Optimierung der Kosten und Verfügbarkeit von IT-Dienstleistungen durch Lösung eines Redundanz-Allokation-Problems

Dissertation

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

angenommen durch die
Fakultät für Informatik der
Otto-von-Guericke-Universität Magdeburg

von
Sascha Bosse, M. Sc.
geboren am 28.03.1986 in Beckendorf-Neindorf

Gutachter:
Prof. Dr. Klaus Turowski, ITI, Otto-von-Guericke-Universität Magdeburg
Prof. Dr. Sanaz Mostaghim, IKS, Otto-von-Guericke-Universität Magdeburg
Prof. Dr. Stefan Voß, IWI, Universität Hamburg

Magdeburg, den 29. April 2016

„Soweit die Zuverlässigkeit herrscht, lässt sich alles beherrschen.“
— Lü Buwei, chinesischer Kaufmann, Politiker und Philosoph (239 v. Chr.)

Zusammenfassung

Die Verfügbarkeit von IT-Dienstleistungen ist eine der wichtigsten Qualitätseigenschaften für die Bezieher dieser Dienstleistungen, gibt sie doch an wie hoch die Wahrscheinlichkeit ist, dass ein Dienst zu einem Zeitpunkt in der Lage ist seine Funktion zu erfüllen. Um die erwartete Qualität einer IT-Dienstleistung zu beschreiben, werden Dienstgütevereinbarungen zwischen Anbieter und Kunde geschlossen. Für Erstere ist es entscheidend, die dort beschriebenen Qualitätsgarantien einzuhalten, um Vertragsstrafen oder einen langfristigen Reputationsverlust vorzubeugen.

Auf der anderen Seite führt unter anderem steigender Wettbewerbsdruck in der IT-Dienstleistungsbranche dazu, dass Dienstgütevereinbarungen kosteneffizient eingehalten werden müssen. Im Sinne der Fehlertoleranz bedeutet dies, dass die Verfügbarkeit mit den Kosten der Systemkomponenten und der Redundanzmechanismen eines IT-Dienstleistungsdesign abgewogen werden muss.

In der klassischen Zuverlässigkeitsanalyse wird dazu ein so genanntes Redundanz-Allokation-Problem definiert, um zum Beispiel ein vorgegebenes Verfügbarkeitsziel zu minimalen Kosten zu erreichen. Aufgrund der Annahme unabhängiger Komponentenausfälle bei der Definition eines solchen Problems sind diese jedoch nicht auf moderne IT-Systeme anwendbar. In der vorliegenden Arbeit wird daher ein Redundanz-Allokation-Problem für IT-Service-Design, das ITRAP, definiert. Es ermöglicht die Einbindung von Abhängigkeiten zwischen den genutzten Infrastruktur-, Hardware- und Softwarekomponenten sowie von fehlerbehafteter menschlicher Interaktion. Damit kann die Realität im Betrieb von IT-Dienstleistungen besser nachgebildet werden.

Auf Basis einer Prozedur zur automatischen Erstellung von Diskreten-Ereignis-Simulationen können Meta-Heuristiken geeignete Lösungen für das ITRAP identifizieren. Dabei können sowohl die einkriterielle Optimierung von Verfügbarkeit oder Kosten unter Nebenbedingungen als auch die mehrkriterielle, Pareto-basierte Optimierung beider Aspekte adressiert werden.

Der Einsatz des ITRAP in einem Anwendungsfall eines Application Service Provider zeigt, dass andere Designvorschläge als in einem klassischen Ansatz gemacht werden. Durch die realitätsnähere Definition wird damit die Entscheidungsunterstützung für IT-Dienstleistungsdesigner verbessert, um den Kunden von IT-Dienstleistungen hochverfügbare Dienste zu günstigen Konditionen anbieten zu können.

Abstract

One of the most important quality attributes to customers of IT services is the service availability. It is defined as the likelihood that a service is in the condition to deliver its function when required. In order to describe the expected service quality, an IT service provider and its consumers agree on service level agreements. Meeting the availability goals defined in these agreements is vital to the providers, since violations will lead to penalty costs and a loss of reputation.

On the other hand, increasing competitive pressure is one of the reasons to meet service level objectives with a minimum of cost. One way to achieve the balance between service availability and costs is the selection of suitable components and redundancy mechanisms for a fault-tolerant IT service design.

In traditional reliability engineering, a so-called redundancy allocation problem can be defined to achieve, for instance, a defined availability objective with a minimum of cost. However, due to the assumption of independent component failures, these approaches are not applicable to modern IT systems. Therefore, a redundancy allocation problem for IT service design, the ITRAP, is developed in this thesis. In its definition, dependencies between the infrastructure, hardware and software components of an IT service design can be introduced as well as defective operator interaction. Therefore, it provides are more realistic model of IT service operation.

In order to identify suitable solutions to the ITRAP, adapted meta-heuristics are utilized which base on an automatic generation procedure for discrete-event simulations. In the ITRAP, the single-objective optimization of availability or costs subject to constraints as well as the multi-objective, Pareto-based optimization of both aspects are addressed.

The application of these methods in a use-case of an application service provider demonstrates that other design suggestions are achieved than by a classical approach. Due to the more realistic definition of the ITRAP, the decision support for IT service designers can be improved. Thus, IT service providers are able to offer high-quality services to their customers for reasonable prices.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

1 Einleitung

In diesem ersten Kapitel werden zunächst die Motivation der Dissertation beleuchtet und die offenen Forschungsfragen aufgedeckt. Daraus werden die Ziele der Arbeit abgeleitet und die zur Beantwortung herangezogene wissenschaftliche Methodik vorgestellt. Außerdem wird die Struktur der vorliegenden Arbeit auf Basis dieser Methodik grob skizziert.

1.1 Motivation

Die anhaltende digitale Revolution hat die Bedeutung von effektivem und effizientem Informationsmanagement massiv erhöht. Durch sie wurde nicht nur die Unterstützung klassischer, sondern auch die Erschaffung rein IT-basierter Wertschöpfungsketten möglich. Im Zuge der Professionalisierung der IT-Leistungserbringung löst das Konzept der IT-Dienstleistung oder des IT-Service zunehmend das Konzept des reinen IT-Produktes ab [Krc15, S. 544].

Vor allem das Cloud Computing, d.h. die Bereitstellung von IT-Ressourcen als elastische Dienste über das Internet [MG11], hat diesen Trend in den letzten Jahren forciert. Während im Cloud-Infrastrukturbereich die großen IT-Konzerne wie *Amazon*, *Google* und *Microsoft* dominieren, besteht der Markt für spezialisierte Softwaredienste aus einer Vielzahl von Unternehmen. Dies führt – im Zusammenspiel mit zunehmender Standardisierung – zu steigendem Wettbewerbsdruck für Anbieter von IT-Dienstleistungen [Lü14]. Doch das Konzept des IT-Service wird nicht nur im Kontext des IT-Outsourcing angewendet. Auch die unternehmenseigene IT wird immer häufiger als interner Dienstleister betrachtet, um Qualität und Kosten der IT durch Methoden des IT-Service-Managements besser zu kontrollieren [ZB03].

Ein wichtiges Instrument dieser Kontrolle ist die vertragsähnliche Dokumention der Art und erwarteten Qualität der bezogenen IT-Dienstleistung. Dies geschieht in Vereinbarungen zwischen Dienstanbieter (Service Provider) und -konsument (Service Consumer), die je nach interner oder externer Leistungserbringung als Operational- oder Service-Level-Agreements (übersetzt etwa Betriebs- und Dienstgütevereinbarung) bezeichnet werden [Can11, S. 52][Lew99]. Neben anderen Informationen enthalten sie auch Ziele für Qualitätsmetriken, die als Service-Level-Objectives (übersetzt etwa Dienstgüteziele) bezeichnet werden. Eine Nichterfüllung dieser Ziele in einem festgelegten Zeitraum kann dabei zu Strafzahlungen führen und langfristig die Reputation des Dienstanbieters beschädigen [ENC+12, CMT+12].

Eine der wichtigsten Qualitätsmetriken für IT-Services ist dabei die Verfügbarkeit

als Maß für die Wahrscheinlichkeit, dass ein angeforderter Dienst in der Lage ist Anfragen eines Kunden zu beantworten [FJK14]. Üblicherweise wird sie als Verhältnis von tatsächlicher zu geplanter verfügbarer Zeit gemessen [Sho02, S. 14]. Natürlich ist eine hohe Verfügbarkeit im Sinne der Dienstqualität wünschenswert, jedoch ist eine höhere Verfügbarkeit in der Regel mit zusätzlichen Kosten verbunden. Erschwert wird die Planung hochverfügbarer Systeme durch den Bedarf an ständigem Wandel in der betriebenen IT-Systemlandschaft, um neuen Geschäftsanforderungen oder innovativen Technologien zu begegnen [ERS+07]. Bei der (Weiter-)Entwicklung von IT-Dienstleistungen müssen also die möglichen Qualitätsziele für die Verfügbarkeit mit den Kosten zur Erreichung dieser Ziele abgewogen werden.

Dieser Prozess sollte mit geeigneten Vorhersagemodellen für Kosten und Verfügbarkeit unterstützt werden [Hun11, S. 148f.]. Zu diesem Zweck wurden unter anderem in den letzten Jahren Verfügbarkeitsvorhersagemodelle entwickelt, die bereits in der Entwurfsphase von IT-Diensten anwendbar sind. Mithilfe dieser Modelle können die Eigenschaften eines bestimmten Entwurfes geschätzt werden. Die Optimierung eines Entwurfes, um zum Beispiel die gewünschte Verfügbarkeit mit minimalen Kosten zu erreichen, ist jedoch mit den existierenden Modellen nicht möglich.

Für diese Aufgabe kann im klassischen Zuverlässigkeitsmanagement ein so genanntes Redundanz-Allokation-Problem (RAP) definiert werden. Durch Definition möglicher Komponenten und Redundanzmechanismen können Optimierungsverfahren angewendet werden, um Systemkonfigurationen zu identifizieren, die Kosten und Verfügbarkeit optimieren. Ein RAP, welches auf geeigneten Vorhersagemodellen aufbaut, könnte für das IT-Service-Design angewendet werden. Jedoch existiert eine solche Definition bisher nicht. Dies liegt vor allem darin begründet, dass existierende RAP-Definitionen und -Lösungsmethoden von unabhängigen Komponentenausfällen ausgehen. Diese Annahme wurde jedoch für IT-Systeme als unzutreffend eingestuft [Lit06]. Die Eignung des RAP-Konzeptes für die Optimierung von IT-Service-Designs bleibt damit fragwürdig.

1.2 Ziele und Methodologie

Das Ziel dieser Arbeit ist die Verbesserung der Wirtschaftlichkeit von IT-Service-Providern durch optimiertes Service-Design im Kontext der Verfügbarkeit und Kosten. Wird die zukünftige Verfügbarkeit in der Design-Phase unterschätzt, werden Einsparpotentiale nicht genutzt. Bei Überschätzung besteht die Gefahr späterer Verletzungen der Service-Level-Agreements.

Diese Fehleinschätzungen können unter Umständen nur sehr kostspielig in der Betriebsphase korrigiert werden, zum Beispiel aufgrund der Entscheidung für einen bestimmten Anbieter [TK11]. Bisher entwickelte Vorhersageansätze bieten nur eine eingeschränkte Entscheidungsunterstützung für IT-Service-Designer durch die fehlende Optimierungskomponente. Daher sollten diese mit den Konzepten aus dem Bereich der Redundanz-Allokation-Probleme verbunden werden, um die Abwägung von Verfügbarkeit und Kosten eines IT-Service-Designs zu unterstützen. Damit wäre es möglich Kosten zu senken oder

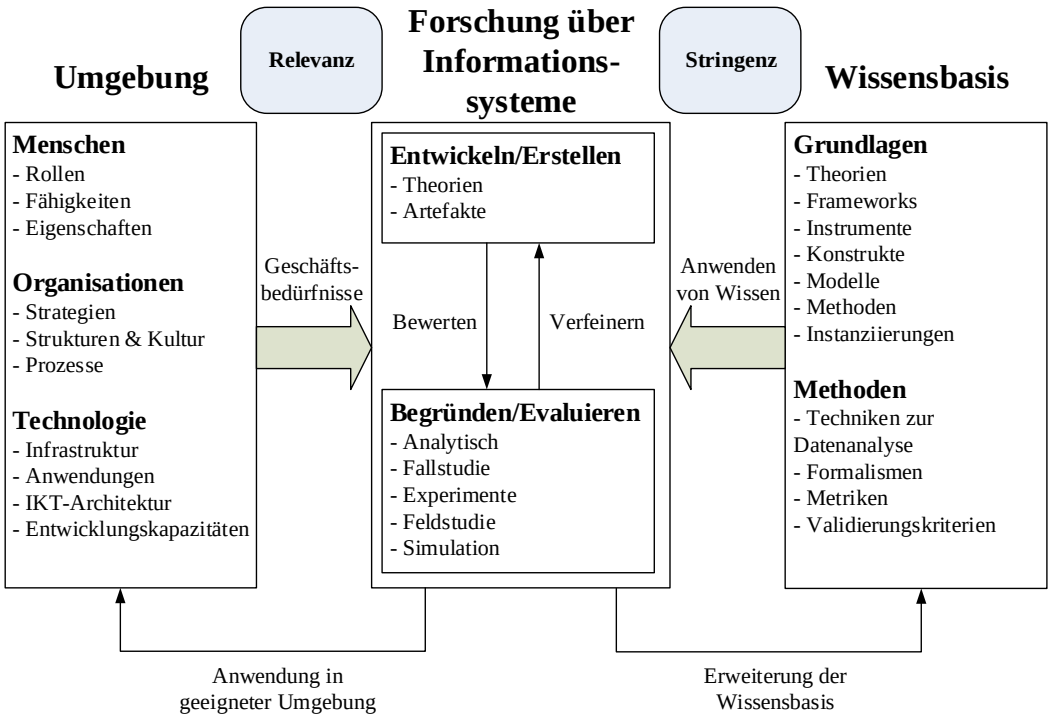

Abbildung 1.1: Framework für die Forschung über Informationssysteme (ins Deutsche übersetzt aus [HMPR04])

die Qualität zu verbessern und somit die Wirtschaftlichkeit des IT-Service-Providers zu erhöhen.

Mit diesem Ziel einher geht die bearbeitete Forschungsfrage, ob die Konzepte des RAP-Ansatzes auf IT-Service-Design übertragbar sind und wenn ja, wie diese angepasst werden müssen. Zur Beantwortung dieser Frage wird der Design-Science-Methodik für die Forschung über Informationssysteme gefolgt (vgl. [HMPR04, PTRC08]). Dies ist ein ingenieurwissenschaftlicher, konstruktivistischer Ansatz, der durch die Schaffung relevanter Artefakte real-weltliche Probleme wissenschaftlich adressiert. In Abbildung 1.1 kann die prinzipielle Struktur dieser Forschungsmethode nachvollzogen werden. Aus der Umgebung wird ein Problem identifiziert, welches durch das Artefakt adressiert wird. Die Lösung eines Problems aus der Umgebung belegt die Relevanz der Forschung. In der vorliegenden Arbeit ist das Problem mangelnde Entscheidungsunterstützung von IT-Service-Designern, die zu nicht genutztem wirtschaftlichen Potenzial führt. Ein RAP für IT-Service-Design, in der Folge als ITRAP bezeichnet, würde als Artefakt dieses Problem adressieren.

Um festzustellen, ob das Artefakt das Problem auch hinreichend adressiert, muss das entwickelte Artefakt evaluiert werden. Auf Basis der Evaluation kann das Artefakt verbessert werden. Dieser Zyklus kann so oft wiederholt werden, bis eine gewünschte Qualität erreicht wird. In dieser Dissertationsschrift wird die Evaluierung des ITRAP einerseits durch einen verifizierten Prototypen als Proof-of-Concept, andererseits durch Demonstration in einem realen Anwendungsfall durchgeführt.

Das Artefakt wird dabei nach stringenten Methoden und mit vorhandenen Informationen aus der Wissensbasis entwickelt, zum Beispiel aus dem Bereich des Software-Engineering und der Redundanz-Allokation-Probleme. Die beschriebene Forschung schafft auch neue Inhalte für die Wissensbasis durch die Beantwortung der Forschungsfrage, ob das RAP-Konzept für IT-Services angewendet werden kann und inwieweit sich die für IT-Services entwickelten Verfügbarkeitsvorhersagemethoden in ein Optimierungsframework integrieren lassen.

1.3 Eigene Veröffentlichungen

Im Rahmen dieses Dissertationsvorhabens sind eigene Veröffentlichungen entstanden, die Schnittmengen mit dieser Arbeit besitzen. Sie werden im Folgenden chronologisch beschrieben.

In [BST13] wird ein Petri-Netz-Ansatz zur Verfügbarkeitsvorhersage von IT-Services vorgestellt, der die Modellierung von Abhängigkeiten zwischen den Servicekomponenten ermöglicht. Dieser Ansatz wird in [Bos13] um die Möglichkeit erweitert, fehlerbehaftete Administratorinteraktion einzuführen. Die in dieser Arbeit zur Designevaluation genutzte Simulationsmethode baut auf diesen beiden Veröffentlichungen auf.

Für die Integration des Vorhersageansatzes in ein Optimierungsframework wird eine automatische Generierungsmethode für die Petri-Netz-Simulation genutzt. Die Grundlagen dieser Methode werden in [BT13] vorgestellt. Der Stand der Technik für Verfügbarkeitsvorhersageansätze im IT-Service-Bereich wurde analytisch in [Bos14] vorgestellt und in [BST14b] durch eine strukturierte Literaturanalyse quantitativ beschrieben.

Die Abhängigkeit der Verfügbarkeit von der Performance eines IT-Service wird in [BST14a] diskutiert und in ein Vorhersagemodell integriert. Dieses wird in [BHS+15] um den Kostenaspekt erweitert. Teilkonzepte dieses Modells werden auch in dieser Arbeit verwendet. In [SBST15] wird ein Evaluationsframework für Load-Balancing-Algorithmen in Cloud-Rechenzentren beschrieben. Dabei wird der Energieverbrauch von Komponenten abhängig von der Auslastung modelliert. Teile dieses Konzepts werden auch in dem RAP für IT-Services verwendet.

Das ITRAP wird im Folgenden drei Optimierungsprobleme adressieren. Eines davon, die Minimierung der IT-Service-Kosten bei gleichzeitiger Einhaltung einer Verfügbarkeitsgrenze (Optimierungsproblem (2) in dieser Arbeit) durch ein RAP für IT-Services ist in [BST15] beschrieben. Die mehrkriterielle Optimierung von Verfügbarkeit und Kosten (Optimierungsproblem (3)) wird in [BST16] präsentiert. Dabei wird ausführlich auf die Evaluation einzelner Lösungskandidaten eingegangen und die Anpassung eines Genetischen Algorithmus und einer Tabu-Suche als Optimierungsmethoden beschrieben.

1.4 Sprachliche Anmerkungen

Das behandelte Forschungsgebiet, das sich im Bereich des IT-Service-Managements befindet, ist von englischen Fachbegriffen durchsetzt. Im weiteren Verlauf der Arbeit werden

daher oft englischsprachige synonym zu deutschen Fachbegriffen benutzt, zum Beispiel IT-Dienstleistung und IT-Service. Für manche Konzepte wie Service-Level-Agreements wird nur der englische Fachterm benutzt, da eine Übersetzung auch in der deutschsprachigen Fachliteratur unüblich ist.

1.5 Struktur der Arbeit

Neben diesem Kapitel, in dem das Thema, die Ziele und die Methodologie der Arbeit vorgestellt werden, enthält die Arbeit vier weitere Kapitel. Im zweiten Kapitel wird der aktuelle Stand der Technik, einerseits für die Vorhersage der Verfügbarkeit und Kosten von IT-Services, andererseits für das Redundanz-Allokation-Problem, beschrieben.

Das dritte Kapitel „Ein Redundanz-Allokation-Problem für IT-Dienstleistungen“ verfolgt die Entwicklung des ITRAP-Artefakts von der Anwendungsanalyse über die ITRAP-Definition, die Beschreibung einer möglichen Evaluationsmethode bis hin zur Vorstellung geeigneter Lösungsverfahren. Auf Basis dieses Konzepts wird in Kapitel 4 die Evaluierung des Artefakts vorgestellt. Dazu wird der entwickelte Prototyp sowie dessen Verifikation illustriert. Im weiteren Verlauf des Kapitels wird zunächst der untersuchte Anwendungsfall vorgestellt, bevor die durchgeführten Experimente präsentiert und ausgewertet werden.

