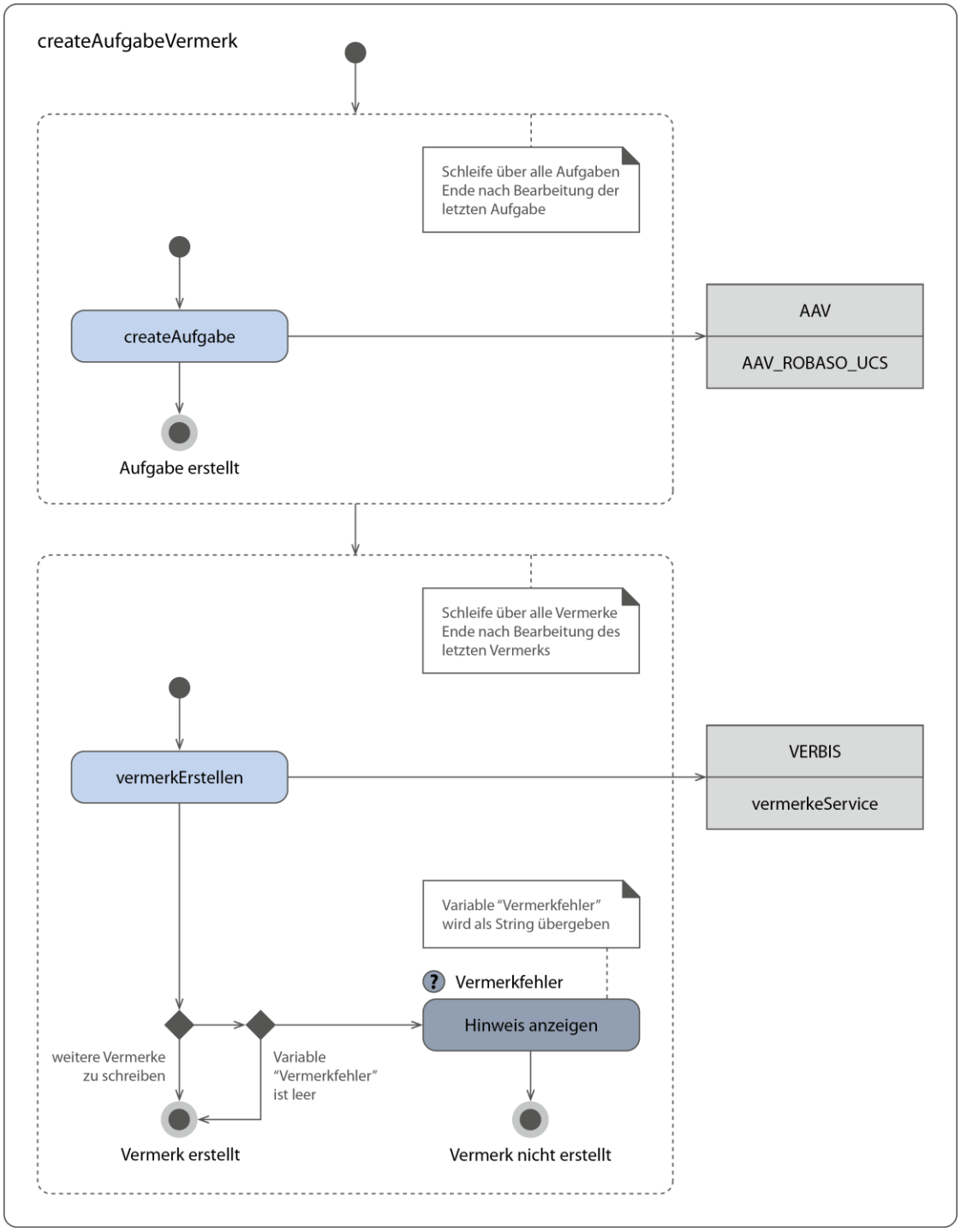


Abbildung B.1. Ablauf der Operation createAufgabeVermerk des AAV_ROBASO_UCS Services mit Darstellung der vom CS verwendeten Services. Quelle: Eigene Darstellung, nach Projektdokumentation.

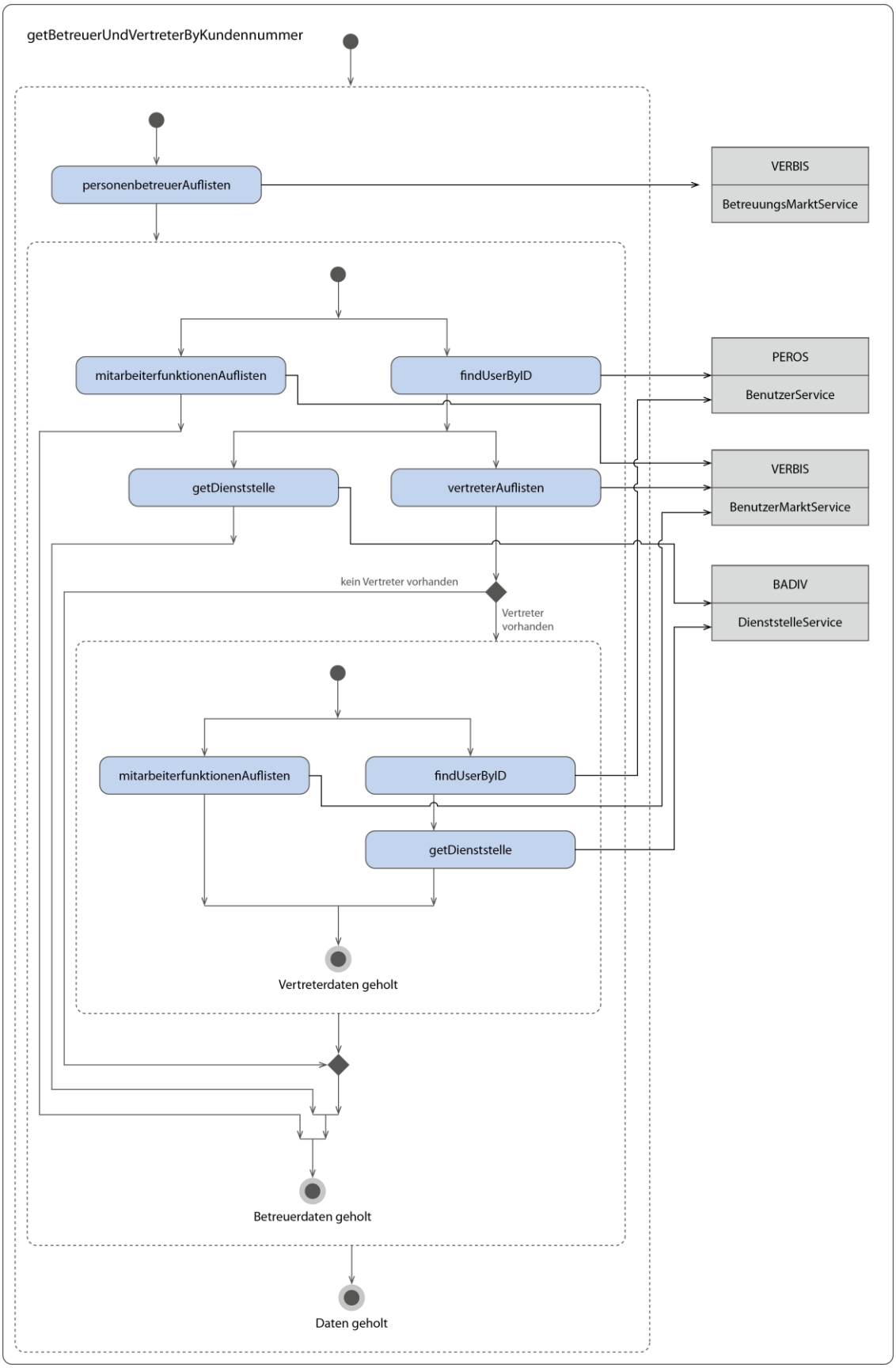
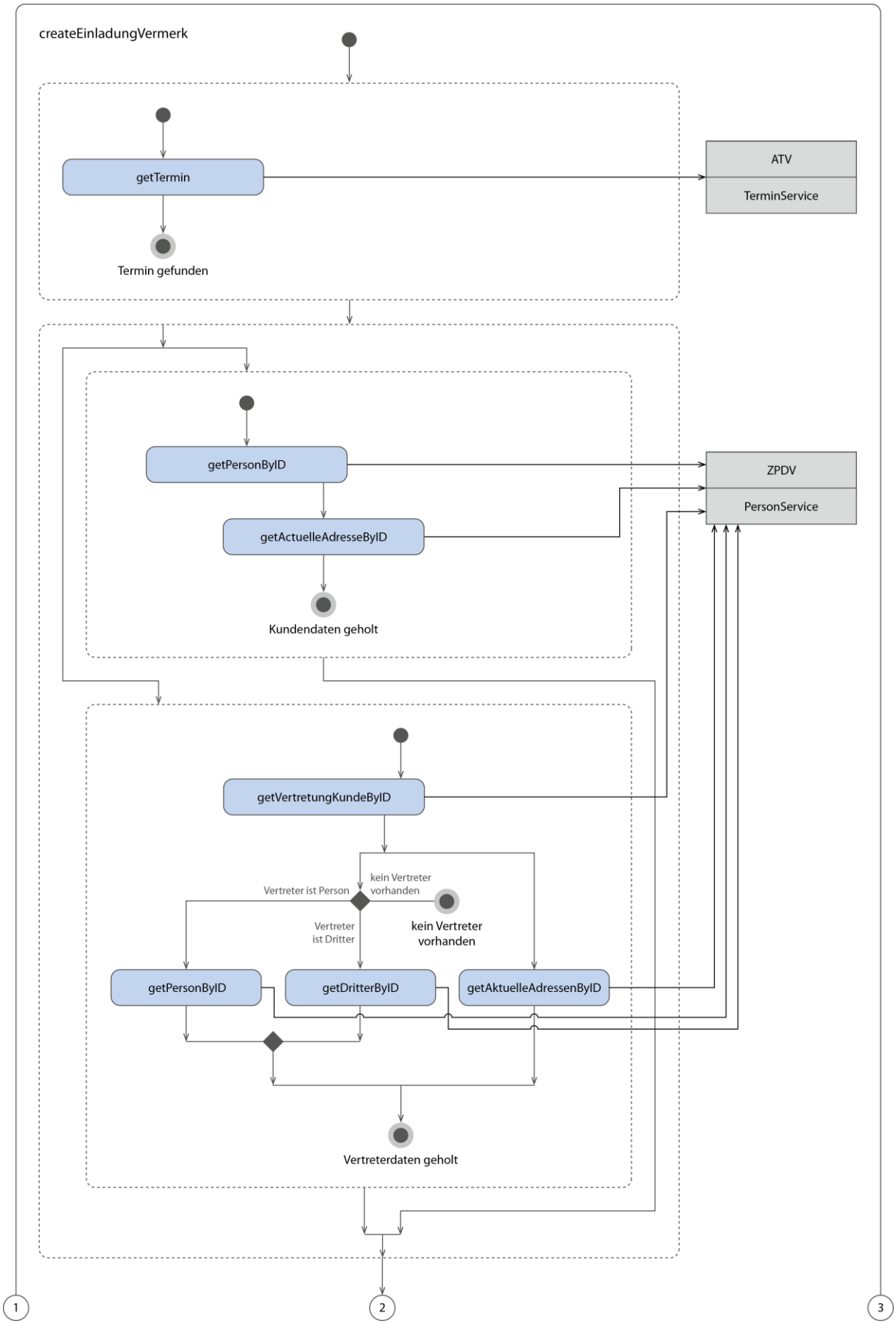