Die Arbeit endet mit dem Schlusskapitel, in dem zunächst die Dissertation mit deren wichtigsten Ergebnissen zusammengefasst wird. Darauf aufbauend werden weitere Forschungspotenziale, die aus dieser Arbeit erwachsen, im Ausblick vorgestellt. Außerdem werden die gewonnenen Erkenntnisse kritisch beleuchtet, um das Erreichen der Ziele final zu beurteilen und die Dissertation in einen größeren Kontext einzuordnen.

2 Stand der Technik

Dieses Kapitel widmet sich den relevanten Grundlagen für das Verständnis der Arbeit. Dabei wird auch die bisher erfolgte Forschung in den Bereichen der Verfügbarkeitsvorhersage für IT-Service-Designs und des Redundanz-Allokation-Problems beleuchtet. Dies soll im Besonderen die Forschungslücke aufzeigen, die in dieser Dissertation zu schließen versucht wird. Die Ergebnisse der Analyse zum Stand der Technik bilden die Grundlage für das in der Arbeit entwickelte Artefakt, ein Redundanz-Allokation-Problem für IT-Service-Design.

2.1 Vorhersage der Verfügbarkeit und Kosten von IT-Dienstleistungen

Im ersten Teil des zweiten Kapitels wird zunächst eine allgemeine Einführung in das Gebiet der Verfügbarkeit und Kosten von IT-Dienstleistungen geboten. Dazu werden die Konzepte der IT-Dienstleistung und des IT-Service-Managements kurz erläutert, bevor die Brücke zum Availability-Management geschlagen wird. In diesem Teilprozess des IT-Service-Managements werden die existierenden Ansätze zur Vorhersage der Verfügbarkeit von IT-Services eingeordnet und beispielhaft erläutert.

2.1.1 IT-Dienstleistungen

Seit dem Beginn der Computerisierung in der zweiten Hälfte des 20. Jahrhunderts ist Informationstechnologie (IT) aus dem täglichen Leben nicht mehr wegzudenken. Die effektive und effiziente Nutzung von IT ist inzwischen zu einem Hygienefaktor für erfolgreiche Unternehmen geworden. Diese Entwicklung verlief jedoch nicht geradlinig: Hohe IT-Investitionen führten spätestens nach dem Platzen der „Dotcom-Blase“ 2000 zu einer Diskussion, inwieweit IT-Investitionen zu einem anhaltenden strategischen Vorteil führen könnten.

Eine besonders beachteter Meinungsbeitrag stammt von Nicholas Carr, der in seinem Artikel „IT doesn’t matter“ die Nachhaltigkeit der Suche nach strategischen Vorteilen durch IT-Investitionen in Frage stellte [Car03]. Über zehn Jahre nach diesem Beitrag scheint diese Meinung sich nicht bewahrheitet zu haben. Der anhaltende Erfolg von IT-Unternehmen wie Amazon, Apple, Facebook, Google oder Microsoft zeigt dies genauso wie aktuelle IT-Trends, zum Beispiel *Machine Learning*, *Internet of Things* oder autonome Kraftfahrzeuge, die in der öffentlichen Wahrnehmung stark diskutiert werden [RvdM15].

Ein Grund für diese Entwicklung ist der Wandel der IT-Organisationen in Unterneh-

men. War es früher üblich, eine IT-Abteilung neben anderen Abteilungen eines Unternehmens zu verankern, wird die unterstützende IT heute häufiger von internen oder externen Dienstleistern betrieben. Dieser Wandel wurde zunächst vom IT-Outsourcing getrieben, von dem sich Unternehmen eine Senkung der (Fix-)Kosten sowie das Verschieben von Risiken versprechen [SH02, S. 464]. Dies sind traditionell Ziele, die durch den Bezug von Dienstleistungen statt Produkten erreicht werden können [Can11, S. 13].

Da mit dem Outsourcing der IT jedoch auch ein Kontrollverlust erfolgt [SH02, S. 466f.], ist es für den Dienstleistungsbezieher essentiell, eine ausreichende Qualität des Dienstes (QoS von engl. *Quality of Service*) zu verlangen und sicherzustellen. In Großbritannien beschäftigte sich die Regierungsbehörde „Central Computer and Telecommunications Agency“ (später „Office of Government Commerce“, heute „Cabinet Office“) zwischen 1989 und 1998 mit der Dokumentation von Ansätzen für den kosteneffektiven und effizienten Gebrauch von IT-Ressourcen für öffentliche Organisationen [ITSMF05, S. 9f.]. Diese Sammlung wird als „IT Infrastructure Library“ (ITIL) bezeichnet und später in den aktualisierten Versionen 2 (2003), 3 (2007) und 2011 (ebendann) veröffentlicht.

Die ITIL stellte eine der ersten umfassenden Beschreibungen eines IT-Service-Managements dar. IT-Organisationen werden als unternehmensinterne oder -externe Dienstleister beschrieben. Studien wie in [HTB05] zeigen, dass die Adaption von ITIL in Unternehmen zu einer gesteigerten Kundenorientierung sowie Effizienz und Transparenz der IT-Prozesse führt, die die Einführungskosten rechtfertigen. Auch die Adaptionen der ITIL-Inhalte durch IT-Unternehmen wie Hewlett-Packard (*HP ITSM Reference Model*), IBM (*IT Process Model*) oder Microsoft (*Microsoft Operations Framework*) unterstreichen nicht nur, dass die ITIL ein Standardwerk im Bereich des IT-Service-Managements darstellt, sondern auch wie weit verbreitet das Konzept der Dienstleistungsorientierung der IT heute ist.

Der Begriff des IT-Service wird in der ITIL 2011 wie folgt beschrieben:

> “A service provided by an IT service provider [...] made up of a combination of information technology, people and processes.” [Can11, S. 13]
>
> Übersetzt etwa: „Eine Dienstleistung, die von einem IT-Dienstleister erbracht wird und aus einer Kombination von Informationstechnologie, Menschen und Prozessen erstellt wird.“

Diese Definition konzentriert sich damit auf die gesamte Leistungserstellung, die natürlich mittels der IT erbracht wird, aber auch Menschen und Prozesse involviert. Dabei ist ein IT-Service immer direkt oder indirekt erfolgswirksam, unterstützt also Geschäftsprozesse, um die Geschäftsziele des Servicekonsumenten zu erreichen. In diesem Sinne werden direkt an den Kunden adressierte und unterstützende IT-Dienstleistungen unterschieden. Eine IT-Dienstleistung kann dabei für das eigene und für andere Unternehmen erbracht werden.

Die zu erreichende Servicequalität sollte dabei in Service-Level-Agreements (SLA) für

externe Kunden und in Operational-Level-Agreements (OLA) für die interne Leistungserstellung dokumentiert werden. Diese enthalten u.a. Rechte und Pflichten der Service-Partner [Lew99]. Verletzungen dieser Vereinbarungen, insbesondere der definierten Qualitätsziele, der Service-Level-Objectives (SLO), können zu Strafzahlungen, Opportunitätskosten und dem Verlust von Reputation führen [ENC+12, CMT+12]. Im Laufe dieser Arbeit wird der Begriff SLA synonym zu OLA benutzt.

Sicherzustellen, dass angebotene IT-Services die SLO einhalten und die Geschäftsprozesse der Konsumenten effektiv unterstützen, ist eine Aufgabe des IT-Service-Managements.

2.1.2 IT-Service-Management und Availability-Management

Der Begriff des IT-Service-Managements (ITSM) wird in der ITIL 2011 wie folgt definiert:

> "The implementation and management of quality IT services that meet the needs of the business [...] through an appropiate mix of people, processes and information technology" [Can11, S. 16]
>
> Zu deutsch etwa: „Die Umsetzung und Steuerung hochwertiger IT-Dienstleistungen, welche die Geschäftsbedürfnisse durch eine geeignete Mischung aus Menschen, Prozessen und Informationstechnologie erfüllen"

Um dieser Aufgabe gerecht zu werden, definiert die ITIL zahlreiche Prozesse, die den fünf Phasen des Lebenszyklus von IT-Services zugeordnet sind. Diese sind nach [Can11, S. 5ff.] Service Strategy, Service Design, Service Transition, Service Operation und Continual Service Improvement. Die einzelnen Prozesse der fünf Phasen sind in Tabelle 2.1 dargestellt.

Service Strategy ist die Phase, in der die strategische Ausrichtung des IT-Service-Providers sowie seine Beziehung zum Geschäft der Konsumenten festgelegt wird. Alle späteren Aktivitäten des ITSM sollten an der definierten Strategie ausgerichtet und bewertet werden. Dazu werden in dieser Phase Themen wie die Entwicklung eines Service-Portfolios, die finanziellen Rahmenbedingungen oder die Maximierung des Wertbeitrages behandelt.

Im Service Design werden Bedingungen geschaffen, um konkrete Dienste in hoher Qualität anbieten zu können, u.a. in Fragen der Verfügbarkeit. Dabei werden die in der Service Strategy formulierten Ziele verfolgt. Die meisten Prozesse dieser Phase haben direkten Bezug zur Erfüllung der SLA, zum Beispiel das Capacity-, das IT-Service-Continuity- oder das Information-Security-Management. Auch das Availability-Management ist ein Teilprozess des Service Designs. Es ist in dieser Phase besonders wichtig, korrekte Entscheidungen zu treffen, da Änderungen in späteren Lebenszyklusphasen in der Regel mit hohen Kosten verbunden sind [TK11].

Die Phase der Service Transition beinhaltet Prozesse, die neu oder weiter entwickelte Services in die Systemlandschaft des IT-Service-Providers einführen. Dabei soll die

Tabelle 2.1: Prozesse der IT Infrastructure Library in den vier Lebenszyklusphasen von IT-Services

Lebenszyklusphase	Prozesse
Service Strategy	Strategy Management for IT services Service Portfolio Management Financial Management for IT services Demand Management Business Relation Management
Service Design	Design Coordination Service Catalogue Management Service Level Management Availability Management Capacity Management IT Service Continuity Management Information Security Management Supplier Management
Service Transition	Transition Planning and Support Change Management Service Asset and Configuration Management Release and Deployment Management Service Validation and Testing Change Evaluation Knowledge Management
Service Operation	Event Management Incident Management Request Fulfilment Problem Management Access Management
Continual Service Improvement	Seven-Step Improvement Process

Einführung neuer Services möglichst risikofrei vonstatten gehen und Wissen darüber abgerufen und dokumentiert werden. Daher finden in dieser Phase Prozesse wie das Change-Management oder die Service-Validierung statt.

Service Operation bezeichnet die Lebenszyklusphase des Betriebs einer IT-Dienstleistung. In dieser Phase wird die Strategie des IT-Service-Providers im Idealfall konkret umgesetzt. Dazu soll vor allem die Störungsfreiheit des Services und die Kundenzufriedenheit garantiert werden. Dies wird durch Prozesse wie Event-, Incident- und Problem-Management erreicht.

Die Phase der Continual Service Improvement begleitet einen Service über seinen gesamten Lebenszyklus von der Definition einer Strategie bis zum Betrieb. Hier soll sichergestellt werden, dass Verbesserungspotenzial aufgedeckt und dokumentiert wird. Dabei werden beispielsweise Prinzipien des Qualitätsmanagement genutzt.

Das Availability-Management (AM) ist dabei der Prozess, in dem maßgeblich die künftige Verfügbarkeit eines IT-Service gesteuert wird [Hun11, S. 125ff.]. Im Sinne der ITIL ist Verfügbarkeit dabei die Fähigkeit einer Dienstleistung, die gewünschte Funktion bereitzustellen, wenn sie benötigt wird [Hun11, S. 128]. Neben dem AM haben auch andere Prozesse Berührungspunkte zu Fragen der Verfügbarkeit [Hun11, S. 154f.].

Zum Beispiel wird im Service-Level-Management die Festlegung und die Prüfung der Verfügbarkeits-SLO durch das AM unterstützt. Im Incident- bzw. Problem-Management werden Service-Unterbrechungen mittels der im AM definierten Maßnahmen adressiert. Das Capacity-Management und das AM sind voneinander abhängig, da mangelnde Ressourcen zu Verfügbarkeitsproblemen führen können. Auch das Change-Management muss sich mit Fragen der Verfügbarkeit auseinander setzen. So muss jede Änderung am Service auf ihren Effekt bezüglich der Verfügbarkeit untersucht werden. Neben dem AM ist auch das Continuity-Management zentriert auf Verfügbarkeit. Während das AM jedoch auf (erwartbare) Probleme im Betrieb fokussiert ist, beschäftigt sich das Continuity-Management mit der Wiederherstellung des Betriebs nach einer außerordentlichen Unterbrechung des Services, z.B. durch eine Katastrophe.

Neben der ITIL als inzwischen anerkanntes Standardwerk im ITSM existieren eine Vielzahl anderer oder angepasster Sammlungen von Prozessen für das IT-Service-Management. Auch dort sind Prozesse zur Steuerung der Verfügbarkeit vorgesehen, so zum Beispiel das „Managen von Verfügbarkeit und Kapazität“ im Komplex „Aufbauen, Beschaffen und Implementieren“ in COBIT 5.0 [Inf12] oder das Service-Continuity-and-Availability-Management als Service-Delivery-Process in der ISO 20000 [ISO11].

Somit kann festgestellt werden, dass das Management der Verfügbarkeit von IT-Dienstleistungen einen essentiellen Prozess des ITSM darstellt und – über das Service-Level-Management – mitentscheidend für den Erfolg eines IT-Service-Providers ist.

Metriken und Maßnahmen im Availability-Management Wie bereits oben erwähnt, wird die Verfügbarkeit eines IT-Service in der ITIL wie folgt beschrieben:

> “the ability of a service [...] to perform its agreed function when required”
>
> Übersetzt etwa: „die Fähigkeit einer Dienstleistung, ihre vereinbarte Funktion zu erfüllen wenn diese angefragt wird.“

Als Metrik für die Verfügbarkeit wird dabei meist das Verhältnis der Zeit, in der der Service seine vereinbarte Funktion erfüllen konnte, zu der Zeit, in der dies geplant war (also zum Beispiel nicht im Wartungsfenster o.ä.), angegeben [Hun11, S. 128].

Die Ausfallzeit (TTF von engl. *time to failure*) ist als kontinuierliche Zufallsvariable X definiert, die oft als exponentialverteilt angenommen wird. $F(x) = P(X \leq x)$ sei die Verteilungsfunktion dieser Zufallsvariable. Die mittlere Ausfallzeit (MTTF) ist dann der Erwartungswert von X [Mil10].

Die Zuverlässigkeit (engl. *reliability*) ist nur von X abhängig und bezeichnet die Eigenschaft eines Systems, die geforderte Funktion in einer bestimmten Zeitspanne zu erfüllen [ITU08]. Sie ist als die bedingte Wahrscheinlichkeit $R(t, t + \Delta t) = P(X > t + \Delta t | X > t)$ definiert, dass das System bis zum Zeitpunkt $t + \Delta t$ in Betrieb ist, wenn es zur Zeit t verfügbar war. Für $t = 0$ und $\Delta t > 0$ gilt dann $R(0, \Delta t) = P(X > \Delta t | X > 0)$ und wenn das System initial nicht ausgefallen ist ($P(X > 0) = 1$), gilt $R(0, \Delta t) = R(\Delta t) = P(X > \Delta t) = 1 - F(\Delta t)$ [GSFT08]. Damit bezeichnet $R(t)$ die Wahrscheinlichkeit, dass ein System bis zum Zeitpunkt t nicht ausgefallen ist.

Im Gegensatz zur Zuverlässigkeit betrachtet die Verfügbarkeit (engl. *availablity*) neben der Ausfallzeit auch die Wiederherstellungszeit (TTR von engl. *time to recovery*) eines Systems. Sie kann durch eine Zufallsvariable Y mit Erwartungswert $MTTR$ beschrieben werden. Ein reparables System wechselt also den Zustand von verfügbar nach ausgefallen nach einer zufälligen Zeit X. Der umgekehrte Zustandsübergang findet daraufhin nach einer zufälligen Zeit Y statt. Damit kann die mittlere Zeit zwischen den Ausfällen (MTBF von engl. *mean time between failures*) als $E(X) + E(Y) = MTTR + MTTF$ beschrieben werden.

Ein stochastischer Prozess Z gibt an, ob sich ein System zu einem Zeitpunkt entweder im verfügbaren (Wert 1) oder im nicht verfügbaren Zustand (Wert 0) befindet. Für die Verfügbarkeit können dann folgende Metriken verwendet werden [Mil10]:

Die unmittelbare Verfügbarkeit (engl. *instantaneous availability*) ist die Wahrscheinlichkeit, dass das System zu einem beliebigen Zeitpunkt verfügbar ist:

$$A(t) = P(Z(t) = 1) = E(Z(t))$$

Die Intervall-Verfügbarkeit (engl. *interval availability*) ist die Wahrscheinlichkeit, dass ein System über einen Zeitraum $[t_0, t]$ verfügbar ist:

$$A(t_0, t) = \frac{1}{t - t_0} \int_{t_0}^{t} Z(\tau) \mathrm{d}\tau$$

Die stabile Verfügbarkeit (engl. *steady-state availability*) ist der Anteil der verfügbaren Zeit über die gesamte Lebenszeit. Damit ist die in der ITIL beschriebene Verfügbarkeitsmetrik analog zur stabilen Verfügbarkeit [STP12, S. 33f.]:

$$A = \lim_{t \to \infty} A(t) = \frac{MTTF}{MTBF}$$

Da es die Hauptaufgabe des AM ist eine Verfügbarkeit zu garantieren, die die Umsetzung der Service-Strategie möglich macht, sollten Service-Unterbrechungen minimiert werden. Diese Unterbrechungen werden als Ausfall (engl. *failure*) bezeichnet. Diese können von einer Abweichung (engl. *error*) verursacht werden, die meist einen Fehler (engl. *fault*) als Grund hat [Lap95].

Um die Ausfallzeit zu minimieren, gibt es vier prinzipielle Ansätze [Lap95]. Die **Fehlervermeidung** (engl. *fault prevention*) bezeichnet Aktivitäten zur Verhinderung von Fehlern, z.B. die Einführung hochwertiger Software oder die geeignete Ausbildung von

Mitarbeitern, während die **Fehlerentfernung** (engl. *fault removal*) Maßnahmen subsumiert, die Anzahl und Wirksamkeit von Fehlern direkt zu verringern, z.B. durch Wartungsarbeiten. **Fehlervorhersage** (engl. *fault forecasting*) bezeichnet Möglichkeiten, das Auftreten von Fehlern zu einem Zeitpunkt vorherzusagen, sodass operative Gegenmaßnahmen eingeleitet werden können. Dies wird in der Regel durch die Kombination von Monitoring und Datenanalyse erreicht.

Fehlertoleranz (engl. *fault tolerance*) bedeutet hingegen, die Funktion des Systems auch unter der Präsenz von Fehlern aufrechtzuerhalten. Da Fehler nie komplett ausgeschlossen werden können [LA90, S. 4f.], ist Fehlertoleranz ein effektiver Ansatz um Hochverfügbarkeit zu erreichen [Lap95]. Dabei wird in der Regel versucht, den Fehler einer Systemkomponente durch eine Menge funktional äquivalenter und in einer Weise redundant betriebener Komponenten abzudecken [Sho02, S. xx]. Dies kann jedoch durch unzureichende Abdeckung (engl. *imperfect coverage*) weniger effektiv sein als erwartet [Lap95].

Ist eine Komponente dabei in ständiger Betriebsbereitschaft, wird von aktiver Redundanz gesprochen. Alternativ kann man die redundante Komponente in einem Standby-Zustand halten und erst bei einem Fehler der Primärkomponente aktivieren. Je nach Betriebsbereitschaft der Standby-Komponente kann heiße (hot), warme (warm) und kalte (cold standby) Standby-Redundanz unterschieden werden [AH14, CRNK12]. Mit sinkender Betriebsbereitschaft erhöht sich die Zeit von dem Ausfall der Primärkomponente bis zur Betriebsübernahme der Standby-Komponente, aber gleichzeitig können Fehlerrate [Sho02, S. 21] und Energieverbrauch gesenkt werden [ODAL14].