Abbildung B.2. Ablauf der Operation getBetreuerUndVertreterByKundennummer des BetreuerdatenServices mit Darstellung der vom CS verwendeten Services. Quelle: Eigene Darstellung, nach Projektdokumentation.


createEinladungVermerk
getTermin
ATV
TerminService
Termin gefunden
getPersonByID
getActuelleAdresseByID
ZPDV
PersonService
Kundendaten geholt
getVertretungKundeByID
Vertreter ist Person
kein Vertreter vorhanden
Vertreter ist Dritter
kein Vertreter vorhanden
getPersonByID
getDritterByID
getAktuelleAdressenByID
Vertreterdaten geholt
1
2
3

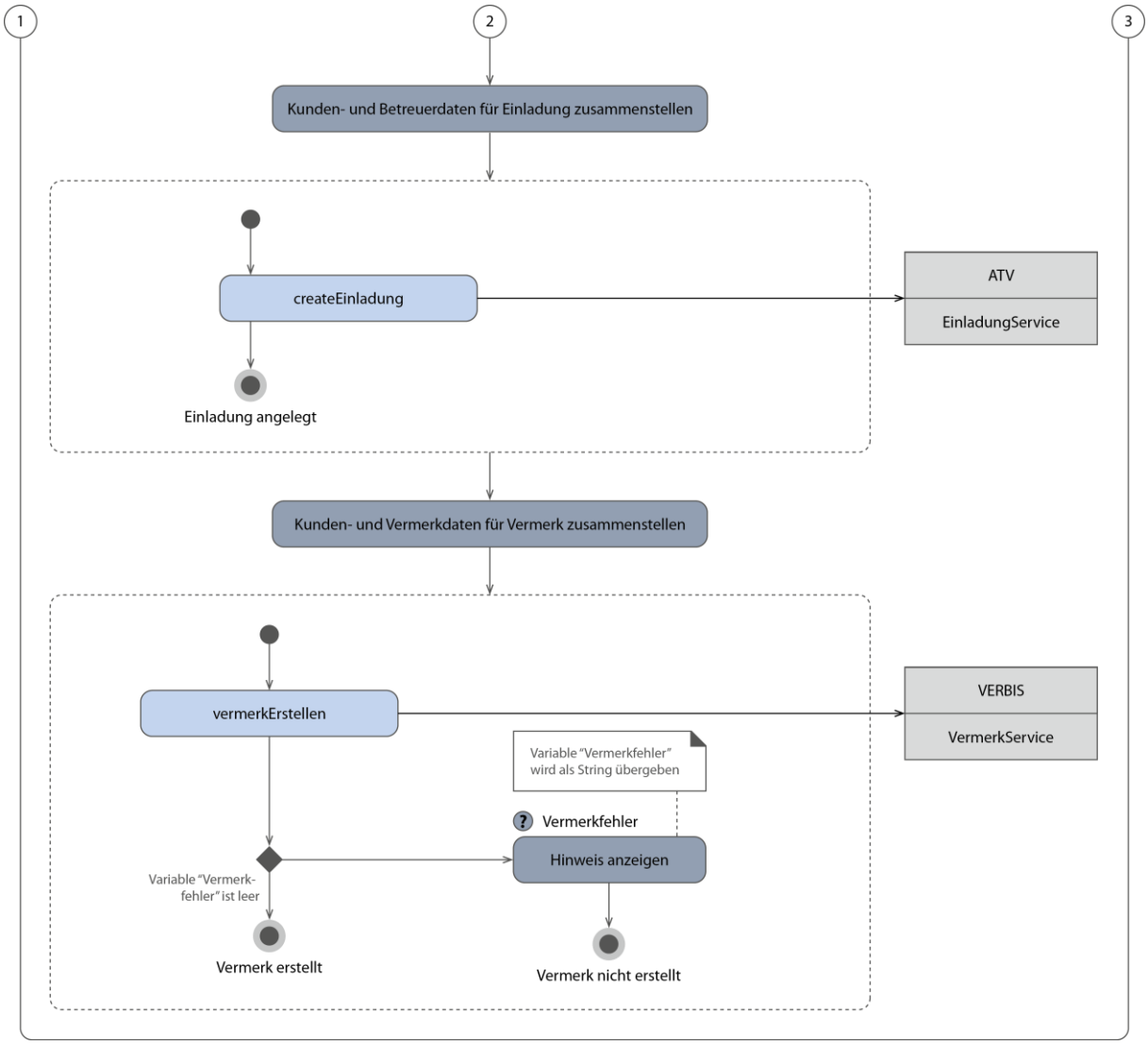


Abbildung B.3. Ablauf der Operation createEinladungVermerk des BetreuerdatenServices mit Darstellung der vom CS verwendeten Services. Quelle: Eigene Darstellung, nach Projektdokumentation.