Neben der Unterscheidung von aktiver und passiver oder Standby-Redundanz können auch homogene und heterogene Redundanz unterschieden werden. Bei ersterer wird nur ein Komponententyp verwendet, während bei der heterogenen oder diversitären Redundanz verschiedene Komponententypen verwendet werden. Damit kann die Wahrscheinlichkeit eines Ausfalles aufgrund gemeinsamer Ursachen (engl. *common cause failure*) verringert werden.

2.1.3 Vorhersagemethoden für die Verfügbarkeit von IT-Services

Eine Möglichkeit, dass AM in der Design-Phase zu unterstützen, ist die Vorhersage der künftigen Verfügbarkeit eines konkreten Entwurfes. Ist die geschätzte Verfügbarkeit unter dem angestrebten SLO-Wert, können weitere Maßnahmen zur Ausfallminimierung ergriffen werden. Eine durchgeführte strukturierte Literaturanalyse, detailliert beschrieben in [BST14b], bildet die Basis für die folgende Analyse von Vorhersageansätzen für die IT-Service-Verfügbarkeit.

Die Ansätze zur Vorhersage der Verfügbarkeit können grob in qualitative und quantitative Ansätze unterteilt werden. Qualitative Ansätze ermöglichen die vergleichende Betrachtung mehrerer möglicher Entwürfe. Auf diesem Ansatz basierende Verfahren wie z.B. Experteninterviews sind in der Regel leicht und schnell durchführbar und deren Ergebnisse können ohne größeren Interpretationsbedarf als Entscheidungsgrundlage dienen. Auf der anderen Seite lassen sich die Ergebnisse nur sehr eingeschränkt auf neue Anwendungsfälle

übertragen. Des Weiteren hängt die Güte der Ergebnisse – zumindest bei subjektiven Verfahren – stark vom Fachwissen der involvierten Personen ab [MM11]. Im Kontext des AM sind diese Verfahren daher wenig geeignet, um die Verfügbarkeit verschiedener Entwürfe präzise abzuwägen.

Dies ermöglichen quantitative Ansätze, da diese Zahlen als Ergebnis produzieren. Deren Ergebnisse können somit in einem Entscheidungsunterstützungswerkzeug computergestützt weiterverarbeitet werden. Dabei können Black-Box- und White-Box-Verfahren unterschieden werden, abhängig von dem Wissen über die Abhängigkeiten der Zielgrößen von den Eingangsgrößen.

Black-Box-Verfahren können ohne Kenntnis über diese Abhängigkeiten angewendet werden. Im Kontext des AM wird mit Methoden des Data-Mining versucht, aus gesammelten (Mess-)Daten die künftige Verfügbarkeit abzuleiten. Dafür können beispielsweise Neuronale Netze und andere Regressionsverfahren eingesetzt werden, die die Verfügbarkeit aus Monitoring-Daten vorhersagen (z.B. in [HSM04, SDKS13, MT06]). Je nach Qualität und Quantität der Input-Daten können diese Verfahren sehr präzise sein, bei Änderungen an der Systemlandschaft kann jedoch erst nach erneuter Datenerfassung eine akzeptable Vorhersagequalität erreicht werden [IN08]. Besonders bei einer umfangreichen Neu- oder Weiterentwicklung eines Services mit mehreren Design-Alternativen sind die benötigten Daten in der Regel nicht verfügbar, sodass diese Verfahren frühestens im Rahmen der Service Transition angewendet werden können. In diesem Fall müssen zusätzliche Informationen über das Design genutzt werden.

Die Verwendung solcher Informationen führt zu den White-Box-Verfahren, die auch als analytische Methoden bezeichnet werden. Nach dem Prinzip „divide et impera“ wird das zu untersuchende System in seine Komponenten und deren Beziehungen zerlegt. Diese Zerlegung erfolgt so detailliert, bis für die identifizierten Komponenten belastbare Verfügbarkeitsdaten vorliegen. So werden einerseits die Wiederverwendung und Übertragung von Daten der Komponenten ermöglicht sowie andererseits fehlende Daten durch Wissen ausgeglichen. Daher sind diese Verfahren bereits in der Design-Phase anwendbar.

White-Box-Ansätze können nach der Art des unterliegenden Modells klassifiziert werden. So kann man kombinatorische, zustandsraumbasierte und hierarchische Verfahren unterscheiden [TCD+08]. In kombinatorischen Modellen wird die Systemverfügbarkeit mithilfe der Wahrscheinlichkeitstheorie aus den Komponentenverfügbarkeiten berechnet. Dabei nehmen diese Verfahren unabhängige Komponentenfehler an, sodass keine bedingten Wahrscheinlichkeiten berechnet werden müssen. Daher sind diese Verfahren in der Regel einfach und schnell anzuwenden. Beispiele sind Reliability Block Diagrams und Fehlerbäume.

Mit dem Fokus auf der Vorhersage der Verfügbarkeit von IT-Services muss jedoch erwähnt werden, dass die Annahme unabhängiger Komponentenfehler die Abbildung der komplexen Beziehungen eines modernen IT-Systems erschwert bzw. unmöglich macht [CMT+12, TCD+08]. Da die Genauigkeit der erhaltenden Ergebnisse damit stark beeinträchtigt ist, sind diese als Entscheidungsgrundlage kaum geeignet [Lit06].

Ein Beispiel für solche Abhängigkeiten ist die bereits oben erwähnte unzureichende

Abdeckung, die unter bestimmten Umständen dazu führt, dass eine redundant betriebene Komponente den Ausfall einer Primärkomponente nicht voll kompensieren kann [MM11]. Auch Ausfälle aufgrund gemeinsamer Ursachen stellen die Annahme unabhängiger Fehler in Frage. Dabei führen bestimmte Situationen dazu, dass mehrere Komponenten gleichzeitig ausfallen [Sho02, S. 99f.]. Dies kann insbesondere bei Software-Systemen aufgrund systematischer Bugs auftreten [CK90].

Zusätzlich zu gemeinsamen Fehlerverhalten können auch scheinbar paradoxe Abhängigkeiten zwischen der Wiederherstellung einer Komponente und dem Ausfall einer anderen auftreten. So zeigen Studien, dass die Ausfallwahrscheinlichkeit einer klassischen Festplatte steigt, wenn sie nach einem Stromausfall erneut hochgefahren wird [PWB07].

Oft verlangen Wiederherstellungs- oder Übernahmemaßnahmen in IT-Systemen die Interaktion von Administratoren. Auch dies kann zu abhängigem Fehlerverhalten führen. Zunächst sind die Administratoren eine beschränkte Ressource, die nur eine begrenzte Anzahl an Aufgaben gleichzeitig wahrnehmen können [FJK14, LD11]. Dazu kommt, dass die Komplexität von IT-Systemen zu einer hohen Fehlerwahrscheinlichkeit bei der Interaktion führt [OGP03].

Um Abhängigkeiten wie diese bei der Berechnung der Systemverfügbarkeit zu berücksichtigen, können alle möglichen Zustände des Systems und ihre Übergangswahrscheinlichkeiten bzw. -raten modelliert werden [CMT+12]. Dies wird als zustandsraumbasierter Ansatz bezeichnet. Markov-Ketten ermöglichen die explizite Modellierung des Zustandsraumes, sie leiden jedoch am Problem der Zustandsraumexplosion, das aufgrund des exponentiellen Wachstums der Zustandsanzahl bei steigender Komponentenanzahl auftritt. Das Konstruieren, Speichern sowie die Lösung dieser Modelle wird dadurch stark erschwert [SKK08].

Alternativ kann der Zustandsraum auch implizit modelliert werden, um den Problemen bei der Konstruktion und Speicherung aus dem Weg zu gehen, z.B. in Petri-Netzen. Dies ist besonders effektiv, wenn die Berechnung des stabilen Zustands nicht mathematisch, sondern durch Simulation erfolgt [ZBGD10]. Dabei kann die Änderung des Zustandsraumes ausschließlich durch zeitdiskrete Ereignisse modelliert werden (Diskrete-Ereignis-Simulation) [Ban98, S. 8]. Dies wird für unterschiedliche Zufallswerte in mehreren unabhängigen Replikationen durchgeführt, um zu statistisch signifikanten Aussagen zu kommen (Monte-Carlo-Simulation).

Der Simulationsansatz ermöglicht auch die dynamische Analyse der Verfügbarkeit [Jew08]. Dies bedeutet, dass neben dem Mittelwert auch die Varianz der Verfügbarkeit betrachtet werden kann, was besonders im Sinne des AM mit Bezug auf das Service-Level-Management sinnvoll ist [Fra12]. Während die Modellkomplexität sinkt und die numerische Stabilität steigt, hat der Simulationsansatz jedoch den Nachteil hoher Zeitkomplexität für die nötige Anzahl an Replikationen. Da die Replikationen jedoch unabhängig sind, ist eine Monte-Carlo-Simulation theoretisch beliebig parallelisierbar, um den Zeitaufwand zu verringern [SKK08].

Ein anderer Ansatz, um die Komplexität von zustandsraumbasierten Ansätzen zu verringern, sind hierarchische Methoden. Hierbei werden zustandsraumbasierte Modelle

eingesetzt, um die Komponentenverfügbarkeit zu berechnen, während die Systemverfügbarkeit kombinatorisch aus den Komponentenverfügbarkeiten errechnet wird [TCD+08].

Eine durchgeführte strukturierte Literaturanalyse von insgesamt 79 Vorhersageansätzen für die Verfügbarkeit von IT-Services, welche in [BST14b] beschrieben wird, kommt zu dem Ergebnis, dass zustandsraumbasierte Methoden die Literatur dominieren, aber nur 19 % dieser Ansätze Petri-Netze als Grundlage haben und nur 12.7 % der zustandsraumbasierten Verfahren Simulation als Auswertungsmethode nutzen, obwohl diese Ansätze eine hohe Skalierbarkeit versprechen. Weiterhin ist bemerkenswert, dass nur für sechs der Ansätze eine quantitative Evaluierung der präsentierten Ansätze durchgeführt wurde. Die Zieldaten wurden jedoch in allen Fällen unter Laborbedingungen, z.B. durch Fault-Injection, erzeugt. Der Grund dafür liegt wahrscheinlich im Widerspruch zwischen der Natur von Verfügbarkeitsdaten, die erst über einen langen Zeitraum gesammelt werden können, und der Dynamik von IT-Systemen, die dazu führt, dass gesammelte Daten nach einer Änderung nicht mehr aussagekräftig sind [Mal08]. Hinzu kommen u.U. betriebliche und rechtliche Aspekte, die eine Sammlung dieser Daten behindern.

Im Folgenden werden die in dieser Arbeit verwendeten Methoden zur Verfügbarkeitsvorhersage eingeführt: es handelt sich dabei um Reliability Block Diagrams, einen kombinatorischen Ansatz, sowie Continuous-Time Markov Chains und Generalized Stochastic Petri Nets als zustandsraumbasierte Ansätze.

Reliability Block Diagram (RBD) Reliability Block Diagrams (zu deutsch etwa Zuverlässigkeits-Blockdiagramme) stellen ein einfaches kombinatorisches Verfügbarkeitsvorhersagemodell dar, das den meisten Lösungsmethoden für Redundanz-Allokation-Problemen zugrunde liegt. Sie wurden ursprünglich im militärischen Bereich entwickelt, um die Erfolgswahrscheinlichkeit von Missionen zu beurteilen (vgl. [DoD81]). Später wurden RBD auf die Zuverlässigkeitsanalyse übertragen.

Das zu untersuchende System wird im RBD zweidimensional zerlegt, zunächst in die benötigten Funktionsbausteine oder Subsysteme (serielle Kombination) und diese dann in redundant betriebene Komponenten (parallele Kombination). Um ein RBD zu visualisieren, werden die parallelen Komponenten c_{ij} für ein Subsystem i übereinander und die so gebildeten Subsysteme nebeneinander angeordnet. Ein Beispiel für eine solche Visualisierung ist in Abbildung 2.1 dargestellt.

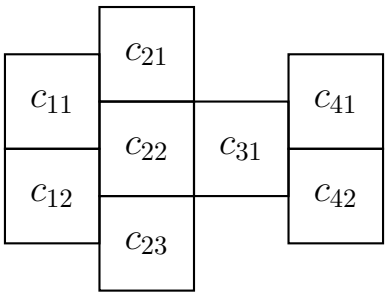

Abbildung 2.1: Beispiel eines Reliability Block Diagrams

Für jede so modellierte Komponente muss die stabile Verfügbarkeit $A(c)$ vorliegen oder aus den Fehler- und Wiederherstellungszeiten berechnet werden. Sind Komponenten in einem Subsystem s parallel angeordnet, bedeutet dies, das zugehörige Subsystem ist nur dann nicht verfügbar, wenn alle parallelen Komponenten gleichzeitig ausfallen. Dies entspricht einer ODER-Verknüpfung. Damit gilt durch die Annahme unabhängiger Komponentenfehler

$$\begin{aligned} A(s,t) = P(Z_s(t)=1) &= P\left(\bigvee_{c\in s}(Z_c(t)=1)\right) = 1 - P\left(\bigwedge_{c\in s}(Z_c(t)=0)\right) \\ &= 1 - \prod_{c\in s} P(Z_c(t)=0) = 1 - \prod_{c\in s}(1 - P(Z_c(t)=1)) = 1 - \prod_{c\in s}(1 - A(c,t)) \end{aligned}$$

und mit $t \to \infty$ $A(s) = 1 - \prod_{c\in s}(1 - A(c))$.

Das Gesamtsystem ist genau dann verfügbar, wenn alle Subsysteme verfügbar sind (UND-Verknüpfung). Hier gilt

$$A(t) = P(Z(t)=1) = P(\bigwedge_s Z_s(t)=1) = \prod_s P(Z_s(t)=1) = \prod_s A(s,t))$$

und mit $t \to \infty$ $A = \prod_s A(s)$.

Somit kann ein RBD ein seriell-paralleles System beschreiben. Für das Beispiel aus Abbildung 2.1 gilt also

$$\begin{aligned} A = [1 - (1 - A(c_{11})(1 - A(c_{12})] \cdot [1 - (1 - A(c_{21})(1 - A(c_{22})(1 - A(c_{23})] \\ \cdot A(c_{31}) \cdot [1 - (1 - A(c_{41})(1 - A(c_{42})]. \end{aligned}$$

Continuous-Time Markov Chain (CTMC) Die Zeitkontinuierliche Markov-Kette beschreibt einen diskreten Zustandsraum, in dem sich die Zustandsverteilung kontinuierlich verändert (vgl. [Fin14]). Dazu werden in der CTMC alle Zustände und die Transitionsraten zwischen ihnen modelliert. Dabei gilt die Markov-Eigenschaft, das heißt die Übergangsrate zwischen zwei Zuständen hängt nur vom aktuellen Zustand ab.

Eine CTMC ist als Tupel (S, Q) definiert, wobei S die Menge der Zustände und Q die so genannte Generatormatrix darstellen. In dieser Matrix sind die Transitionsraten definiert. Zum Beispiel kann ein System aus zwei Komponenten mit den vier Zuständen $S = \{11, 10, 01, 00\}$ beschrieben werden. Jeder Zustand repräsentiert dabei $Z(t)$ für beide Komponenten. In dieser Repräsentation besteht eine CTMC für n Komponenten aus 2^n Zuständen. Die Generatormatrix in diesem Beispiel kann wie folgt beschrieben werden:

$$Q = \begin{array}{c} \\ 11 \\ 10 \\ 01 \\ 00 \end{array} \begin{array}{c} \begin{array}{cccc} 11 & 10 & 01 & 00 \end{array} \\ \begin{pmatrix} -ZS_1 & 1/MTTF_2 & 1/MTTF_1 & 0 \\ 1/MTTR_2 & -ZS_2 & 0 & 1/MTTF_1 \\ 1/MTTR_1 & 0 & -ZS_3 & 1/MTTF_2 \\ 0 & 1/MTTR_1 & 1/MTTR_2 & -ZS_4 \end{pmatrix} \end{array}$$

Die Werte $ZS_i = \sum_{i \neq j} q_{ij}$ bezeichnen dabei die Zeilensumme der Zustandsraten. Damit gilt $\sum q_{ij} = 0$ für alle i.

Zur Visualisierung wird meist eine Graphnotation verwendet. Für das obige Beispiel ist diese Darstellung in Abbildung 2.2 illustriert. Die Zustände sind als Knoten dargestellt und die Transitionsraten als Kantenbeschriftungen eingetragen.

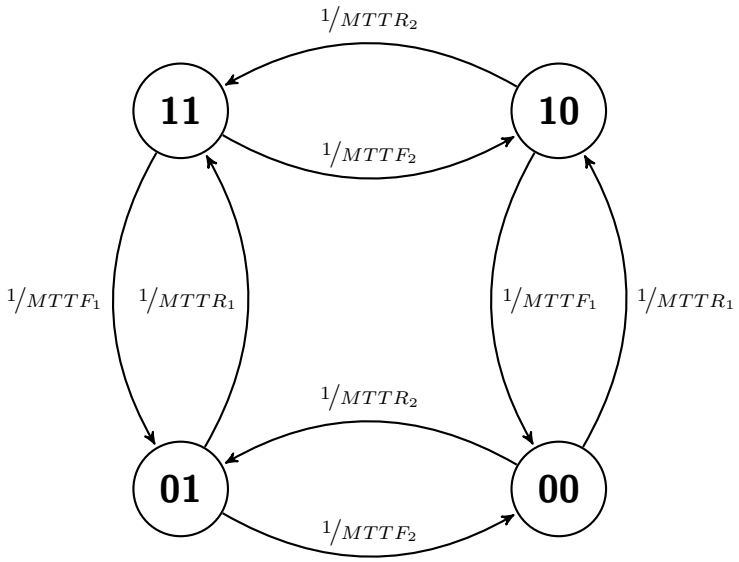

Abbildung 2.2: Beispiel einer Zeitkontinuierlichen Markov-Kette (CTMC) für zwei Komponenten

Für die Verfügbarkeitsanalyse ist vor allem das stationäre Verhalten der CTMC von Interesse. Daher soll eine Zustandsverteilung $\pi = (\pi_{11}, \pi_{10}, \pi_{01}, \pi_{00})$ identifiziert werden für die $\pi Q = 0$ gilt. Diese Zustandsverteilung ändert sich also nicht weiter, wenn sie einmal erreicht wird. Dabei repräsentiert $\pi_{ij} = \lim_{t \to \infty} P(Z_1(t) = i \wedge Z_2(t) = j)$. Die stabile Verteilung kann beispielsweise numerisch berechnet oder durch Simulation angenähert werden. Zusammen mit der Information, wie sich der Systemzustand auf die Verfügbarkeit auswirkt, kann diese berechnet werden.

Sind also die im Beispiel eingeführten Komponenten seriell abhängig, müssen beide Komponenten verfügbar sein, damit das System es auch ist. Hier gilt also $A(c_1 \wedge c_2) = \pi_{11}$. Für den Fall der parallelen Kombination gilt jedoch $A(c_1 \vee c_2) = \pi_{11} + \pi_{10} + \pi_{01}$, da bereits eine verfügbare Komponente ausreicht, damit das System verfügbar ist.

Generalized Stochastic Petri Net (GSPN) Ein GSPN (übersetzt etwa Generalisierte Stochastische Petri-Netze) ist eine Weiterentwicklung der Stochastischen Petri-Netze, die wiederum aus den Stellen/Transitions-Netzen hervorgegangen sind, die von Carl Adam Petri in den 1960er Jahren entwickelt wurden (vgl. [CMT89]). Sie können genutzt werden, um einen zeitkontinuierlichen Zustandsraum implizit zu modellieren.

Ein GSPN ist dabei ein bipartiter Graph $(S, T, F, I, m_0, G, R, D, P)$ mit

- einer Menge an Stellen S,
- einer Menge an stochastischen, deterministischen und unmittelbaren Transitionen $T = T_r \cup T_d \cup T_i$,
- einer partiellen Kantengewichtsfunktion $F :\subseteq (S \times T) \cup (T \times S) \to \mathbb{N}$,
- einer partiellen Inhibitorkantengewichtsfunktion $I :\subseteq S \times T \to \mathbb{N}$,
- einer Anfangsmarkierung $m_0 : S \to \mathbb{N}_0$,
- einer Menge von Aktivierungsfunktionen $G :\subseteq T \to B$ mit $B : M \to \{0; 1\}$ (M ist die Menge aller möglichen Markierungen),
- einer zufällig verteilten Feuerzeit $R : T_r \to X$ mit $X : \Omega \to \mathbb{R}_+$,
- einer deterministischen Feuerzeit $D : T_d \to \mathbb{R}_+$ und
- einer Feuerwahrscheinlichkeit $P : T_i \to [0; 1]$.