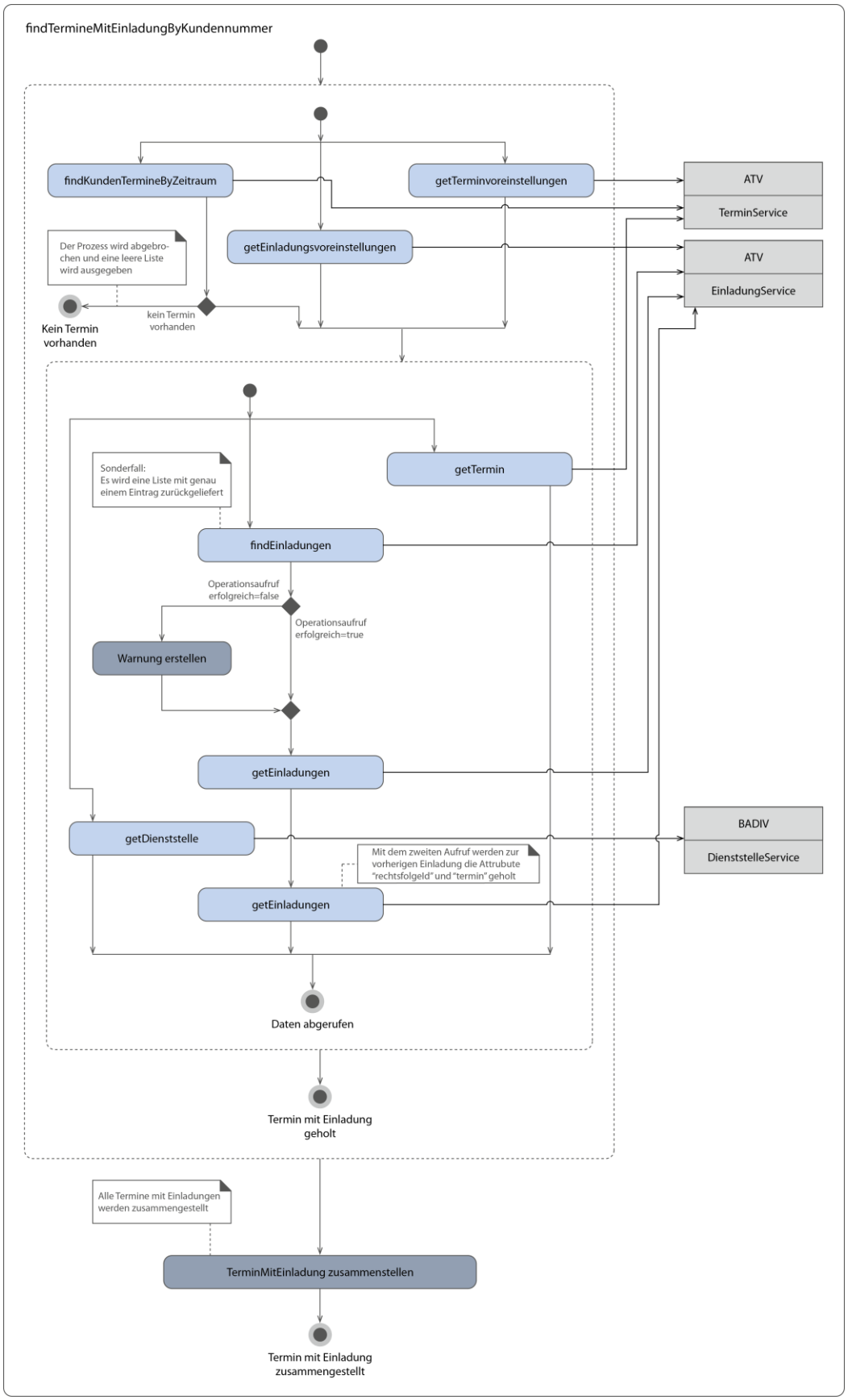
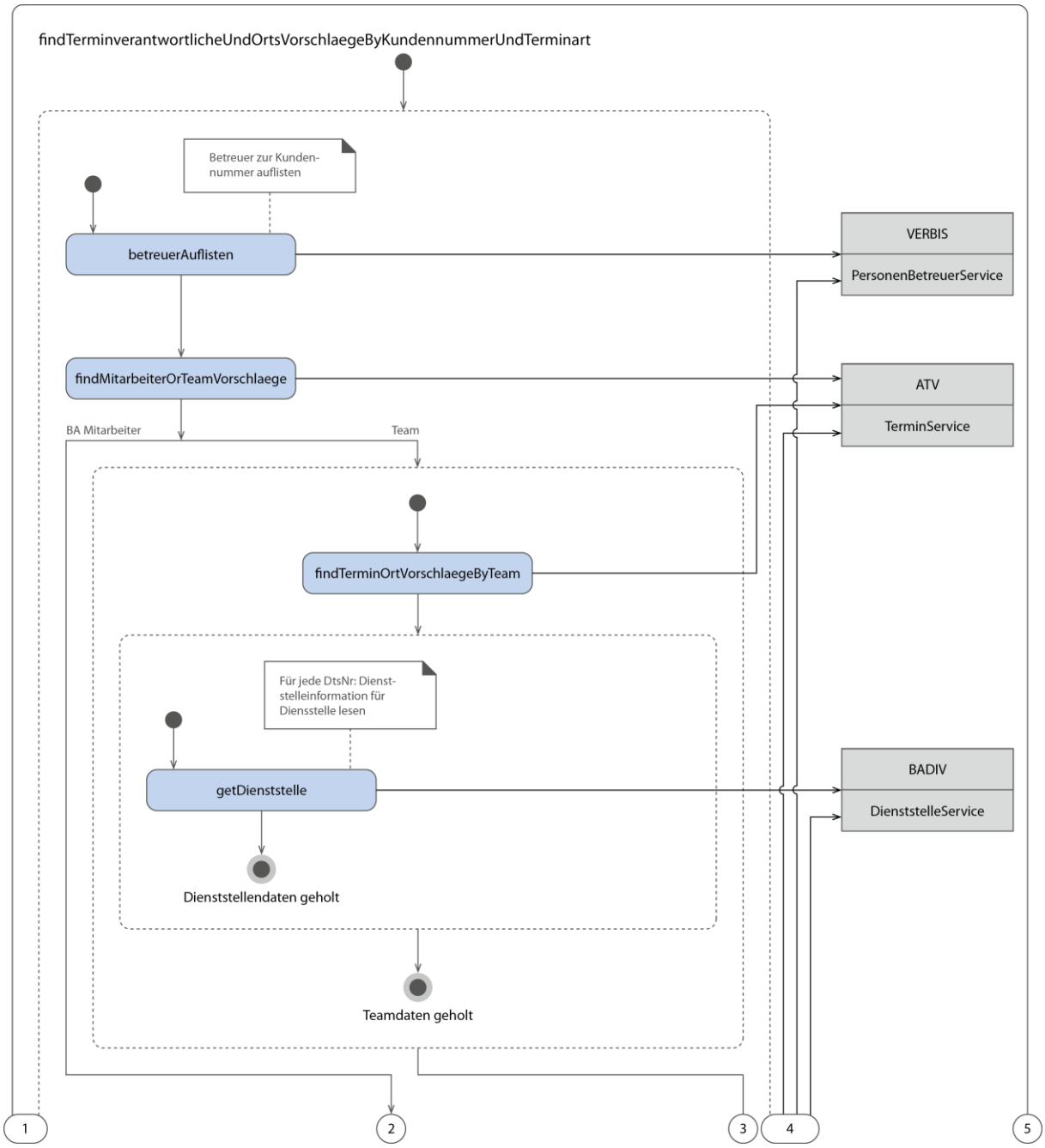