Stellen repräsentieren dabei Zustände oder Bedingungen. Die partiellen Funktionen können in totale Funktionen überführt werden, indem für nicht-definierte Argumente $F(s, t) = 0, I(s, t) = \infty$ und $\forall m : G_t(m) = 1$ gelten. Eine Stelle kann mit einer bestimmten Anzahl an Marken gefüllt (markiert) sein. Dies bestimmt, ob und inwieweit der Zustand oder die Bedingung gilt. Jede mögliche Markierung aller Stellen eines Netzes repräsentiert dabei einen möglichen Systemzustand. Durch Änderungen an der Markierung wird der Systemzustand manipuliert. Dies geschieht durch das so genannte Feuern von aktivierten Transitionen. Eine Transition $t \in T$ ist genau dann aktiviert, wenn

$$\forall s \in S : m(s) \geq F(s, t) \wedge m(s) < I(s, t) \wedge G_t(m) \neq 0.$$

Dies bedeutet, die Stellen im Vorbereich einer Transition müssen den Kantengewichten entsprechend markiert sein (notwendige Bedingungen). Zusätzlich darf keine Stelle mindestens w-markiert sein, die mit einer Inhibitorkante mit Gewicht w mit der Transition verbunden ist. Außerdem darf eine eventuell definierte Aktivierungsfunktion für diese Markierung nicht den Wert 0 ergeben. Das Konzept der Aktivierungsfunktion generalisiert damit die Funktionsweise von Inhibitorkanten. Durch die von Inhibitorkanten und Aktivierungsfunktionen ermöglichte Modellierung von ausschließenden Bedingungen („wenn. . . dann nicht“) gilt ein GSPN als Turing-vollständig [Zai13].

Um diese Petri-Netze zu visualisieren, können die in Abbildung 2.3 dargestellten Elemente verwendet werden. In dieser Notation sind Stellen größere leere Kreise, deren Markierung durch kleinere schwarz ausgefüllte Kreise oder durch Zahlen ausgedrückt werden kann. Diese Stellen können durch (gewichtete) Kanten mit Transitionen verbunden werden. In dieser Arbeit können einerseits stochastische und deterministische (leere Rechtecke), andererseits unmittelbare (schwarz ausgefüllte Rechtecke) Transitionen grafisch un-

terschieden werden. Inhibitorkanten haben statt einer Pfeilspitze am Ende einen kleinen leeren Kreis.

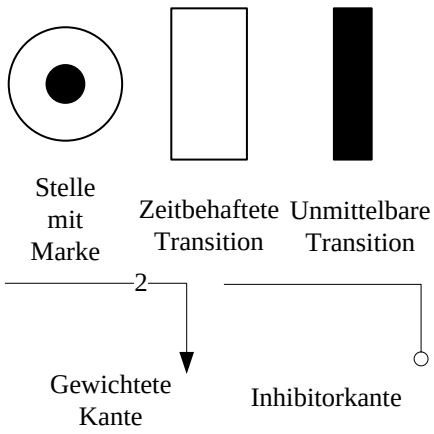

Abbildung 2.3: Grundelemente eines Generalisierten Stochastischen Petri-Netzes (GSPN)

Mit dieser Notation können auch komplexere Petri-Netze nachvollziehbar beschrieben werden. Für das oben verwendete Beispiel von zwei Komponenten könnte ein Petri-Netz wie in Abbildung 2.4 verwendet werden. Zwei Stellen geben dabei an, ob die jeweilige Komponente verfügbar (Zustand 1) oder nicht-verfügbar (Zustand 0) ist. Zwei zeitbehaftete Transitionen entsprechen dem Ausfall bzw. der Wiederherstellung einer Komponente. So sind vier Markierungen möglich (11, 10, 01 und 00) und der Erreichbarkeitsgraph dieses Netzes entspricht der CTMC aus Abbildung 2.2. Folgt man diesem Ansatz, erhält man für n Komponenten $4n$ Knoten und genauso viele Kanten.

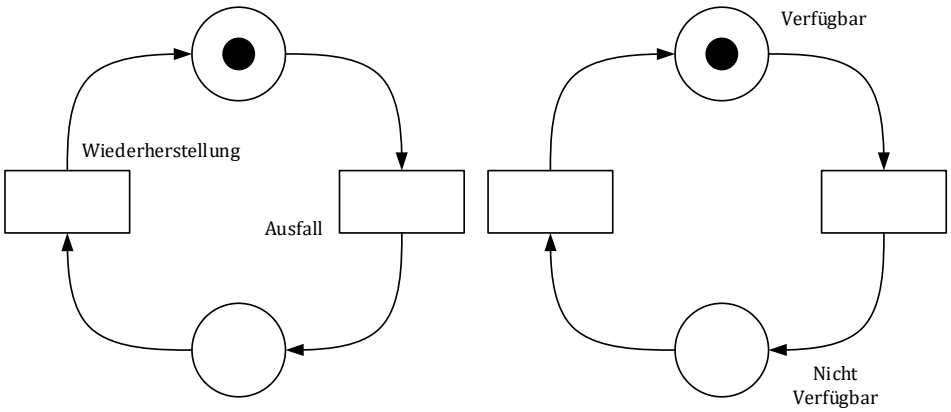

Abbildung 2.4: Einfaches GSPN-Modell für zwei Komponenten

Ist eine Transition t unter m aktiviert (z.B. die Transition „Ausfall“ in Abbildung 2.4), so wird abhängig von der Art der Transition eine Feuerzeit bestimmt. Ist $t \in T_s$,

wird eine zufällige Feuerzeit $R(t, \omega)$ bestimmt. Für deterministische Transitionen $t \in T_d$ ist die Feuerzeit $D(t)$. Unmittelbare Transitionen $t \in T_i$ haben eine Feuerzeit von Null, feuern also unverzüglich nach Aktivierung.

Bleibt eine Transition nach der Aktivierungszeit für die Feuerzeit aktiviert, so feuert sie und verändert die Markierung des Netzes

$$m \rightarrow_t m' \text{ mit } m'(s) = m(s) - F(s,t) + F(t,s) \text{ für alle } s \in S.$$

Der Fall, dass eine Transition vor Verstreichen der Feuerzeit wieder inaktiv wird, kann eintreten, wenn mehrere Transitionen unter einer Markierung aktiviert sind, die gemeinsame Stellen im Vorbereich haben. Dies wird als Konflikt bezeichnet. Für die Auflösung von Konfliktsituationen wird jeder zeitbehafteten Transition $t \in T_r \cup T_d$ eine Altersvariable a_t zugeordnet. Ist diese Altersvariable zum Zeitpunkt der Aktivierung ungleich Null, wird der Wert der Altersvariable als Feuerzeit angesetzt.

Besteht ein Konflikt zwischen einer unmittelbaren und mehreren zeitbehafteten Transitionen, feuert die erstere. Sind in einer Konfliktsituation mehrere unmittelbare Transitionen aktiviert, wird unter Berücksichtigung von P eine zufällige Transition zum Feuern ausgewählt.

Bei einem Konflikt zwischen mehreren zeitbehafteten Transitionen, bei dem keine unmittelbare Transition involviert ist, feuert die Transition, für welche die Aktivierungszeit addiert mit der Feuerzeit am geringsten ist. Dies wird als Rennen (engl. *race*) bezeichnet. In diesem Fall kann die Altersvariable der nicht feuernden Transitionen neu gesetzt werden, abhängig von der Rennstrategie der entsprechenden Transition [MBB+89].

Im Falle von *race resampling* wird die Altersvariable aller Transitionen immer auf Null gesetzt (Standard). Ist die Strategie *race enabled*, wird die Altersvariable einer Transition, die auch unter der neuen Markierung aktiviert ist, auf die verbleibende Feuerzeit gesetzt. Wenn für die Transition *race age* ausgewählt wurde, wird die Altersvariable einer nicht feuernden Transition immer auf die verbleibende Feuerzeit gesetzt. Je nachdem welche Art Aktivität von der Transition repräsentiert wird, können verschiedene Rennstrategien sinnvoll sein.

Zusätzlich können so genannte *Rewards* definiert werden [CMT89]. Diese verändern globale Variablen entweder einmalig durch das Eintreten von Ereignissen im Petri-Netz (Impulse-Reward) oder kontinuierlich solang ein bestimmter Zustand gilt (Rate-Rewards). Diese können zum Beispiel genutzt werden, um die Verfügbarkeit als globale Variable abhängig vom Zustand zu verändern.

Gelten für zwei Markierungen m und m' sowie eine Transition t $m \rightarrow_t m'$, stehen m' von m in einer Erreichbarkeitsrelation. Durch die gegebene Anfangsmarkierung m_0 bilden alle erreichbaren Folgemarkierungen den so genannten Erreichbarkeitsgraph des GSPN. Dieser entspricht dem expliziten Zustandsraum.

In einer Diskreten-Ereignis-Simulation wird ein Pfad des Erreichbarkeitsgraphs abhängig von einem Zufallszahlengenerator verfolgt. Dabei werden globale Variablen, z.B. für die durchschnittliche Verfügbarkeit, durch Rewards verändert. Da dieses Einzelergebnis je-

doch statistisch nicht signifikant ist, wird diese Simulation mit verschiedenen Anfangswerten für den Zufallsgenerator (*random seeds*) in unabhängigen Replikationen durchgeführt (Monte-Carlo-Simulation). Somit kann die Verfügbarkeit des stabilen Zustands $\hat{A}$ durch ein α-Konfidenzintervall aus den Verfügbarkeiten der Replikationen A_i geschätzt werden (Mittelwert eines unbekannt verteilten Merkmals mit unbekannter Varianz, z_q ist dabei das q-Quantil der Standardnormalverteilung):

$$P(\hat{A} \in [\bar{A} - z_{1-\alpha/2}\frac{s}{\sqrt{n}}; \bar{A} + z_{1-\alpha/2}\frac{s}{\sqrt{n}}]) = 1 - \alpha$$

$$\text{mit } \bar{A} = \frac{1}{n}\sum_{i=1}^{n} A_i$$

$$\text{und } s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(A_i - \bar{A})^2.$$

Aus diesen Konfidenzintervallen können Informationen über die Varianz für die oben erwähnte dynamische Analyse der Zielgrößen abgeleitet werden.

2.1.4 Kosten von IT-Services

In den untersuchten Ansätzen zur Verfügbarkeitsvorhersage werden die Kosten der IT-Services in der Regel nicht betrachtet. Im Sinne einer Optimierung der Verfügbarkeit sollten diese jedoch in ein gemeinsames Modell integriert werden, da eine Steigerung der Verfügbarkeit in der Regel mit höheren Kosten verbunden ist. Dazu müssen die Kosten eines IT-Services zumindest grob aufgeschlüsselt werden.

Allgemein lassen sich die Kosten von IT-Systemen in kapitale und operationale Kosten unterteilen [BCH13]. Kapitale Kosten bezeichnen dabei Investitionen, die über einen bestimmten Zeitraum abgeschrieben werden. Dies sind normalerweise initiale Kosten, wie die Beschaffungskosten des IT-Equipments, also der benötigten Hard- und Software, sowie u.U. Konstruktionskosten des Rechenzentrums, um Ressourcen wie Platz, Strom, Netzanbindung oder Wasser zur Verfügung zu stellen. Diese können auch Kosten für die Entwicklung oder Anpassung von Software-Komponenten beinhalten.

Operationale Kosten sind Ausgaben, die im laufenden Betrieb anfallen. Dies sind im Falle von IT-Systemen hauptsächlich die Stromkosten zum Betrieb des IT-Equipments sowie der Kühlungsinfrastruktur [BCH13]. Studien wie [PS05] zeigen jedoch, dass die Kühlkosten direkt proportional zu den Stromkosten der IT-Komponenten sind [PS05] und daher geschätzt werden können. Weiterhin zählen zu den operationalen Ausgaben auch Personalkosten für Administratoren, Sicherheitspersonal, etc. [BCH13, S. 71] sowie Kosten, die durch die Ersetzung defekter IT-Komponenten resultieren, die als Wiederherstellungskosten bezeichnet werden können [LD09]. Falls die Erbringung eines IT-Services durch andere IT-Services unterstützt wird, so fallen operationale Kosten für die Beziehung dieser Dienstleistungen an.

2.2 Das Redundanz-Allokation-Problem

In diesem Abschnitt werden die Grundlagen des Redundanz-Allokation-Problems (RAP) vermittelt. Zusätzlich wird die bisherige Entwicklung der RAP-Definitionen und -Lösungsalgorithmen in der Forschung verfolgt. Im Anschluss werden die in dieser Arbeit verwendeten Lösungsalgorithmen vorgestellt.

Um die Verfügbarkeit eines Designs zu erhöhen, bestehen prinzipiell zwei Möglichkeiten: die Nutzung zuverlässigerer Komponenten oder die Einführung von Redundanz-Mechanismen [GS13]. Die Effektivität des ersteren Ansatzes ist jedoch insbesondere bei der Verwendung von Software-Komponenten limitiert, da sich deren Verfügbarkeit nicht beliebig erhöhen lässt [CK90]. Andererseits können bei der Einführung von Redundanz-Mechanismen hohe Kosten entstehen. Diese müssen also durch die erzielte Steigerung der Verfügbarkeit gerechtfertigt sein.

Diese Abwägung kann durch Definition eines Redundanz-Allokation-Problems (RAP) unterstützt werden. Dabei werden nötige Funktionsbausteine eines Systems, so genannte Subsysteme, sowie mögliche Komponenten und Redundanzmechanismen für jedes Subsystem modelliert. Durch Lösung eines RAP werden Konfigurationen von Komponenten und Redundanzmechanismen identifiziert, welche die Zielfunktionen, bspw. für Verfügbarkeit oder Kosten, optimieren. Aufgrund der Problemkomplexität kann jedoch i.d.R. die Identifikation eines optimalen Designs nicht garantiert werden. Im Folgenden werden daher die Lösungen eines RAP als (sub)optimale Designs bezeichnet.

Die Geschichte der RAP geht bis in die 1960er-Jahre zurück, als die ersten Beschreibungen dieser Art von Optimierungsproblemen veröffentlicht wurden. Anfangs konnten diese exakt durch mathematische Verfahren gelöst werden. Nicht zuletzt durch die immer weiter verbreitete Nutzung von IT-Systemen mussten die RAP-Definitionen erweitert werden. Diese führten zu dermaßen komplexen Problemen, dass auch Meta-Heuristiken und andere Approximationsverfahren zur effizienten Lösung eines RAP eingesetzt wurden. Die Entwicklung der Definitionen wird im folgenden Abschnitt nachgezeichnet. Weiterführende Informationen können u.a. in Literaturanalysen im Bereich RAP ([KP00, GY06, Sol14]) gefunden werden.

2.2.1 Problemdefinitionen

Im Jahr 1962 war Ketelle einer der ersten Wissenschaftler, die ein Redundanz-Allokation-Problem beschrieben [KJ62]. Dabei können für eine Anzahl von Subsystemen homogene Komponenten in beliebiger Zahl aktiv-redundant betrieben werden. Die Kosten des Systems wachsen dabei linear zu der Anzahl verwendeter Komponenten. Eine dynamische Programmierung wird genutzt, um die Kosten eines Systems unter einer Verfügbarkeitsbedingung zu minimieren.

Dieses Optimierungsziel wird z.B. auch in [FHL68, Mis71] und [SS12] verfolgt. Andere Arbeiten nutzen RAP, um die Verfügbarkeit eines Systems unter einer Kostenbedingung zu maximieren (z.B. in [RMC04] und [ZGLW11]). Auch die gleichzeitige Maximierung der

Verfügbarkeit bei Minimierung der Kosten als mehrkriterielles RAP wird angestrebt, z.B. in [CK06, KKCB08, LD09, WCTY09, YH11, CRNK12, SWCC12] und [GS13].

Für die folgende RAP-Definition konnte Chern 1992 deren NP-Schwere beweisen [Che92]:

i Ein System besteht aus n seriell verknüpften Funktionsbausteinen (Subsysteme).

ii In einem Subsystem können eine beliebige Anzahl von Komponenten in aktiver Redundanz betrieben werden. Das System kann also in Anlehnung an elektrische Schaltkreise als seriell-paralleles System beschrieben werden.

iii Eine Komponente in einem Subsystem ist entweder verfügbar oder nicht (binärer Zustand), Komponentenausfälle sind dabei unabhängig voneinander und identisch verteilt für ein Subsystem (homogene Redundanz).

iv Die Verfügbarkeit des Systems ist zu maximieren unter Berücksichtigung linear wachsender Nebenbedingungen wie für Kosten, Gewicht oder Volumen.

Trotz der NP-Schwere dieser Definition kann sie in vielen realen Szenarien aufgrund ihrer Annahmen nicht angewendet werden. Daher wurden die RAP-Definition in den folgenden Jahren erweitert.

Um heterogene Redundanz abzubilden, wurden z.B. in [CS96a, LS04, CY05, CY05, CK06, OKJN07, LD09] und [SWCC12] Komponenten mit unterschiedlicher Verfügbarkeit in einem Subsystem erlaubt. Durch die Einführung von Übernahmezeiten, z.B. in [TY99, CRNK12, AH14] und [SS15], kann auch passive Redundanz in einem RAP definiert werden. Multiple Komponentenzustände wurden beispielsweise in [RMC04, ONG08, TLZ09] und [AAMY13] modelliert, um z.B. das Problem der Leistungsabnahme (engl. *performance degradation*) abzubilden. Eine komplexere Anordnung von Subsystemen als in einer parallel-seriellen Form wurde z.B. in [RMR97, Che06, SBR10] und [Zia13] durch die Analyse hierarchischer oder komplexer Designs untersucht, wenn auch nur für Einzelfälle.

Neben der Erhöhung der Komplexität der Definition ist auch der Aspekt der ungenauen Eingangsparameter für RAP intensiv beforscht worden. Dies wird der schwierigen Parametrisierung von RAP durch das Fehlen von langfristigen Verfügbarkeitsdaten, besonders bei Software-Komponenten, gerecht. Um dies abzubilden, können RAP-Parameter als stochastische, fuzzy, fuzzy-stochastische und Intervall-Größen beschrieben werden [Sol14]. Im Fall der stochastischen Eingangsgrößen werden die Verfügbarkeiten der definierten Komponenten nicht als konkreter Wert, sondern als Zufallsgröße modelliert [PC95]. In der Regel werden dann die Erwartungswerte der Zielgrößen untersucht. Stochastische Ansätze können aber durch eine zusätzliche Betrachtung der Varianz der Zielgrößen risikobewusste Entscheider besser unterstützen [SS15]. Es kann jedoch argumentiert werden, dass die Ungenauigkeit der Parameter eher durch Ungewissheit als durch reinen Zufall entsteht, was die Anwendung des rein stochastischen Ansatzes in Frage stellt [JZMY14, GS13].

Eine Alternative zur Abbildung von Ungewissheit statt Zufall ist die Verwendung von Fuzzy-Größen, z.B. in triangularer Form [WW09]. Auch hier werden normalerweise

Erwartungswert und Streuung der Fuzzy-Zielgrößen untersucht (z.B. in [GS13, GRSV14] und [JZMY14]). Damit geht jedoch ein gewisser Informationsverlust einher, da die konkrete Berechnung der Fuzzy-Zielgrößen komplex ist. Eine einfachere Variante ist daher die Verwendung von Intervall-Größen (z.B. in [TY99, MSB12] und [FAM14]). Dies hat den Vorteil, dass einerseits weniger a priori-Wissen zur Definition der Parametergrößen vonnöten ist [SS15] und andererseits nur die Intervallgrenzen bei Operationen betrachtet werden müssen [SBR10].

Bei der Unterscheidung von Zufall und Ungewissheit kann jedoch eingewendet werden, dass statistisch bestimmte Daten nie vollkommen gewiss oder ungewiss sind, daher sind beide Aspekte oft gemischt [WW09]. Um dies abzubilden, können Zufällige Fuzzy-Größen verwendet werden. Dies sind Fuzzy-Größen, deren Parameter einer Zufallsverteilung unterliegen [ZL04]. Alternativ können Fuzzy-Zufallsgrößen definiert werden. Hierbei ist ein Modellparameter als Zufallsgröße beschrieben, deren Realisierungen jedoch Fuzzy-Größen [Kwa78] sind.

Trotz vieler Erweiterungen der RAP-Definition gibt es nur wenige Arbeiten, die die Annahme unabhängiger Komponentenfehler zu überwinden versuchen. Da die kombinatorische Berechnung der Verfügbarkeit, die in anderen Arbeiten verwendet wird, in diesem Falle nicht anwendbar ist, müssen zustandsraumbasierte Verfahren herangezogen werden.

In [CK90] wurden von Chi und Kuo beispielsweise Ausfälle aufgrund gemeinsamer Ursachen modelliert. Dazu kann eine zusätzliche „Common-Cause-Komponente“ in Subsystemen eingeführt werden. Ein Ausfall dieser Komponente führt dann zum Ausfall des gesamten Subsystems. Dabei wird ein unterliegender Markov-Prozess modelliert, um die Verfügbarkeit zu berechnen. Obwohl ein Ausfall aufgrund gemeinsamer Ursachen auch heterogene Subsysteme betreffen kann, müssen Komponenten in diversitärer Redundanz nicht immer betroffen sein. Diese Tatsache ist im Ansatz von Chi und Kuo jedoch nicht abgebildet. Dies gilt auch für Abhängigkeiten, die sich über mehrere Subsysteme erstrecken können. Weitere Abhängigkeiten wie imperfekte Übernahme oder fehlerbehaftete Administratorinteraktion können ebenfalls nicht modelliert werden.

Lins und Droguett modellieren in [LD09] den Ausfall-Wiederherstellungs-Zyklus von Komponenten als alternativen erneuerbaren Prozess mit einer Weibull-verteilten Ausfallzeit und exponentialverteilten Wiederherstellungszeiten, die durch geplante Wartungsarbeiten manipuliert werden können. Da dieser Prozess nicht analytisch auswertbar ist, wurde eine Diskrete-Ereignis-Simulation zur Berechnung der Komponentenverfügbarkeit genutzt. Die Systemverfügbarkeit wurde mittels eines Binären Entscheidungsdiagramms kombinatorisch berechnet, was dem Schema eines hierarchischen Ansatzes zur Verfügbarkeitsvorhersage entspricht. Trotz der Modellierung von Wartungsarbeiten und -kosten sind keine Kapazitäten für diese Arbeiten modelliert, sodass theoretisch unendlich viele Wartungsarbeiten gleichzeitig stattfinden können. Diese Kapazitäten sind jedoch in der Realität u.U. nicht beliebig hoch- bzw. herunterskalierbar, da ggf. neue und kostspielige Stellen geschaffen werden müssen. Zusätzlich können nur perfekte Wartungen modelliert werden, was bedeutet, dass die Möglichkeit von Interaktionsfehlern nicht berücksichtigt

werden kann.

Die vorangegangene Analyse zeigt, dass Markov- und Simulationsansätze zur Überwindung der Annahme unabhängiger Komponentenfehler in RAP geeignet sind. Dennoch bleibt die Anwendung dieser Ansätze in einem RAP für IT-Service-Design fraglich, da höchstens Teilaspekte der Verfügbarkeit und Kosten von IT-Services in den RAP-Definitionen berücksichtigt werden.

2.2.2 Lösungsalgorithmen

Nachdem in einer RAP-Definition die Abhängigkeit der Systemverfügbarkeit von der Verfügbarkeit der möglichen Komponenten sowie der Suchraum und das Optimierungsziel beschrieben sind, können Lösungsalgorithmen (sub)optimale Konfigurationen ermitteln. In seiner Literaturanalyse [Sol14] identifiziert Soltani drei Klassen von Lösungsalgorithmen: Ansätze der mathematischen Programmierung, Heuristiken und Meta-Heuristiken.

Mathematische Methoden wie lineare, dynamische oder nicht-lineare Programmierung können exakt oder approximativ eingesetzt werden, z.B. in [KJ62, FHL68, Mis71, OKJN07, KDA12, CMC13] und [CV15]. Ihre Anwendbarkeit wird jedoch durch die NP-Schwere aufgrund der komplexen Definitionen eingeschränkt [Sol14], weswegen diese Methoden oft den Lösungsraum massiv beeinträchtigen oder sehr ineffizient durchsuchen [CS96a]. Daher führte der Bedarf an flexibleren und effizienteren Lösungsverfahren zur Entwicklung bzw. Anpassung von (Meta-)Heuristiken [KKSC03].

Heuristiken sind Verfahren, die für ein spezielles Problem entwickelt werden, was i.d.R. deren Transferierbarkeit auf andere Probleme erschwert [Sol14]. Meta-Heuristiken stellen allgemeine Heuristiken dar, deren Prozedur auf eine Vielzahl verschiedener Probleme angewendet werden kann, wenn bestimmte problemspezifische Operationen definiert werden. Auch wenn sie mehr Parametereinstellungen als Heuristiken benötigen, produzieren sie im Allgemeinen bessere Ergebnisse [CK06]. Damit bieten sie einen guten Kompromiss aus Übertragbarkeit, Effizienz und Lösungsqualität.

Dabei basieren die meisten Meta-Heuristiken auf beobachtbaren Naturphänomenen und damit mehr auf künstlicher Logik als auf klassischer Mathematik [KP00]. Durch die Tatsache, dass normalerweise viele Lösungskandidaten in einem Iterationsschritt einer Meta-Heuristik betrachtet werden, können gleichzeitig die Exploration des gesamten Suchraumes und die Exploitation vielversprechender Lösungsregionen durchgeführt werden. Dies ist besonders bei der Lösung von mehrkriteriellen Problemen sehr effektiv [LD09].

Soltani identifiziert folgende Arten von Meta-Heuristiken, die auf RAPs angepasst wurden [Sol14]: Genetische Algorithmen, Tabu-Suche, Partikelschwarm-, Honigbienenkolonie- und Ameisenkolonieoptimierung sowie Simuliertes Ausglühen, Honigbienen-Paarungsalgorithmen, Harmoniesuche, Immunbasierte Algorithmen und Kuckuck-Suche.

In einem Genetischen Algorithmus (GA) werden Lösungskandidaten als Zeichenfolgen kodiert. Ähnlich der Entwicklung der evolutionären Entwicklung einer Genomsequenz werden neue Lösungen durch Mutation und Rekombination dieser Zeichenfolgen kreiert. Durch Nachahmung der natürlichen Selektion wird eine Verbesserung der Lösungsqualität

über mehrere Generationen angestrebt. Beispiele für die Anwendung eines GA im RAP-Kontext finden sich in [CS96b, CS96a, TY99, LD09, WCTY09, CRNK12, SBR10, LD11, SWCC12, AH14] und [ZHA14].

Tabu-Suche (TS) bezeichnet einen Prozess, in dem in jeder Iteration versucht wird, einen Lösungskandidaten durch eine Nachbarlösung höherer Qualität abzulösen. Um Zyklen oder die vorzeitige Konvergenz dieser Verfahren zu verhindern, werden so genannte Tabus definiert, die bestimmte Suchrichtungen einschränken. Diese werden meist aus der bisherigen Bewegung im Suchraum abgeleitet und nach einigen Iterationen nicht mehr berücksichtigt. Tabu-Suche wurde beispielsweise in [KKSC03, KKSN06, KKCB08, ONG08] und [ONG10] für RAP angewendet.

Bei der Partikelschwarmoptimierung (PSO) werden Methoden der Schwarmintelligenz genutzt. So „bewegt“ sich ein Partikel durch den Lösungsraum und orientiert sich dabei an seiner bisherigen Bewegung, dem bisherigen Optimum des Partikels sowie dem globalen Optimum des Schwarms (z.B. in [GS13, Yeh14] und [DZKD15]).

Das Suchverhalten von Honigbienen bei der Nahrungsbeschaffung wird in der Künstlichen Bienenkolonie (ABC von engl. *artificial bee colony*) nachgeahmt. Dabei repräsentiert jede Lösung eine Nahrungsquelle und Arbeiterinnen (lokale Optimierung), Zuschauer (wechseln zwischen lokalen Optima) sowie Kundschafter (Zufallssuche) suchen nach optimalen Lösungen. Die ABC wird beispielsweise in [YH11, HY12] und [JZMY14] auf ein RAP angepasst.

In der Ameisenkolonieoptimierung (ACO von engl. *ant colony optimization*) wird die Qualität von (Teil-)Lösungen zwischen den Kandidaten kommuniziert, um so den Suchprozess zu steuern, ähnlich wie dies durch Pheromone bei der Nahrungssuche von Ameisen geschieht. Wie ihre natürlichen Pendants „verflüchtigen“ sich Qualitätseinschätzungen, wenn bessere Lösungen gefunden werden (z.B. in [LS04]).

Das Verfahren des Simulierten Ausglühens (SA) bildet die Aushärtung von heißem Metall bei seiner Abkühlung nach. So bestimmt ein über die Iterationen steigender Temperaturparameter inwieweit sich eine Lösung in der nächsten Generation verschlechtern darf. Damit soll die verfrühte Konvergenz zu einem lokalen Optimum verhindert (Exploration) und gleichzeitig gegen Ende des Verfahrens Regionen mit hoher Lösungsqualität durchforstet werden (Exploitation). Dieses Verfahren wurde z.B. in [RMR97] und [CNRK13] für die Lösung von RAP genutzt.

Ein Honigbienen-Paarungs-Algorithmus (HBMO von engl. *honey bee mating optimization*) basiert auf der Kombination von Lösungskandidaten (Drohnen) mit globalen Optima (Königinnen). Verschiedene Heuristiken (Arbeiterinnen) werden genutzt, um Drohnen lokal zu modifizieren (z.B. in [SS12]).

Bei der Harmoniesuche (HS) wird das Finden einer harmonischen Melodie in einer Band von Musikern imitiert. Es werden neue Lösungskandidaten aus bisherigen Lösungen (Gedächtnis) mit zufälligem Einfluss konstruiert (Tonhöheneinstellung). Zusätzlich können Lösungen innerhalb definierter Grenzen zufällig erstellt werden (Improvisation). Dieses Verfahren wird beispielsweise in [ZGLW11] und [WL12] angewendet.

Immunbasierte Algorithmen (IA) nutzen ein Immungedächtnis, um Crowding (d.h.

die Konvergenz der Population) zu vermeiden. Ähnliche Lösungskandidaten werden dabei konsolidiert, wie z.B. in [CY05, Che06] und [HY11].

Die Kuckuck-Suche (CS von engl. *cuckoo search*) ahmt die Fortpflanzungsstrategie dieser Tiere nach. Dabei werden bisherige (normale Eier) und neue (Kuckuckseier) Lösungen zusammen in einem „Nest“ betrachtet. Je höher die Qualitätsdifferenz zwischen dem Kuckucksei und den normalen Eiern ist, desto höher ist die Wahrscheinlichkeit, dass die neue Lösung oder sogar das gesamte Nest verworfen werden (eingesetzt z.B. in [KPJ13] und [VV13]).

Die meisten dieser Algorithmen basieren auf gemeinsamen (evolutionären) Prinzipien. Daher sind die zu instanziierenden Operationen in der Regel eine Auswahl aus Mutation (lokale Nachbarschaftssuche), Rekombination zweier oder mehrerer Lösungskandidaten sowie Selektion (zufällige, aber qualitätsorientierte Auswahl von Lösungskandidaten). Die Qualität einer Lösung wird dabei als Fitness bezeichnet. In der Regel wird versucht, Lösungen höherer Fitness zu erzeugen und Lösungen niedriger Fitness auszusortieren (engl. *survival of the fittest*). Dieser so genannte Selektionsdruck sollte über die Generationen steigen.

Die Auswahl einer geeigneten Fitnessfunktion hängt vom untersuchten Optimierungsproblem ab. Für RAP bedeutet dies, das entweder die Verfügbarkeit oder die negativen Kosten als Fitness ausgewählt werden, da die Meta-Heuristiken für Maximierungsprobleme definiert sind. Jedoch muss verhindert werden, das am Ende des Lösungsverfahrens die Ergebnisse aufgrund von Verletzungen der Nebenbedingungen unzulässig sind. Dies kann z.B. durch Reparaturmechanismen gewährleistet werden. Diese führen jedoch meist zu einer ungewünschten Beeinflussung des Suchprozesses, da zulässige Lösungen mit hoher Fitness u.U. nur durch die Berücksichtigung unzulässiger Lösungen gefunden werden können. Daher werden in der Literatur Strafterme für die Fitness empfohlen, abhängig vom Grad der Verletzung und der Generationszahl [CS96a].

Im Falle der mehrkriteriellen Optimierung kann ein einzelner Fitnesswert durch Gewichtung der Zielwerte berechnet werden [FF95]. Dies erfordert jedoch a priori-Wissen über das konkrete Problem. Ist dieses nicht vorhanden, kann ein Pareto-Ansatz gewählt werden. Hierbei wird jedem Kriterium ein eigener Fitnesswert zugeordnet, sodass ein Fitnesstupel entsteht. Als Ergebnis eines Pareto-basierten Optimierungsansatzes wird eine Menge nicht-dominierter Lösungen gesucht, die so genannte Pareto-Front. Ein Lösungskandidat x_1 mit Fitness $(f_1(x_1), \ldots, f_i(x_1), \ldots, f_n(x_1))$ dominiert einen Lösungskandidaten x_2 mit $(f_1(x_2), \ldots, f_i(x_2), \ldots, f_n(x_2))$ genau dann, wenn ein k existiert, sodass $f_k(x_1) > f_k(x_2)$ mit $f_i(x_1) \geq f_i(x_2)$ für $1 \leq i \leq n, i \neq k$. Dies wird mit $x_1 \succ x_2$ dargestellt.

Die Performance der adaptierten Meta-Heuristiken wird dabei meistens in numerischen Beispielen, z.B. Nakagawas und Miyazakis 33 Probleme aus [NM81], vergleichsweise bestimmt. Die Frage welcher Algorithmus unter welchen Bedingungen zu empfehlen ist, könnte erst durch extensive numerische Untersuchungen beantwortet werden [KP00]. Daher werden in dieser Arbeit mehrere Lösungsalgorithmen adaptiert, nämlich ein Genetischer Algorithmus, Tabu-Suche, Partikelschwarmoptimierung und die Künstliche Bienenkolonie.

Damit werden in dieser Arbeit sowohl zwei Algorithmen mit starker Rekombinationskomponente (Genetischer Algorithmus und Partikelschwarmoptimierung) als auch zwei Algorithmen mit Fokus auf Nachbarschaftssuche (Tabu-Suche und Künstliche Bienenkolonie) eingesetzt. Da besonders der Genetische Algorithmus, aber auch die Tabu-Suche, intensiv in der RAP-Forschung untersucht wurden, bieten sich diese Algorithmen für einen Vergleich der Lösungsalgorithmen an. Sowohl die Partikelschwarmoptimierung [Yeh14] als auch die Künstliche Bienenkolonie [HY12] konnten die Performance des Genetischen Algorithmus und der Tabu-Suche in bestimmten Szenarien übertreffen und wurden daher ausgewählt.

2.2.2.1 Genetischer Algorithmus

Der Genetische oder Evolutionäre Algorithmus (GA) ist ein aus algorithmischer Perspektive sehr einfacher Algorithmus, verlangt aber die Definition einer hohen Zahl an problemspezifischen Operationen. Dabei versucht der GA die natürliche Evolution als Zusammenspiel aus Mutation, Rekombination und Selektion über mehrere Generationen nachzuahmen. In diesem Kontext wird ein Lösungskandidat als Individuum bezeichnet, die Menge der Lösungskandidaten einer Generation als Population.

Nach Bäck et al. [Bäc96] können grundsätzlich zwei Formen evolutionärer Algorithmen unterschieden werden, die Plus- und die Komma-Selektion. Sie unterscheiden sich in der Tatsache, welche Individuen für die nächste Generation berücksichtigt werden können. Im Fall der Plus-Selektion werden in einer Generation sowohl die Individuen der letzten Generation als auch die durch Rekombination dieser Lösungen erstellten Individuen betrachtet (vgl. Algorithmus 1).

Algorithmus 1 Genetischer Algorithmus: $(\mu + \lambda)$-Plus-Selektion

```
 1: procedure GA_PS(μ, λ)
 2:     gen ← 0
 3:     pop ← INITIALIZE(μ)
 4:     EVALUATE(pop,0)
 5:     while not TERMINATIONCRITERIA(pop, gen) do
 6:         pop ← pop ∪ RECOMBINE(pop, λ)
 7:         pop ← MUTATE(pop)
 8:         EVALUATE(pop,gen)
 9:         pop ← SELECT(pop, μ)
10:         gen ← gen + 1
11:     end while
12:     return pop
13: end procedure
```

Dieses Vorgehen garantiert, dass Lösungen mit hoher Fitness in der Population verbleiben und dennoch Subjekt der Mutation sind. Dadurch kann jedoch die vorzeitige Konvergenz zu einem lokalen Optimum eintreten. In der Komma-Selektion können nur Individuen der aktuellen Generation ausgewählt werden, womit die Wahrscheinlichkeit der frühzeitigen Konvergenz gesenkt wird. Auf der anderen Seite kann jedoch nicht garantiert

werden, dass die besten gefundenen Lösungen auch in der Ergebnispopulation vorhanden sind. Alternativ kann die Komma-Selektion so genannten n-Elitismus umsetzen. In diesem Fall werden die n Individuen der höchsten Qualität unverändert in die nächste Generation übernommen, können aber dennoch durch Mutation verbessert werden (vgl. Algorithmus 2).

Algorithmus 2 Genetischer Algorithmus: (μ, λ)-Komma-Selektion mit 1-Elitismus

```
 1: procedure GA_CS(μ, λ)
 2:     gen ← 0
 3:     pop ← INITIALIZE(μ)
 4:     EVALUATE(pop,0)
 5:     while not TERMINATIONCRITERIA(pop, gen) do
 6:         elite ← BEST(pop)
 7:         pop ← RECOMBINE(pop, λ)
 8:         pop ← MUTATE(pop)
 9:         EVALUATE(pop,gen)
10:         pop ← SELECT(pop, μ − 1)
11:         pop ← pop ∪ elite
12:         gen ← gen + 1
13:     end while
14:     return pop
15: end procedure
```

Dabei bezeichnet μ die Größe der Population und λ die Anzahl an Individuen, die in jeder Generation durch Rekombination neu erstellt werden. Zunächst werden also μ Individuen zufällig erzeugt. Dazu muss eine Zufallserstellung definiert werden (Funktion INITIALIZE). Daraufhin wird die Fitness der erstellten Population evaluiert (Funktion EVALUATE).

Solang eine von Population und Generationsanzahl abhängige Terminierungsbedingung (Funktion TERMINATIONCRITERIA) nicht erfüllt ist, wird eine neue Generation der Population gebildet. Im Falle der in Algorithmus 1 dargestellten Plus-Selektion wird die (temporäre) Population durch die „Nachkommen“ der Originalindividuen angereichert, wobei die Nachkommen in der Komma-Selektion die vorherige Population ersetzen. Für die Bildung von Nachkommen werden λ Individuen aus den Individuen der letzten Generation durch Rekombination erstellt (Funktion RECOMBINE). In der Regel werden für die Erstellung zweier Nachkommen zwei Individuen zufällig, aber unter Berücksichtigung der Fitness, ausgewählt und deren Merkmale rekombiniert. Theoretisch sind aber auch Rekombinationen von mehr als zwei Individuen denkbar.

Nach der Rekombination wird die Population der Mutation unterworfen (Funktion MUTATE). Dabei werden normalerweise Individuen mit Wahrscheinlichkeit p_{mut} ausgewählt und einzelne Merkmale verändert. Nun wird die Population erneut evaluiert, damit eine fitnessbasierte Selektion (Funktion SELECT) erfolgen kann. Sobald die Terminierungsbedingung erfüllt ist, wird die aktuelle Population als Ergebnis ausgegeben.