Abbildung B.4. Ablauf der Operation findTermineMitEinladungByKundennummer des TerminVerwaltungService mit Darstellung der vom CS verwendeten Services. Quelle: Eigene Darstellung, nach Projektdokumentation


findTerminverantwortlicheUndOrtsVorschlaegeByKundennummerUndTerminart
Betreuer zur Kunden-nummer auflisten
betreuerAuflisten
VERBIS
PersonenBetreuerService
findMitarbeiterOrTeamVorschlaege
ATV
TerminService
BA Mitarbeiter
Team
findTerminOrtVorschlaegeByTeam
Für jede DtsNr: Dienst-stelleinformation für Diensstelle lesen
getDienststelle
BADIV
DienststelleService
Dienststellendaten geholt
Teamdaten geholt
1
2
3
4
5

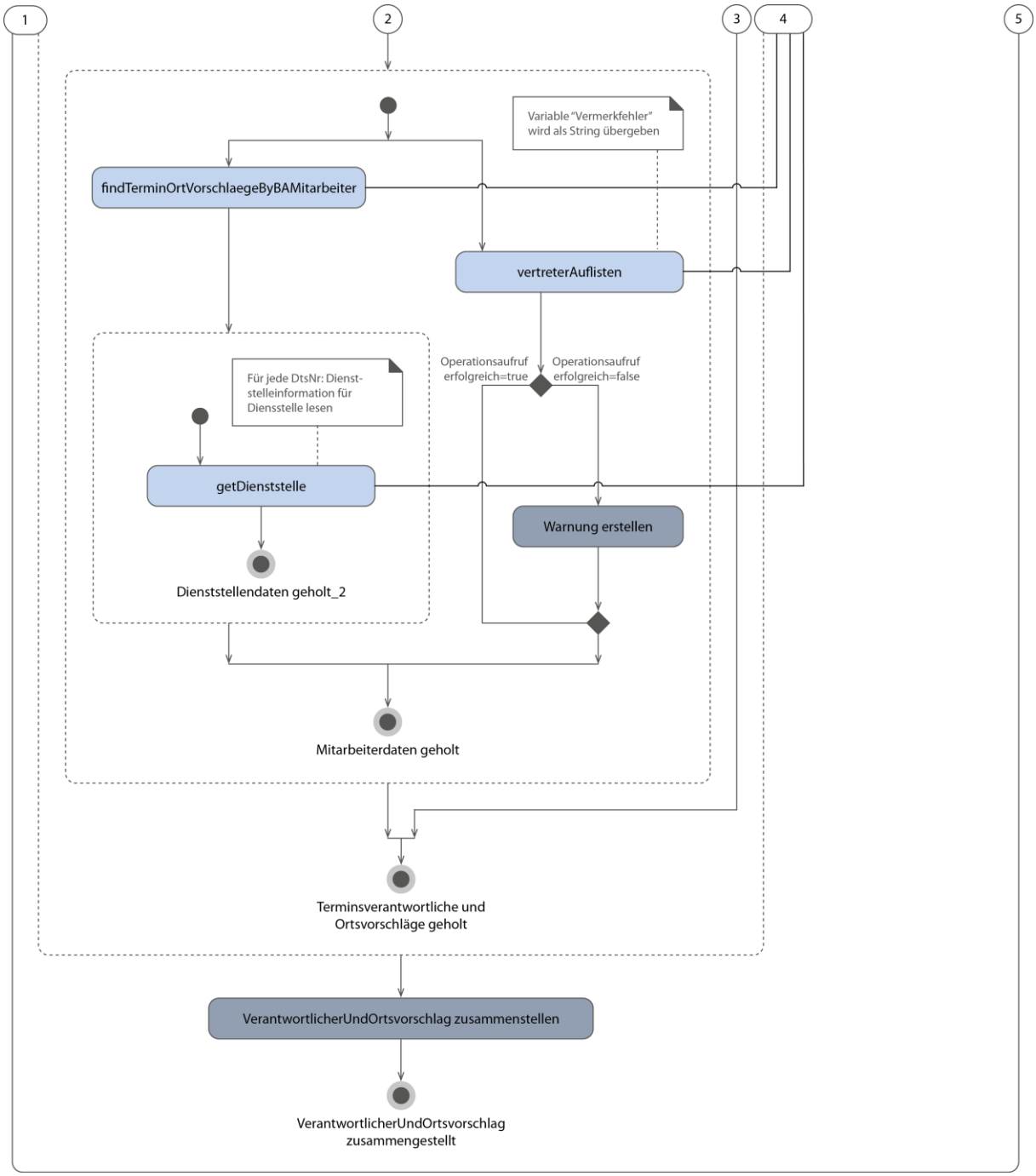


Abbildung B.5. Ablauf der Operation findTerminverantwortlicheUndOrtsVorschlaegeByKundennummerUnd-Terminart des TerminVerwaltungService mit Darstellung der vom CS verwendeten Services. Quelle: Eigene Darstellung, nach Projektdokumentation.

.