Selektionsmethoden Für die Operation der Selektion stehen mehrere Standardverfahren zur Verfügung. Die einfachste Möglichkeit ist es, die μ Individuen mit der höchsten Fitness auszuwählen. Dies wird als rangbasierte Selektion bezeichnet und z.B. in [CS96a, CS96b] und [TY99] für RAP angewendet. Dieses Verfahren hat den Vorteil, dass frühes Crowding vermieden wird. Nachteilig ist jedoch die fehlende Zufallskomponente. Dadurch besteht die starke Gefahr, dass die Population zu lokalen Optima konvergiert. Alternativ kann eine zufällige Auswahl der Individuen auf Basis des Fitnessrangs erfolgen (rangproportionale Selektion, z.B. in [SWCC12]).

Eine zufällige und unabhängige Auswahl, die nicht auf Basis des Rangs, sondern auf den konkreten Fitnessverhältnissen beruht, wird als fitnessproportionale oder Glücksradselektion bezeichnet. Dies kann jedoch bei hoher Dominanz einzelner Individuen schnell zu Crowding führen. Als Alternative wurde das Stochastische Universelle Sampling entwickelt. Dabei ist die Auswahlwahrscheinlichkeit eines Individuums analog zur fitnessproportionalen Selektion. Jedoch werden die μ Individuen nicht in μ Versuchen, sondern nur in einem Versuch ausgewählt. Aus einem Zufallswert werden äquidistant μ Werte im Intervall $[0; 1]$ bestimmt, welche dann die Auswahl der Individuen aus der Fitnessverteilung festlegen.

Eine recht einfach zu implementierende Selektionsmethode ist die Turnierselektion. Dabei werden μ mal k Individuen zufällig aus der Population ausgewählt. Das i-fitteste dieser Individuen wird mit einer Wahrscheinlichkeit $p_t(1-p_t)^{i-1}$ für die nächste Generation selektiert (z.B. in [SBR10] und [ZHA14]).

Wenn ein Pareto-Ansatz für mehrkriterielle Probleme genutzt wird, sind diese Verfahren wenig geeignet, da sie nur für einzelne Fitnesswerte anwendbar sind. Daher gibt es einige Pareto-basierte Selektionsverfahren. Einer der effizientesten und jüngsten Algorithmen dieser Art ist der *Fast Non-Dominated Sorting Genetic Algorithm*, der als NSGA-II bezeichnet wird und von Deb et al. entwickelt wurde [DAPM00] (zu deutsch etwa Genetischer Algorithmus mit nicht-dominierter Sortierung). Dieser Algorithmus wurde für mehrkriterielle RAP z.B. in [LD09, WCTY09, CRNK12] und [AHA15] benutzt.

Dabei werden zunächst die nicht von anderen Lösungen dominierten Individuen aus der Population extrahiert und ihnen der Rang 1 zugewiesen. Dieses Prozedere wird mit aufsteigendem Rangwert so oft wiederholt, bis die Population erschöpft ist. Weiterhin wird für die Individuen ein Distanzwert berechnet, ein Maß für die Entfernung zum (im Sinne des Fitnesstupels) nächsten anderen Individuums selben Ranges. Aus der Sortierung der Individuen aufsteigend nach Rang und, bei gleichem Rang, absteigend nach Distanzwert, werden die ersten μ Individuen selektiert. Dies entspricht einer rangbasierten Selektion.

Kodierung Die meisten der beschriebenen Operationen sind von der Kodierung der Individuen abhängig. Die Kodierung erfolgt dabei oft – in Anlehnung an eine DNA-Sequenz – als Zeichenfolge, z.B. binär oder ganzzahlig. Dies ermöglicht die Anwendung einiger Standardoperationen, wie dem Crossover. Hierbei werden einzelne Elemente der Zeichenfolgen für die Rekombination zweier Individuen ausgetauscht. Welche Elemente ausgetauscht werden, kann rein zufällig bestimmt werden (uniformes Crossover) oder von einer Einteilung

der Zeichenfolge abhängen (N-Punkt-Crossover).

Genetische Algorithmen, die für RAP angepasst werden, basieren oft auf einer ganzzahligen Kodierung. In [CS96a]) wird z.B. eine obere Schranke für die Anzahl an Komponenten in einem Subsystem k ausgenutzt. Eine Kodierung für ein RAP mit n Subsystemen besteht dann aus $n \cdot k$ Genen für alle möglichen k Positionen in den Subsystemen n. Jedes Gen gibt dabei an, welche Komponente in einem Subsystem verwendet wird. Dabei werden die m_i möglichen Komponententypen im Subsystem i nach ihrer Verfügbarkeit absteigend sortiert. Ein Wert $m_i + 1$ gibt an, dass keine Komponente verwendet wurde, falls in einem Subsystem weniger als k Komponenten genutzt werden sollen.

Ein Beispiel für eine Kodierung ($n = 3, m_1 = 3, m_2 = m_3 = 2, k = 3$) wäre dann:

$$(1\,4\,2\,|\,2\,3\,3\,|\,1\,1\,1).$$

Während im ersten Subsystem eine Komponente des verfügbarsten (1) und eine des zweitverfügbarsten Typs (2) ausgewählt werden, wird im zweiten Subsystem nur eine Komponente des Typs mit der zweithöchsten Verfügbarkeit (2) genutzt. Der freie Platz im ersten (4) und die zwei freien Plätze im zweiten Subsystem (3) werden nicht genutzt ($4 = m_1 + 1, 3 = m_2 + 1$). Das dritte Subsystem besteht aus drei Komponenten des Typs mit der höchsten Verfügbarkeit (1).

Alternativ kann wie z.B. in [LD09] eine Kodierung der Länge $\sum_{i=1}^{n} m_i$ gewählt werden und jedes Gen durch eine Zahl aus $\{0, \ldots, k\}$ für die Anzahl der verwendeten Komponenten eines Typs repräsentiert werden. Werden jedoch z.B. mehrere Redundanzstrategien ermöglicht, so kann eine Matrix-Kodierung wie beispielsweise in [CRNK12] oder [AH14] verwendet werden, um der gesteigerten Komplexität zu entsprechen. Operationen können dann auf einzelne Dimensionen der Matrix, wie Zeilen oder Spalten, angewendet werden.

Unabhängig von der Kodierungsart wird für die Rekombinationsmethoden ein uniformes Crossover empfohlen, da dieses Verfahren für kombinatorische Probleme überlegen ist [CS96a]. Dabei werden entweder einzelne Komponenten-Gene oder ganze Subsysteme, abhängig von der Kodierung, getauscht. Bei der Mutationsoperation wird hingegen meist ein einzelnes Komponenten-Gen manipuliert. Die Initialisierung der Lösungskandidaten erfolgt durch die zufällige Auswahl von Komponenten.

2.2.2.2 Tabu-Suche

Der Tabu-Suchalgorithmus wurde erstmals von Kulturel-Konak et al. in [KKSC03] auf ein RAP angepasst und in [KKSN06] auch auf mehrkriterielle RAP erweitert. Die Tabu-Suche basiert dabei auf einer „gierigen“ Nachbarschaftssuche, bei der bestimmte Suchrichtungen durch Tabus eingeschränkt werden. Der Algorithmus für einkriterielle Probleme ist im Pseudo-Code in Algorithmus 3 dargestellt.

Im Gegensatz zum GA und zu den meisten anderen Meta-Heuristiken wird bei der Tabu-Suche (TS) in jeder Iteration nur ein Lösungskandidat betrachtet. Daher kann auch keine Rekombination von Lösungen erfolgen. Stattdessen werden alle möglichen bzw. ausreichend viele Mutationen einer Lösung gebildet, um eine neue Lösung aus der so

Algorithmus 3 Tabu-Suche

```
 1: procedure TS(maxIterations, size, tabuSize)
 2:     tabu ← ∅
 3:     count, lastRestart, iteration ← 0
 4:     candidate ←INITIALIZE
 5:     globalOpt ← candidate
 6:     while count < maxIterations do
 7:         newCandidate ←SEARCHNEIGHBORHOOD(candidate, tabu, globalOpt, size)
 8:         if newCandidate ≥ globalOpt then
 9:             globalOpt ← newCandidate
10:             count ← 0
11:         else
12:             count ← count + 1
13:         end if
14:         tabu ←UPDATETABULIST(candidate, newCandidate, tabuSize)
15:         candidate ← newCandidate
16:     end while
17:     return globalOpt
18: end procedure

19: function SEARCHNEIGHBORHOOD(candidate, tabu, globalOpt, size)
20:     newCandidate ← ∅
21:     while newCandidate = ∅ do
22:         neigborhood ←MOVE(candidate, size)
23:         newCandidate ←SELECTBEST(neighborhood)
24:         if ISTABU(newCandidate) ∧ newCandidate ≤ globalOpt then
25:             newCandidate ←SELECTBESTNONTABU(neighborhood)
26:         end if
27:     end while
28:     return newCandidate
29: end function
```

erstellten Nachbarschaft auszuwählen. Zunächst wird dazu ein zufälliger Lösungskandidat (*candidate*) erstellt, der als globales Optimum (*globalOpt*) gespeichert wird. Nun wird die Nachbarschaft des Kandidaten auf weitere Kandidaten durchsucht (Funktion SEARCHNEIGHBORHOOD).

Dabei werden so genannte Züge (engl. *moves*) durchgeführt. Jeder Zug entspricht einer lokalen Veränderung des Kandidaten, also einer Mutation mit $p_{mut} = 1$. Es werden maximal *size* Züge in einem Schritt durchgeführt. Aus den neu erstellten Kandidaten wird der mit der höchsten Fitness betrachtet. Dieser wird als neuer Kandidat ausgewählt, wenn er nicht einem Tabu entspricht oder das bisherige globale Optimum dominiert. Ansonsten wird der fitteste Nachbar ausgewählt, der keinem Tabu entspricht. Tabus werden auf Basis von früher durchgeführten Zügen definiert und sollen Zyklen im Suchprozess verhindern. Dabei ist es möglich, bestimmte Typen von Zügen oder Teilergebnisse eines Zuges zu tabuisieren.

Nachdem ein neuer Kandidat ausgewählt wurde, wird die Tabuliste aktualisiert (Funk-

Algorithmus 4 Tabu-Suche für mehrkriterielle Optimierungsprobleme

```
 1: procedure MTS(maxIterations, size, tabuSize)
 2:     tabu ← ∅
 3:     count, lastRestart, iteration ← 0
 4:     candidate ←INITIALIZE
 5:     nonDom ← {candidate}
 6:     while count < maxIterations do
 7:         o ←SELECTOBJECTIVE
 8:         newCandidate ←SEARCHNEIGHBORHOOD(candidate, tabu, nonDom, o, size)
 9:         if ISNOTDOMINATED(newCandidate, nonDom) then
10:             nonDom ← nonDom ∪ {newCandidate}
11:             count ← 0
12:         else
13:             count ← count + 1
14:         end if
15:         nonDom ←DELETEDOMINATED(nonDom, newCandidate)
16:         tabu ←UPDATETABULIST(candidate, newCandidate, tabuSize)
17:         candidate ← newCandidate
18:         if count − lastRestart > maxIterations/4 then
19:             lastRestart ← count
20:             candidate ←RANDOMELEMENT(nonDom)
21:             tabu ← ∅
22:         end if
23:     end while
24:     return nonDom
25: end procedure
```

tion UPDATETABULIST). Wird durch das Hinzufügen eines neuen Tabus die definierte Tabulistengröße (*tabuSize*) überschritten, wird das älteste Tabu aus der Liste gelöscht. Ein weiterer Unterschied zu anderen Meta-Heuristiken ist die Zahl der maximalen Iterationen, die in diesem Verfahren variabel ist. So wird das Finden neuer Kandidaten solang wiederholt, bis es für *maxIterations* Iterationen keine Verbesserung des globalen Optimums gab. Dieses wird dann als Ergebnis ausgegeben.

Für die Lösung eines RAP wird in [KKSC03] eine ganzzahlige Kodierung analog zum oben beschriebenen Ansatz von [CS96a] genutzt. Für die Züge kann – im Falle homogener Redundanz – eine Komponente in einem beliebigen Subsystem hinzugefügt oder entfernt werden. Bei heterogener Redundanz gibt es zusätzlich die Möglichkeit eines Zuges, in dem eine ausgewählte Komponente durch einen anderen Typ ersetzt wird. Als Tabu wird in diesem Ansatz das Teilergebnis eines Zuges, also das veränderte Subsystem des neuen Kandidaten, verwendet. Ein möglicher Kandidat, der genau ein solches Subsystem enthält, ist durch ein Tabu betroffen und wird folglich nur als neuer Kandidat ausgewählt, wenn er das globale Optimum dominiert.

Für mehrkriterielle Probleme ist die Tabu-Suche leicht abgewandelt, wie in Algorithmus 4 beobachtet werden kann. Dabei basiert die mehrkriterielle Tabu-Suche auf der einzelnen Betrachtung der verschiedenen Zielwerte. Statt eines einzelnen globalen Optimums wird eine Liste nicht-dominierter Lösungen verwendet. In jeder Iteration muss das

zu betrachtende Kriterium ausgewählt werden. Kulturel-Konak et al. zeigen in [KKSN06], dass die gleichverteilte zufällige Auswahl eines Kriteriums dabei anderen Verfahren überlegen ist.

Die Auswahl eines Zielkriteriums ist bei der Durchforstung der Nachbarschaft entscheidend. Dort wird nämlich die Lösung in Betracht gezogen, die im gewählten Kriterium überlegen ist, ungeachtet der Werte in anderen Kriterien. Ist ein neuer Kandidat ausgewählt, wird nicht nur die Tabuliste, sondern auch die Liste der nicht-dominierten Lösungen aktualisiert. Hier werden dann die Iterationen gezählt, seit denen diese Liste nicht mehr verändert wurde. Um frühzeitige Konvergenz in einem Gebiet des Lösungsraums zu verhindern, wird jedoch nach einem Viertel von $maxIterations$ ohne Veränderung der Liste nicht-dominierten Lösungen der nächste Kandidat zufällig aus dieser Liste ausgewählt. Führt auch dies nicht zu neuen nicht-dominierten Lösungen, wird das Verfahren beendet und die nicht-dominierten Lösungen ausgegeben.

2.2.2.3 Partikelschwarmoptimierung

Die Partikelschwarmoptimierung (PSO) basiert auf der Bewegung von Lösungskandidaten (Partikel) durch den Lösungsraum, die sich ähnlich eines Schwarms an der bisherigen Bewegung sowie an einem lokalen und globalen Gedächtnis orientiert und wurde z.B. in [GS13] auf RAP angewendet. Für einkriterielle Probleme ist die PSO in Algorithmus 5 dargestellt.

Algorithmus 5 Partikelschwarmoptimierung

Require: $c, l, g \geq 0 \wedge c + l + g = 1$
1: **procedure** PSO($size, maxIt, c, l, g$)
2: $swarm \leftarrow$INITIALIZE($size$)
3: EVALUATE($swarm$)
4: $globalOpt \leftarrow \emptyset$
5: $i \leftarrow 0$
6: **for** $i < maxIt$ **do**
7: **for** $p_i \in swarm$ **do**
8: **if** $p_i > p_i$.GETLOCALOPT **then** p_i.SETLOCALOPT(p_i)
9: **end if**
10: **if** $p_i > globalOpt$ **then** $globalOpt \leftarrow p_i$
11: **end if**
12: **end for**
13: **for** $p_i \in swarm$ **do**
14: $r_1, r_2 \leftarrow$ RANDOM(0,1)
15: $p_{i+1} \leftarrow p_i + c(p_i - p_{i-1}) + r_1 l(p_i.\text{GETLOCALOPT} - p_i) + r_2 g(globalOpt - p_i)$
16: **end for**
17: EVALUATE($swarm$)
18: $i \leftarrow i + 1$
19: **end for**
20: **return** $globalOpt$
21: **end procedure**

Zunächst wird ein Schwarm von Lösungen der Größe $size$ zufällig erstellt und eva-

luiert. In jeder Iteration werden aus dem Schwarm das globale Optimum und für jeden einzelnen Partikel die bisherige Position mit der besten Fitness (lokales Optimum) ermittelt. Um die neue Position eines Partikels zu bestimmen, werden die aktuelle und letzte Position sowie das lokale und globale Optimum rekombiniert. Dabei steuern die Parameter c, l und g welchen Einfluss die letzte, die lokal-optimale und die global-optimale Einzelposition auf diese Kombination haben. Der Einfluss der letzteren beiden Positionen wird zusätzlich durch zwei Zufallswerte gesteuert.

Zur Durchführung der Kombination muss also die Differenz $(-)$ zwischen zwei Positionen als Bewegungsvektor sowie die Kombination $(+)$ dieser Vektoren ermittelt werden. Dies ist für Lösungskandidaten im $\mathbb{R}^n$ intuitiv, was die PSO für diese Art Probleme besonders effektiv macht.

Für Probleme, deren Kandidaten nicht als reelle Vektoren dargestellt werden können, kann die Definition der Differenz und Kombination sich schwierig gestalten. In [Yeh14] wird mit der Simplifizierten Schwarmoptimierung (SSO) eine alternative Positionsbestimmung angewendet, die auf einem Zufallswert r basieren. Dabei wird das k-te Merkmal einer neuen Position für die Iteration $i+1$ wie folgt bestimmt:

$$p_{i+1}^k \leftarrow \begin{cases} p_i^k & \text{wenn } r \in [0; c) \\ localOpt_p^k & \text{wenn } r \in [c; c+l) \\ globalOpt^k & \text{wenn } r \in [c+l; c+l+g) \\ random^k & \text{sonst.} \end{cases}$$

Dabei gilt für die SSO $c + l + g < 1$, da so die Erzeugung zufälliger und damit neuer Teilmerkmale ($random^k$) möglich ist. Durch diese Art der Rekombination wird weder die Differenz zwischen zwei Positionen, noch die Kombination dieser Differenzen verlangt.

Für RAP mit mehreren Optimierungszielen kann die mehrkriterielle Partikelschwarmoptimierung (MOPSO von engl. *multi-objective particle swarm optimization*) verwendet werden, wie z.B. in [DZKD15]. Dabei wird statt einem einzelnen globalen Optimum ein Repository aus nicht-dominierten Lösungen gepflegt. Für die Berechnung der neuen Position der Schwarmpartikel wird in jeder Iteration ein zufälliges Element aus dem Repository als globales Optimum verwendet. Statt einem Vergleich der Fitness für die Identifikation des lokalen Optimums wird die Pareto-Dominanz als Kriterium herangezogen.

2.2.2.4 Künstliche Bienenkolonie

Die Künstliche Bienenkolonie (ABC) ahmt die Futtersuche der Honigbiene nach. Bei dieser werden drei Typen von Bienen eingesetzt, um Nahrung zu beschaffen: Arbeiterinnen (engl. *employed bees*), Zuschauer (engl. *onlookers*) und Kundschafter (engl. *scouts*) [YH11]. Arbeiterinnen beuten Nahrungsquellen aus und teilen die Informationen über deren Qualität mittels Tanzbewegungen den Zuschauern mit.

Zuschauerbienen beobachten dabei zahlreiche Arbeiterinnen, bevor sie sich mit hoher Wahrscheinlichkeit für eine der besten Quellen entscheiden. Arbeiterinnen, die keine Quellen ausbeuten, sondern danach suchen, werden als Kundschafter bezeichnet. Finden

sie eine Nahrungsquelle, beginnen sie mit der Ausbeutung. Ist eine Nahrungsquelle vollkommen aufgebraucht, werden alle assoziierten Arbeiterinnen zu Kundschaftern.

In dem ABC-Algorithmus repräsentiert jede mögliche Lösung eine Nahrungsquelle, deren Potenz der Lösungsqualität entspricht. Er ist in Algorithmus 6 im Pseudo-Code dargestellt.

Ein Bienenschwarm der Größe $2 \cdot size$ wird in der ABC in zwei Hälften geteilt: Auf der einen Seite Arbeiterinnen sowie Kundschafter und auf der anderen Seite Zuschauer. Jeder Arbeiterin wird eine zufällige Lösung initial zugeordnet. In jeder Iteration wird für die Arbeiterinlösung eine benachbarte Lösung identifiziert. Für jede Zuschauerbiene wird dann zufällig, aber fitness-proportional, eine Arbeiterin ausgewählt. Für jede zugeordnete Zuschauerbiene wird eine weitere Nachbarlösung für die Arbeiterinlösung konstruiert. Besitzt eine Nachbarlösung eine höhere Fitness als die Arbeiterinlösung, wird diese als neue Lösung für die Arbeiterin gesetzt.

Wird für eine Arbeiterin in *limit* Iterationen keine Nachbarlösung mit höherer Fitness identifiziert, wird die Arbeiterin zum Kundschafter. Dabei wählt sie eine zufällige Lösung als Nahrungsquelle aus. Das Verfahren wird für $maxIt$ Iterationen durchgeführt und das im Suchverlauf identifizierte globale Optimum ausgegeben.

Eine mehrkriterielle ABC wurde bisher nicht auf ein RAP angewendet, jedoch präsentieren die Autoren von [LPG11] eine mehrkriterielle ABC, um Auftragsfertigungen effizient zu planen. Dieser Algorithmus benutzt ein Repository aus nicht-dominierten Lösungen als Ergebnis. Dabei wird die Pareto-Dominanz als Vergleichskriterium herangezogen, ob eine bessere Nachbarlösung gefunden wurde: Dominiert eine Nachbarlösung die Arbeiterinlösung, so wird diese als neue Lösung ausgewählt. Wird eine Nachbarlösung nicht von der Arbeiterinlösung dominiert, besteht eine Wahrscheinlichkeit von 50 %, dass diese Nachbarlösung als neue Lösung ausgewählt wird. Geschieht dies nicht oder dominiert die Lösung der Arbeiterin alle Nachbarlösungen, werden diese verworfen. Nach *limit* Iterationen ohne Änderung der Arbeiterinlösung wird eine zufällige Lösung neu ausgewählt

Statt der fitnessproportionalen Auswahl werden für jeden Zuschauer drei zufällige Arbeiterinnen selektiert. Die nicht-dominierte dieser drei Lösungen wird für den Zuschauer ausgewählt. Sind mehrere Lösungen nicht-dominiert, wird zufällig eine dieser selektiert.

Da die Anwendbarkeit der einkriteriellen ABC auf RAP bereits bewiesen wurde, sollte dieser Algorithmus ebenfalls übertragen werden können. Dabei sollte jedoch dessen Performance im Vergleich zu anderen mehrkriteriellen Lösungsverfahren untersucht werden.

Algorithmus 6 Künstliche Bienenkolonie

```
1: procedure ABC(size, limit, maxIt)
2:     employed ←INITIALIZE(size)
3:     EVALUATE(employed)
4:     globalOpt ← ∅
5:     i ← 0
6:     for i < maxIt do
7:         for bee ∈ employed do
8:             new ← GETNEIGHBOR(bee)
9:             EVALUATE(new)
10:            if new > bee then
11:                bee ← new
12:                bee.count ← 0
13:            else bee.count ← bee.count + 1
14:            end if
15:            if bee > globalOpt then globalOpt ← bee
16:            end if
17:        end for
18:        onlookers ← NEWARRAY(size)
19:        j ← 0
20:        for j < size do
21:            bee ← FITNESSPROPORTIONALSELECTION(employed, 1)
22:            onlookers[bee] ← onlookers[bee] + 1
23:            j ← j + 1
24:        end for
25:        for bee ∈ employed do
26:            j ← 0
27:            neighbors ← ∅
28:            for j < onlookers[bee] do
29:                neighbors ← neighbors ∪ GETNEIGHBOR(bee)
30:                j ← j + 1
31:            end for
32:            EVALUATE(neighbors)
33:            if SELECTBEST(neighbors)> bee then
34:                bee ← SELECTBEST(neighbors)
35:                bee.count ← 0
36:            else bee.count ← bee.count + 1
37:            end if
38:            if bee.count ≥ limit then
39:                bee ← INITIALIZE(1)
40:                EVALUATE(bee)
41:            end if
42:            if bee > globalOpt then globalOpt ← bee
43:            end if
44:        end for
45:        i ← i + 1
46:    end for
47:    return globalOpt
48: end procedure
```

3 Das ITRAP – Ein Redundanz-Allokation-Problem für IT-Dienstleistungen

Die Analyse des Stands der Technik zeigt, dass die Mehrheit der Redundanz-Allokation-Probleme (RAP) auf der Annahme unabhängiger Komponentenfehler aufbauen und die Verfügbarkeit daher kombinatorisch berechnet wird. Die Analyse zeigt aber auch, dass diese Ansätze für die Verfügbarkeitsvorhersage von IT-Services nicht geeignet sind.

In diesem Kapitel werden zunächst Anforderungen für ein RAP für IT-Service-Design aufgestellt. Da keiner der existierenden Ansätze alle diese Anforderungen erfüllt, wird das ITRAP definiert. Außerdem werden Methoden vorgestellt, um die Verfügbarkeit eines IT-Service-Designs nach einer ITRAP-Definition vorherzusagen und auf dieser Basis (sub)optimale Designs zu identifizieren.

3.1 Anforderungsanalyse eines RAP für IT-Service-Design

Die funktionalen und nicht-funktionalen Anforderungen an ein RAP für IT-Services werden in diesem Abschnitt vorgestellt und beschrieben. Diese werden aus der wissenschaftlichen Literatur sowie aus Studien zur Verfügbarkeit von IT-Systemen abgeleitet.

Das Ziel eines hochwertigen IT-Service-Designs aus Sicht des Availability-Managements ist ein optimales Verhältnis aus Ressourceneinsatz und erreichter Verfügbarkeit. Während die Verfügbarkeit voraussetzt, dass ein Service seine Funktion erfüllt, beschreibt der Ressourceneinsatz die Kosten der Maßnahmen, um dies zu erreichen. Dies sind i.d.R. finanzielle Ressourcen, z.B. für die Beschaffung von Hardwarekomponenten und Softwarelizenzen sowie für Mieten, Kühlung, Personal oder externe Services. Dennoch kann es notwendig sein, bestimmte Ressourcen wie Raum oder maximales Gewicht gesondert zu betrachten, da eine Kapazitätserhöhung dieser Ressourcen mit kaum abschätzbaren Kosten verbunden oder gar unmöglich ist.

Um diesen Zielen gerecht werden zu können, werden in Übereinstimmung mit der RAP-Literatur in dieser Arbeit drei Optimierungsprobleme adressiert. Dabei soll ein Service-Design X gefunden werden, sodass

(1) die Verfügbarkeit eines Designs maximiert werden soll, bei Einhaltung einer Kostenbedingung sowie weiteren Bedingungen: $\max A(X)$ unter Einhaltung von $C(X) \leq C$ sowie $L_i(X) \leq L_i$ für alle $0 < i \leq \nu \in \mathbb{N}_0$,

(2) die Kosten eines Designs minimiert werden sollen, bei Einhaltung einer Verfügbarkeitsbedingung sowie weiteren Bedingungen: $\min C(X)$ unter Einhaltung von $A(X) \geq A$ sowie $L_i(X) \leq L_i$ für alle $0 < i \leq \nu \in \mathbb{N}_0$ oder

(3) die Verfügbarkeit eines Designs maximiert werden soll, bei gleichzeitiger Minimierung der Kosten sowie der Einhaltung von Bedingungen: $\min C(X) \wedge \max A(X)$ unter Einhaltung von $L_i(X) \leq L_i$ für alle $0 < i \leq \nu \in \mathbb{N}_0$.

3.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen eines RAP für IT-Service-Design können in folgende Kategorien eingeteilt werden: Identifikation eines (sub)optimalen IT-Service-Designs (F1), Beschreibung möglicher Designs (F2), Vorhersage der Verfügbarkeit eines IT-Service-Designs (F3), Vorhersage der Kosten eines IT-Service-Designs (F4) sowie Vorhersage der Werte für die weiteren Nebenbedingungen (F5).

Identifikation eines (sub)optimalen IT-Service-Designs (F1) Diese Anforderung leitet sich direkt aus den drei oben definierten Optimierungsproblemen ab. Daher kann sie entsprechend der Problemnummer in die Anforderungen F1.1 (Maximierung der Verfügbarkeit), F1.2 (Minimierung der Kosten) und F1.3 (gleichzeitige Optimierung der Verfügbarkeit und Kosten) aufgeschlüsselt werden.

Die Anforderungen F1.1 und F1.2 verlangen dabei die Identifikation eines Designs, das aus einer hinreichend großen Menge an Lösungskandidaten bestimmt wird und diese im Sinne des adressierten Optimierungsproblems (1) oder (2) dominiert. Da es sich um ein NP-schweres Optimierungsproblem handelt, kann die Identifikation des Optimums in vertretbarer Zeit nicht garantiert werden. Daher kann auch ein suboptimales Design als Ergebnis dienen.

Für die Anforderung F1.3 wird das mehrkriterielle Optimierungsproblem adressiert. Da eine a priori-Gewichtung der Kriterien nicht immer möglich oder sinnvoll ist, wird ein Pareto-Ansatz zur Lösung des Problems (3) gefordert. Daher wird als Ergebnis eine Menge von gegenseitig nicht-dominierten Designs erwartet. Auch hier muss die Identifikation einer optimalen Pareto-Front nicht garantiert werden.

Beschreibung möglicher IT-Service-Designs (F2) Für diese Anforderung muss der Suchraum beschrieben werden, der mögliche Lösungen für die definierten Optimierungsprobleme beinhaltet. Dies schließt die Modellierung einer Lösung als System aus Subsystemen (F2.1), möglicher Komponenten und Redundanzmechanismen in einem Subsystem (F2.2) sowie von Administratoren (F2.3) ein. Dabei soll die Anzahl der verwendeten Komponenten bzw. Administratoren in einem Lösungskandidaten nicht beschränkt sein (F2.4).

Das zu definierende System aus Subsystemen beschreibt die Abhängigkeit von der Systemverfügbarkeit von den Komponentenverfügbarkeiten in geeigneter Form. Auf diesen Aspekt wird in der Anforderung F3.2 näher eingegangen. Die Anforderung F2.2 setzt eine Beschreibung von Komponenten sowie derer verfügbarkeitsrelevanten Charakteristi-

ka voraus, ohne die eine quantitative Abwägung von Verfügbarkeit und Kosten eines IT-Service-Designs nicht möglich ist. Zusätzlich müssen für eine Komponente mögliche Redundanzmechanismen definiert werden, da diese Fehlerrate und Betriebskosten der Komponenten beeinflussen können. Die Art und Anzahl von Administratoren hat starken Einfluss auf Verfügbarkeit und Kosten, falls z.B. mehrere Interaktionen parallel ausgeführt werden sollen oder eine Interaktion fehlerhaft war. Aus diesem Grund sollte eine Möglichkeit zur Modellierung von Administratoren in die Definition des Lösungsraumes bereitgestellt werden.

Eine Beschränkung der verwendeten Anzahl von Komponenten bzw. Administratoren sollte nicht per Definition erfolgen, sondern durch die entsprechende Definition der Nebenbedingungen (z.B. Volumen). Durch die Beschränkung der Subsystemgröße könnte ein optimales Design nicht identifizieren werden. Daher soll auf diese Beschränkung verzichtet werden. Dies führt zu einem (theoretisch) unendlich großen Lösungsraum.

Vorhersage der Verfügbarkeit eines IT-Service-Designs (F3) Diese Anforderung beinhaltet sowohl die Abbildung der Verfügbarkeit von Komponenten, z.B. Hardware, Software oder Infrastruktur (F3.1), als auch die Modellierung der Systemverfügbarkeit aus diesen Daten (F3.2). Weiterhin soll der Einfluss der Administratoren auf diese Verfügbarkeiten abgebildet werden (F3.3).

Für die Detaillierung der Anforderung F3.1 können zunächst die in Kapitel 2 behandelten Vorarbeiten herangezogen werden. Dabei sollte zunächst der Zustandsraum einer Komponente modellierbar sein (F3.1.1). Jede Zustandsänderung ist dabei reversibel, um Wiederherstellungsmaßnahmen abzubilden [LD09]. Dies beinhaltet die Definition von Zuständen, die die Leistungsfähigkeit einer Komponente im Sinne der Verfügbarkeit repräsentieren. Für die Abbildung von Leistungsabnahme bedeutet dies, dass auch mehr als zwei Zustände für eine Komponente modellierbar sein sollen [AAMY13].

Eine weitere Anforderung ist die Modellierung verschiedener Fehler- bzw. Wiederherstellungsarten als Zustandsübergänge (F3.1.2). Beispielsweise können für Hardwarekomponenten transiente, intermittierende und permanente Fehler auftreten [BCDGG00]. Erstere sind Fehler, die nur eine kurze Zeit auftreten und ohne äußeren Eingriff wiederhergestellt werden. Intermittierende Fehler treten zunächst mit niedriger Wahrscheinlichkeit auf, während sie nach dem ersten Auftreten in hoher Frequenz die Verfügbarkeit einer Komponente beeinträchtigen. Permanente Fehler führen zum Ersetzen der Komponente.

Für Software-Systeme wurden weitere Fehlerkategorien definiert. So kann man beispielsweise so genannte Bohrbugs, Mandelbugs, Heisenbugs und Altersbugs (engl. *age-related bug*) unterscheiden [GT05]. Bohrbugs sind dabei einfach zu reproduzierende und zu eliminierende Fehler, die eher häufig auftreten. Im Gegensatz dazu haben Mandelbugs oft komplexe Ursachen, die zu einem scheinbar chaotischen Verhalten führen. Dies macht diese Art von Fehlern schwierig zu beheben. Eine spezielle Unterart dieser Fehler sind die Heisenbugs. Analog zur heisenbergschen Unschärferelation lässt sich dabei ein Fehler immer schlechter reproduzieren, je enger man sein Auftreten isoliert. Altersbugs treten erst mit längerer Laufzeit eines Softwareprogramms auf und sind damit dem Phänomen

der Leistungsabnahme zuzuordnen. Ihnen kann mit Konzepten der Software-Verjüngung (engl. *rejuvenation*) begegnet werden [RD07].

Da Fehler- und insbesondere Wiederherstellungszeiten oftmals nicht einer exponentialverteilten Zufallsverteilung entsprechen [CV09], müssen in einem RAP für IT-Service-Design beliebig verteilte Zufallsgrößen abbildbar sein (F3.1.3). Die Parametrisierung dieser Verteilungen ist oftmals schwierig, da Herstellerangaben optimistisch sind [PWB07] oder Testdaten erst statistisch ermittelt werden müssen. Daher muss die Parametrisierung auch unter ungenauen Bedingungen erfolgen können (F3.1.4) [FAM14, GRSV14].

Die modellierten Komponenten stehen in Abhängigkeit, um die Verfügbarkeit des Systems sicherzustellen. Das bereits in Anforderung F2.1 erwähnte System aus Subsystemen bildet die Systemverfügbarkeit auf die Subsystemverfügbarkeiten ab. Diese Abhängigkeit kann seriell oder komplex sein (F3.2.1) [Che06].

Ein Subsystem kann dabei aus einer beliebigen Anzahl funktional äquivalenter Komponenten bestehen, deren Zustandsverteilung die Verfügbarkeit des Subsystems definieren [AAMY13]. Die Zustandsverteilung dieser Komponenten kann jedoch unterschiedlich sein, um heterogene Redundanz abzubilden (F3.2.2) [CS96a]. Weiterhin können diese Komponenten in unterschiedlichen Redundanzmechanismen betrieben werden, die laut Anforderung F2.2 definiert werden sollen. Dabei hängt die Zustandsverteilung der Komponente vom Redundanztyp ab (F3.2.3) [AH14].

Neben Redundanzabhängigkeiten wurden jedoch in Kapitel 2 auch andere Abhängigkeiten wie unzureichende Abdeckung oder Fehler aufgrund gemeinsamer Ursachen identifiziert. Um diese abbilden zu können, müssen generische Abhängigkeiten zwischen den Komponenten modelliert werden (F3.2.4), bei denen Zustandsänderungen einer Komponente auch andere Komponenten beeinflussen können.

Administratoren haben viele Aufgaben in einem IT-System. In Bezug auf die Verfügbarkeit betrifft dies vor allem Wiederherstellungs- und Übernahmemaßnahmen, um Zustandsänderungen von Komponenten zu ermöglichen (F3.3.1) [ZBGD10]. Diese Administratoren sind jedoch in ihrer Anzahl beschränkt, daher muss deren Kapazität und eine Priorisierung der Aufgaben abgebildet werden (F3.3.2) [CV09]. In Studien wie z.B. [OGP03] wurde festgestellt, dass Administrationsfehler eine Hauptursache für Service-Downtimes sind. Daher sollte die fehlerbehaftete Interaktion modelliert werden können (F3.3.3), sodass geplante Zustandsübergänge auch nach einer Interaktion nicht stattfinden können [TY00].

Vorhersage der Kosten eines IT-Service-Designs (F4) Die Kosten eines IT-Services setzen sich aus den kapitalen und operationalen Ausgaben zusammen (vgl. Abschnitt 2.1.4). Die kapitalen Kosten eines IT-Service-Designs sind dabei maßgeblich von den Akquisitions- bzw. Initialkosten der Komponenten geprägt, was auch in allen gesichteten RAP-Definitionen berücksichtigt wird (F4.1). Eventuell zusätzlich anfallende kapitale Ausgaben wie Kosten für die Errichtung von Rechenzentren sind in einer RAP-Definition nicht sinnvoll, da diese i.d.R. fix sind und eher über weitere Nebenbedingungen wie für

Volumen oder Gewicht abgebildet werden sollten.

Die operationale Ausgaben eines IT-Service-Designs setzen sich maßgeblich aus den Betriebskosten für die Komponenten (F4.2), Personalkosten für Administratoren (F4.3) sowie Wiederherstellungskosten (F4.4) zusammen. Die Betriebskosten, insbesondere die Energiekosten, sollten dabei abhängig von der Zustandsverteilung der Komponenten modelliert werden [FWB07]. Auch die Abbildung von Kosten für externe IT-Services oder laufender Lizenzkosten für Software kann über die Betriebskosten durchgeführt werden.

Vorhersage der Werte für die weiteren Nebenbedingungen (F5) Die Optimierung von Verfügbarkeit und Kosten eines IT-Service-Designs kann weiteren Nebenbedingungen unterliegen. Diese können z.B. für Gewicht und Volumen definiert werden. Die Anzahl der Nebenbedingungen sollte beliebig sein (F5.1) und sich diese linear aus den Komponenteneigenschaften aufsummieren lassen (F5.2). Damit können sie alle statischen Eigenschaften eines Designs abbilden, die sich aus der Summe der jeweiligen Eigenschaften der Komponenten ergeben.

Eine Übersicht der funktionalen Anforderungen wird in Tabelle 3.1 präsentiert.

Tabelle 3.1: Funktionale Anforderungen an ein RAP für IT-Service-Design

F Funktionale Anforderungen	
F1 Designoptimierung	**F2** Suchraumbeschreibung
F1.1 Max A s.t. $C \leq C_0$ **F1.2** Min C s.t. $A \geq A_0$ **F1.3** Max $A \wedge$ Min C	**F2.1** System aus Subsystemen **F2.2** Komponenten und Redundanzen **F2.3** Administratoren **F2.4** Keine Beschränkung
F4 Kosten von IT-Services	**F5** Nebenbedingungen
F4.1 Akquisitionskosten **F4.2** Betriebskosten **F4.3** Personalkosten **F4.4** Wiederherstellungskosten	**F5.1** Beliebige Anzahl **F5.2** Lineare Berechnung

Tabelle 3.2: Funktionale Anforderungen an die Verfügbarkeitsvorhersage

F3 Verfügbarkeitsvorhersage		
F3.1 Komponenten	**F3.2** Abhängigkeiten	**F3.3** Administratoren
F3.1.1 Zustandsraum **F3.1.2** Fehlerarten **F3.1.3** Verteilungen **F3.1.4** Ungenauigkeit	**F3.2.1** Komplexes System **F3.2.2** Heterogenität **F3.2.3** Passive Redundanz **F3.2.4** Abhängigkeiten	**F3.3.1** Aufgaben **F3.3.2** Priorität **F3.3.3** Interaktionsfehler

3.1.2 Nicht-Funktionale Anforderungen

Für ein RAP für IT-Service-Design werden außer den funktionalen Anforderungen entsprechend der ISO/IEC 25000 auch Zuverlässigkeit im Sinne der Korrektheit (NF1), Effizienz (NF2), Änderbarkeit (NF3) und Übertragbarkeit im Sinne der Skalierbarkeit (NF4) gefordert [ISO14]. Auf die Anforderung nach Benutzbarkeit wird im Rahmen dieser Arbeit verzichtet, da zunächst die wissenschaftlichen Fragestellungen im Vordergrund stehen. Diese Anforderung sollte jedoch in künftigen Verbesserungen des ITRAP adressiert werden, um die Anwendbarkeit des Ansatzes in der Praxis zu erhöhen. Die nicht-funktionalen Anforderungen sind in Tabelle 3.3 dargestellt.