C Repository Model des Anwendungsfalls

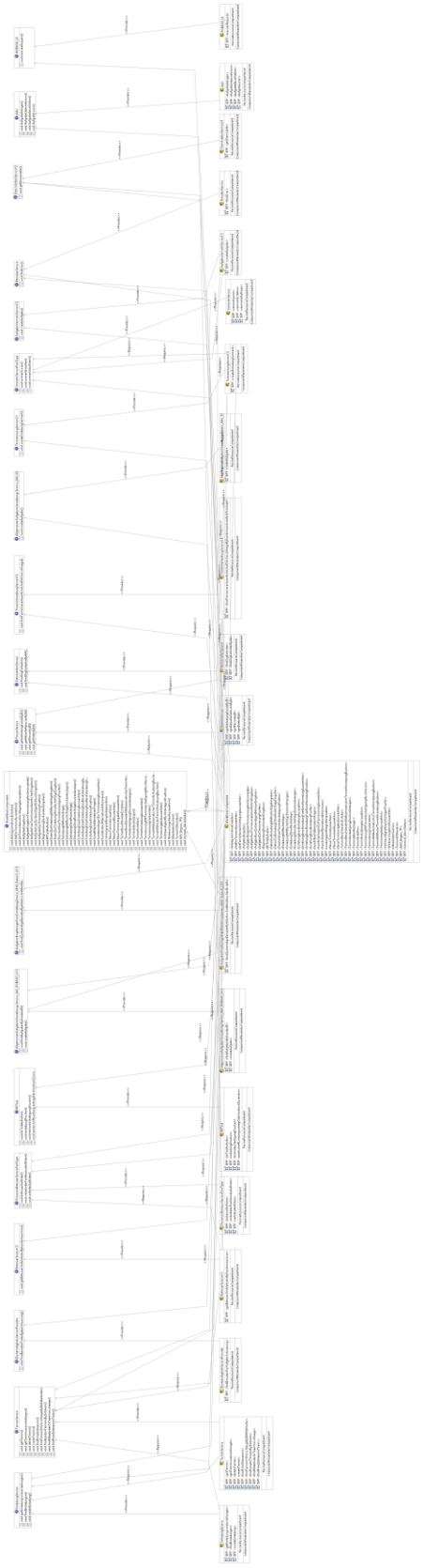


Abbildung C.1. Gesamtansicht des Repository Models des Anwendungsfalls. Quelle: Eigene Darstellung, extrahiert aus dem PCM-Editor.

D Generierte Antwortzeit-Komponentenmodelle

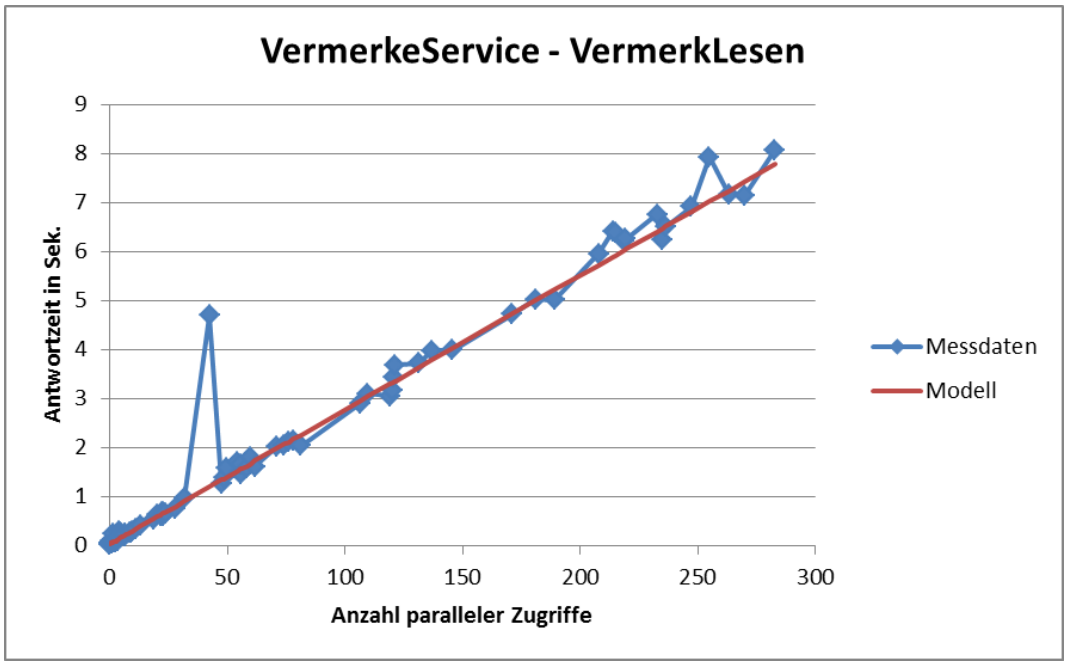


Abbildung D.1. Antwortzeit-Komponentenmodell der Service-Operation VermerkeService.VermerkLesen. Quelle: Eigene Darstellung.

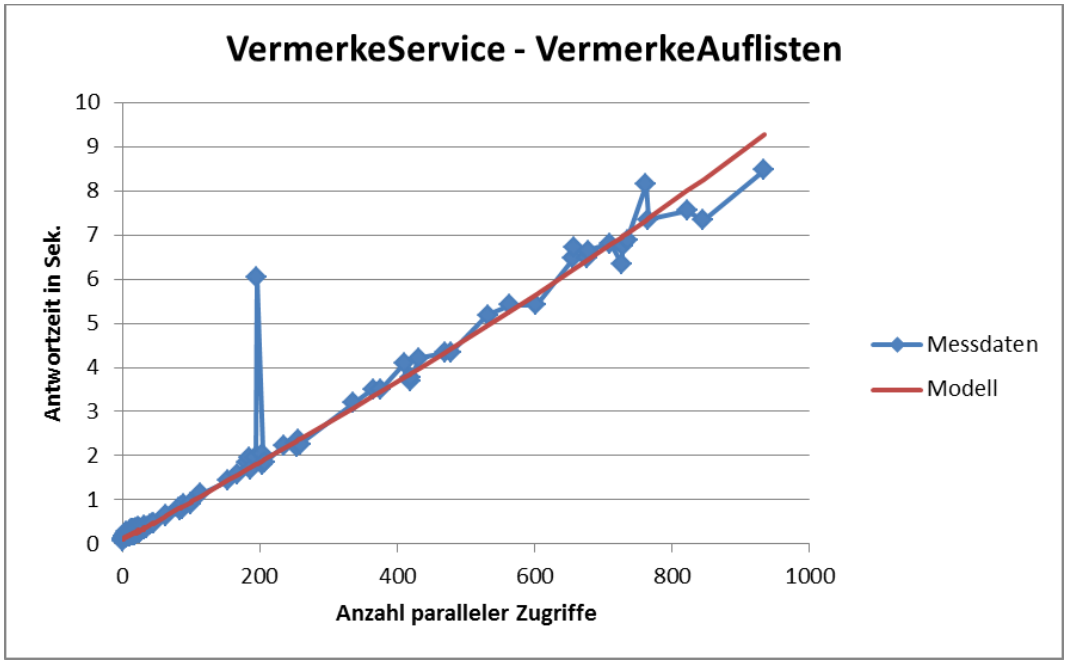


Abbildung D.2. Antwortzeit-Komponentenmodell der Service-Operationen VermerkeService.VermerkeAuflisten. Quelle: Eigene Darstellung.

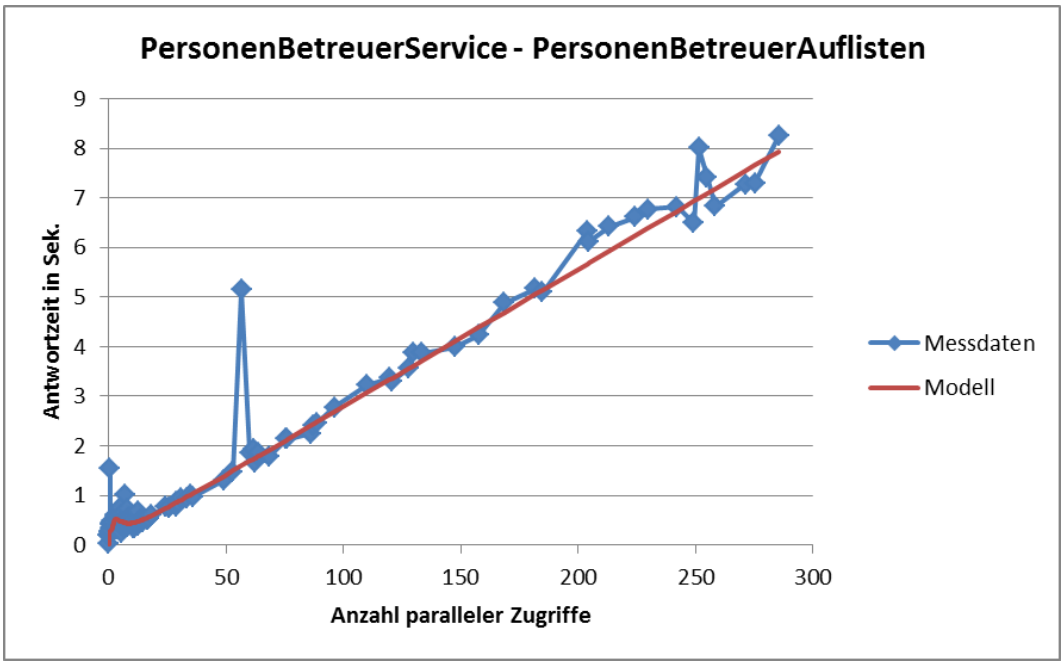


Abbildung D.3. Antwortzeit-Komponentenmodell der Service-Operation PersonenBetreuerService. PersonenBetreuerAuflisten. Quelle: Eigene Darstellung.