Tabelle 3.3: Nicht-funktionale Anforderungen an ein RAP für IT-Service-Design

NF Nicht-funktionale Anforderungen
NF1 Korrektheit
NF2 Effizienz
NF3 Änderbarkeit
NF4 Skalierbarkeit

Damit ein Entscheidungsträger sich auf die Ergebnisse des RAP stützen kann, muss die Korrektheit der Ergebnisse gewährleistet sein. Dies bedeutet, dass die in der Problemdefinition gegebenen Zusammenhänge den zu erwartenden Effekt auf die Endergebnisse haben.

Das RAP ist ein NP-schweres Problem und daher nur mit extensivem Zeitaufwand exakt zu lösen. Durch die Einführung von Abhängigkeiten kann die schnelle, kombinatorische Vorhersage der Verfügbarkeit nicht mehr angewendet werden, was den Zeitaufwand zur Lösung weiter erhöht. Daher soll ein RAP für IT-Service-Design effizient gelöst werden, auch wenn dies die Identifikation des optimalen Designs nicht garantieren kann.

Da diese Anforderungen sowie die konkrete Umsetzung dieser nie komplett sein können, muss die Änderbarkeit im Sinne der Anpassbarkeit auf neue oder veränderte Bestandteile des RAP gewährleistet sein. Dies schließt nicht nur die Möglichkeit der Definitionserweiterung ein, sondern auch die Anpassung der Lösungsalgorithmen mit vertretbarem Aufwand.

Außerdem muss ein RAP für IT-Service-Design skalierbar im Sinne der Problemkomplexität sein, da IT-Service-Provider Systemlandschaften mit hunderten Komponenten betreiben können.

3.2 Problemdefinition

Nachdem im vorangegangenen Abschnitt die Anforderungen an ein RAP für IT-Service-Design präsentiert wurden, wird in diesem Abschnitt eine mögliche RAP-Definition präsentiert. Zusammen mit den in den folgenden Abschnitten vorgestellten Lösungsalgorithmen wird das ITRAP gebildet, welches die gestellten Anforderungen erfüllen soll.

Dazu werden zunächst die Annahmen vorgestellt, die für die Aufstellung der Definition nötig sind.

3.2.1 Annahmen

Trotz der Tatsache, dass die Annahme unabhängiger Komponentenausfälle aufgehoben werden kann, sind andere Annahmen notwendig, um das Datenmodell minimal zu halten und die Lösung zu vereinfachen. Dazu gehört, dass die Beziehung zwischen den Zuständen einer Komponente und ihrer Leistungsfähigkeit im Sinne der Verfügbarkeit auf so genannte Performancelevel abgebildet wird. Dieses Vorgehen ist entliehen aus der Analyse der Verfügbarkeit von Stromkreisen, in der es als Ladungsverlust beschrieben wird (engl. *loss of load*).

Um die Betriebskosten eines Designs zu modellieren, können nur die Betriebskosten einzelner Komponenten abhängig von ihrem Zustand abgebildet werden. Damit werden Betriebskosten, die nicht linear von den Komponenten verursacht werden, ignoriert. Ein Beispiel dafür sind die Kühlkosten für IT-Komponenten. Diese Betriebskosten skalieren jedoch mit den Energiekosten der Komponenten (siehe z.B. [PS05]). Daher können sie mit den Energiekosten der Komponenten verrechnet oder später abgeschätzt werden. Eine möglicher Faktor dafür ist die Leistungs-Nutzungs-Effektivität (engl. *power usage effectiveness*) PUE [BCH13].

Administratoren sind homogene Ressourcen, d.h. sie unterscheiden sich weder in den Personalkosten noch in der Fehlerwahrscheinlichkeit. Weiterhin ist eine einzelne Administratorressource ständig verfügbar, repräsentiert also z.B. drei Acht-Stunden-Stellen. Bei einem Administratorfehler muss die geplante Aufgabe wiederholt durchgeführt werden.

Die Betrachtung eines Designs ist zeitabhängig. Zum Zeitpunkt 0 befindet sich dabei jede Komponente in dem durch den korrespondierenden Redundanztyp definierten Initialzustand. Weiterhin existiert ein unendlicher Vorrat an Wiederherstellungsmaßnahmen, sodass eine Komponente nach einem Fehler immer wiederhergestellt werden kann.

3.2.2 Definition eines ITRAP

In diesem Abschnitt werden die zu modellierenden Größen vorgestellt, die ein ITRAP charakterisieren. Ein $ITRAP_{\Delta t,\nu} = (S, demand, wage)$ beschreibt ein RAP für IT-Service-Design mit dem System aus Subsystemen S, der Lastkurve $demand : \mathbb{R}_{\geq 0} \to \mathbb{R}_+$ sowie dem Lohn $wage \in \mathbb{R}$ per Zeitschritt Δt.

Das System aus Subsystemen $S = ((s_1, \ldots, s_n), M)$ besteht aus einer Folge von Subsystemen sowie einer Adjazenzmatrix $M \in \{0;1\}^{(n+2)\times(n+2)}$. Diese Matrix gibt an, welche Subsysteme miteinander verbunden sind sowie welche Subsysteme mit den (virtuellen) Anfangs- und Endknoten verbunden sind. Ist keine Matrix M angegeben, wird von einem seriellen System ausgegangen mit $M = (m_{ij})_{1\leq i,j\leq n+2}$ mit

$$m_{ij} = \begin{cases} 1, & \text{wenn } |j-i| = 1 \\ 0, & \text{sonst.} \end{cases}$$

Ein Subsystem s_i besteht aus einer Menge an Komponenten $\{x_{i1}, \ldots, x_{im}\}$, die in diesem Subsystem verwendet werden können. Eine Komponente x_{ij} lässt sich als Septupel $(Z, per, op, c_{init}, \mathbf{l}, R, T)$ beschreiben mit

- $Z = \{z_0, \ldots, z_{ij}\}$ einer Menge von Zuständen,
- $per : Z \to \mathbb{R}_{\geq 0}$ der Performancefunktion, die jedem Zustand ein Performancelevel zuordnet,
- $op : Z \to \mathbb{R}_{\geq 0}$ der Betriebskostenfunktion, die jeden Zustand auf die Komponenten-Betriebskosten für den Zeitschritt Δt abbildet,
- $c_{init} \in \mathbb{R}$ die Akquisitionskosten,
- $\mathbf{l} \in \mathbb{R}^{\nu}$ die Werte für die Nebenbedingungen,
- R einer Menge von Redundanztypen und
- T einer Menge von Zustandsübergängen.

Ein Redundanztyp $r \in R$ wird als Sextupel $(z, per_{min}, TTA, \omega, \rho, p_{err})$ charakterisiert. Wird in einem Design eine Komponente x_{ij} im Redundanztyp r betrieben, befindet sie sich initial im Zustand $z \in Z_{ij}$. Fällt das Performancelevel des zugehörigen Subsystems auf bzw. unter $per_{min} \in \mathbb{R}_{\geq 0}$, wird die Aktivierung der Komponente angestrebt. Ist der Boolsche Wert ω wahr, so ist dazu Administratorinteraktion notwendig, welcher eine aufsteigende Priorität ρ zugewiesen wird. Mit Wahrscheinlichkeit p_{err} muss die Interaktion wiederholt werden. Die Übernahmezeit ist dabei zufällig und nach TTA (von engl. *time to activation*) verteilt. Nach erfolgreicher Übernahme befindet sich die Komponente im Zustand $\arg\max_{z \in Z} per(z)$.

Die Menge von Zustandsübergängen T wird dabei von den so genannten Transitionen $tr = (z_{start}, z_{end}, TT, \omega, \rho, p_{err}, c, D)$ gebildet. Damit wird ein möglicher Zustandsübergang von z_{start} nach $z_{end} \in Z_{ij}$ beschrieben, der in einer zufälligen Zeit, verteilt nach TT (von engl. *transition time*), stattfindet und Kosten $c \in \mathbb{R}_{\geq 0}$ erzeugt. Auch hier kann Administratorinteraktion mit Priorität ρ und Fehlerwahrscheinlichkeit p_{err} vonnöten sein, wenn $\omega \in \{0; 1\}$ dies anzeigt.

Ein Zustandsübergang kann zu Zustandsveränderungen in anderen Komponenten führen. Dazu können für jede Transition eine Menge an Abhängigkeiten $D = \bigcup(d, x, p, TTE)$ definiert werden. Dabei verändert die Funktion $d : Z_x \to Z_x$ den Zustand einer Komponente x mit Wahrscheinlichkeit $p \in [0; 1]$ nach einer Zeit verteilt nach TTE.

Ist ein $ITRAP$ gegeben, besteht eine zulässige Komponentenselektion im Subsystem $s_i \in S$ aus einer Komponente $x_{ij} \in s_i$ sowie einem Redundanztyp $r \in R_{ij}$ und kann als Tupel $\chi = (x_{ij}, r)$ referenziert werden. Die Menge aller möglichen Komponentenselektionen im Subsystem s_i kann dann als $\Psi_i = \{(x_{ij}, r) | x_{ij} \in s_i, r \in R_{ij}\}$ beschrieben werden.

Eine beliebige (nicht-leere) Sequenz aus Elementen von Ψ_i (auch mit mehrfach auftretenden Elementen) $\psi_i = (\chi_k)_{k \leq m, m \in \mathbb{N}_+}$ mit $\chi \in \Psi_i$ beschreibt dann eine mögliche Konfiguration für das Subsystem s_i. Ein Lösungskandidat oder Design $X = (\psi_1, \ldots, \psi_n, o)$

besteht dann aus Konfigurationen für alle Subsysteme sowie einer Anzahl von Administratoren $o \in \mathbb{N}$.

Somit können die drei Optimierungsprobleme durch die folgenden Parameter adressiert werden:

(1) $\max A(X,t)$ mit $X \in ITRAP_{\Delta t}$ unter Einhaltung von $C(X,t) \leq C$ sowie $L_i(X) \leq L_i$ für alle $0 < i \leq \nu \in \mathbb{N}_0$,

(2) $\min C(X,t)$ mit $X \in ITRAP_{\Delta t}$ unter Einhaltung von $A(X,t) \geq A$ sowie $L_i(X) \leq L_i$ für alle $0 < i \leq \nu \in \mathbb{N}_0$ und

(3) $\min C(X,t) \wedge \max A(X,t)$ mit $X \in ITRAP_{\Delta t}$ unter Einhaltung von $L_i(X) \leq L_i$ für alle $0 < i \leq \nu \in \mathbb{N}_0$.

3.2.3 Zufällige Intervall-Größe

Um Ungenauigkeit in RAP-Definitionen einfließen zu lassen, wird in der Regel die Komponentenzuverlässigkeit als unbestimmte Größe abgebildet. Wie in Kapitel 2 festgestellt, kann dies durch stochastische, Fuzzy-, Fuzzy-Stochastische- und Intervall-Größen erreicht werden. Da die Komponentenverfügbarkeit im ITRAP als Zustandsverteilung abgebildet ist, wird hier der Argumentation von z.B. [WW09] gefolgt, dass die Komponenteneigenschaften durch Zufall **und** Ungewissheit ungenau sind.

Dies betrifft die zufälligen Zustandswechsel, deren Zeit nach den oben beschriebenen Größen TTA und TT verteilt sind. Diese Verteilung kann jedoch in vielen Fällen nicht genau beschrieben werden [MSB12]. Dabei ist oft Wissen über die Art der Verteilung, jedoch nicht über deren genaue Parametrisierung vorhanden. Zufällige Fuzzy-Größen und Fuzzy-Zufallsgrößen können dieses Problem adressieren (vgl. Abschnitt 2.2.1). Ihre Weiterverarbeitung ist jedoch kompliziert, meist werden die Erwartungswerte der Fuzzy-Größen genutzt und auch deren Streuung wie z.B. in [JZMY14].

Um diesen Nachteil auszugleichen, werden Intervall-Größen empfohlen. Diese ermöglichen die isolierte Betrachtung der Intervallgrenzen und damit eine einfache Weiterverarbeitung. Eine Abbildung der Zustandsübergänge auf Intervall-Größen ist jedoch aufgrund der fehlenden Zufälligkeit nicht genügend. Daher wird in dieser Arbeit versucht, die Vorteile der stochastisch-ungenauen Betrachtung der Eingabeparameter mit der Einfachheit der Intervall-Größen zu verbinden. Das Ergebnis dieser Überlegung ist die Zufällige Intervall-Größe.

Eine Zufällige Intervall-Größe $\tilde{I} = (I, a, b, \triangleright)$ beschreibt dabei eine Schar von stetigen Zufallsverteilungen. Dabei bezeichnet $I : \lambda \to (\mathbb{R} \to [0;1])$ eine Zufallsgröße mit Parameter λ. Dieser Parameter bewegt sich im Intervall $[a;b]$ mit $b \geq a$, sodass der Erwartungswert $E(I_\lambda)$ bezogen auf λ monoton steigend bzw. fallend ist. Damit beschreibt $\tilde{I}$ eine Menge möglicher Zufallsgrößen $\{I_\lambda : \lambda \in [a;b]\}$. Zusätzlich ist eine Ordnung $\triangleright$ zu definieren, sodass $I_b \triangleright I_a$ heißt, dass $E(I_b)$ ein optimistischer und $E(I_a)$ ein pessimistischer Wert für eine Zufallsverteilung ist. Ist $a = b$, beschreibt die Zufällige Intervall-Größe eine normale Zufallsgröße.

Als Beispiel kann eine exponentialverteilte Intervall-Zufallsgröße $\tilde{I}$ mit $P(I_\lambda \leq x) = F_\lambda(x) = 1 - \exp^{-\lambda x}$ mit Parameter $\lambda = [a; b]$ definiert werden. Beschreibt diese Zufallsgröße die Zeit zur Wiederherstellung eines Komponentenzustands, wäre eine kürzere Zeit optimistisch und eine längere Zeit pessimistisch. Daher ist in diesem Falle $I_b \triangleright I_a$, da $b \geq a \Rightarrow 1/a \geq 1/b \Rightarrow E(X_a) \geq E(X_b)$, I_b also die optimistische Verteilung wäre.

Durch die Möglichkeit, die Verteilungen TT und TTA als Zufällige Intervall-Größen abzubilden, kann Ungenauigkeit in einem ITRAP modelliert werden. Dies führt jedoch dazu, dass auch die Ergebnisse für die Verfügbarkeits- und Kostenvorhersage sich in einem Intervall befinden. Um diese zu berechnen, ist eine Betrachtung der Intervallgrenzen jedoch ausreichend, da im Gegensatz zu Fuzzy-Größen keine Aussagen über die Zugehörigkeitsfunktion innerhalb eines Intervalls gefordert sind.

Eine Möglichkeit, die Zielwerte abzuschätzen, ist die Durchführung zweier Optimierungsläufe. Dabei werden einmal für alle Verteilungen die optimistische und einmal die pessimistische Parametrisierung verwendet. Die so erhaltenen Zielwerte sind dann die Intervallgrenzen für die realen Zielwerte.

3.2.4 Grafische Modellierung

Für eine übersichtliche Darstellung der nötigen Informationen, um ein ITRAP zu definieren, wird in diesem Abschnitt eine grafische Modellierung für das ITRAP vorgestellt. Um ein ITRAP grafisch zu modellieren, wird ein UML-basierter Ansatz gewählt. Dazu wird die mathematische Definition in ein UML-Klassendiagramm, dargestellt in Abbildung 3.1, übersetzt.

Dabei werden sieben Klassen unterschieden: *ITRAP* kapselt die globalen Variablen sowie die Adjazenzmatrix der Subsysteme und enthält eine Liste dieser Subsysteme. Jedes Subsystem wird dabei von einer Liste von *ComponentType* charakterisiert, die jeweils die Eigenschaften einer Komponente enthalten. Jedem *ComponentType* werden Listen von *Transition* und *RedundancyType* für die möglichen Zustandsübergänge und Redundanzmechanismen zugeordnet. Eine Transition kann zusätzlich Abhängigkeiten hervorrufen, die in der Klasse *Dependency* gekapselt sind. Alle Zeitverteilungen werden als Instanziierungen der Klasse *RandomIntervalVariable* für die Zufällige Intervallgröße definiert.

Durch Nutzung eines UML-Objektdiagramms kann somit ein ITRAP grafisch definiert werden. In Abbildung 3.2 ist ein Beispiel für ein solches Objektdiagramm dargestellt, welches ein ITRAP definiert. Dabei besteht das System aus einem Subsystem, welches mit einer konstanten Last angefragt wird. Für das Subsystem kann eine Komponente ausgewählt werden, die in aktiver oder passiver Redundanz betrieben werden kann. In letzterem Fall ist Administratorinteraktion mit Priorität 1 und 5 % Fehlerwahrscheinlichkeit vonnöten. Sobald ein Administrator zugeordnet wird, ist die Übernahmezeit nach einer Zufälligen Intervall-Größe verteilt, die eine Dreiecksverteilung zwischen den Werten 1 und 5 mit einem Modus im Intervall von 3 bis 4 beschreibt.

Die zu modellierende Komponente besitzt zwei Zustände, *funktionierend* und *defekt*. In ersterem liefert die Komponente volle Leistung, in letzterem keine. Der Energie-

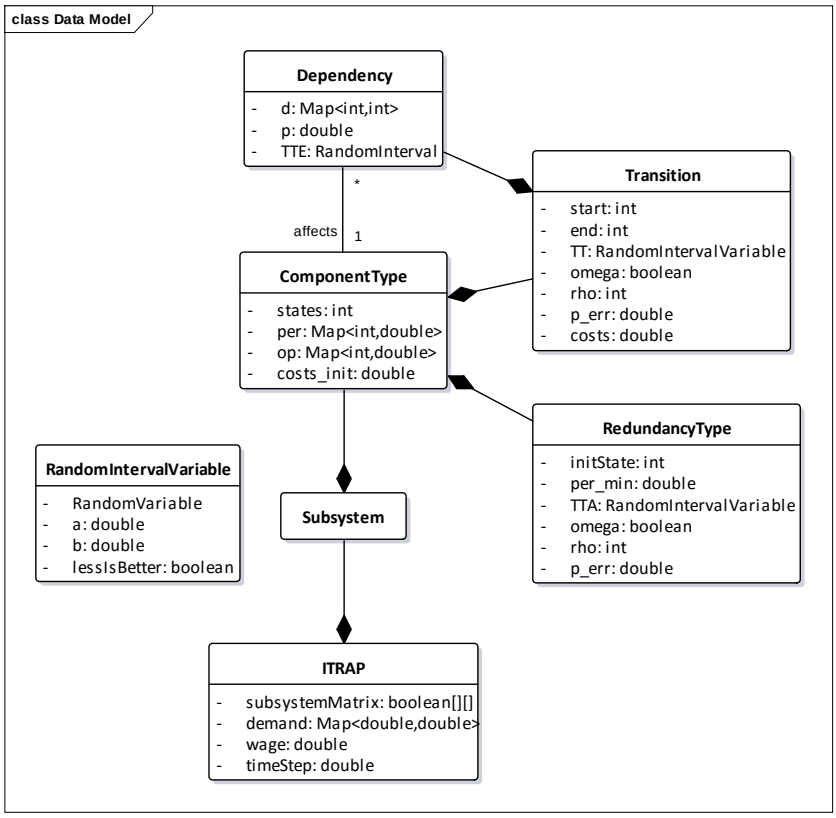

Abbildung 3.1: Meta-Modell für die grafische Modellierung eines ITRAP

verbrauch und damit die Betriebskosten sind im defekten Zustand stark reduziert. Zwei Transitionen, Ausfall und Wiederherstellung, ermöglichen Zustandswechsel. Dabei ist die Ausfallzeit nach einer exponentialen und die Wiederherstellungszeit nach einer normalen Zufälligen Intervall-Größe verteilt. Diese benötigt außerdem Administratorinteraktion, aber mit niedrigerer Priorität als die Übernahme.

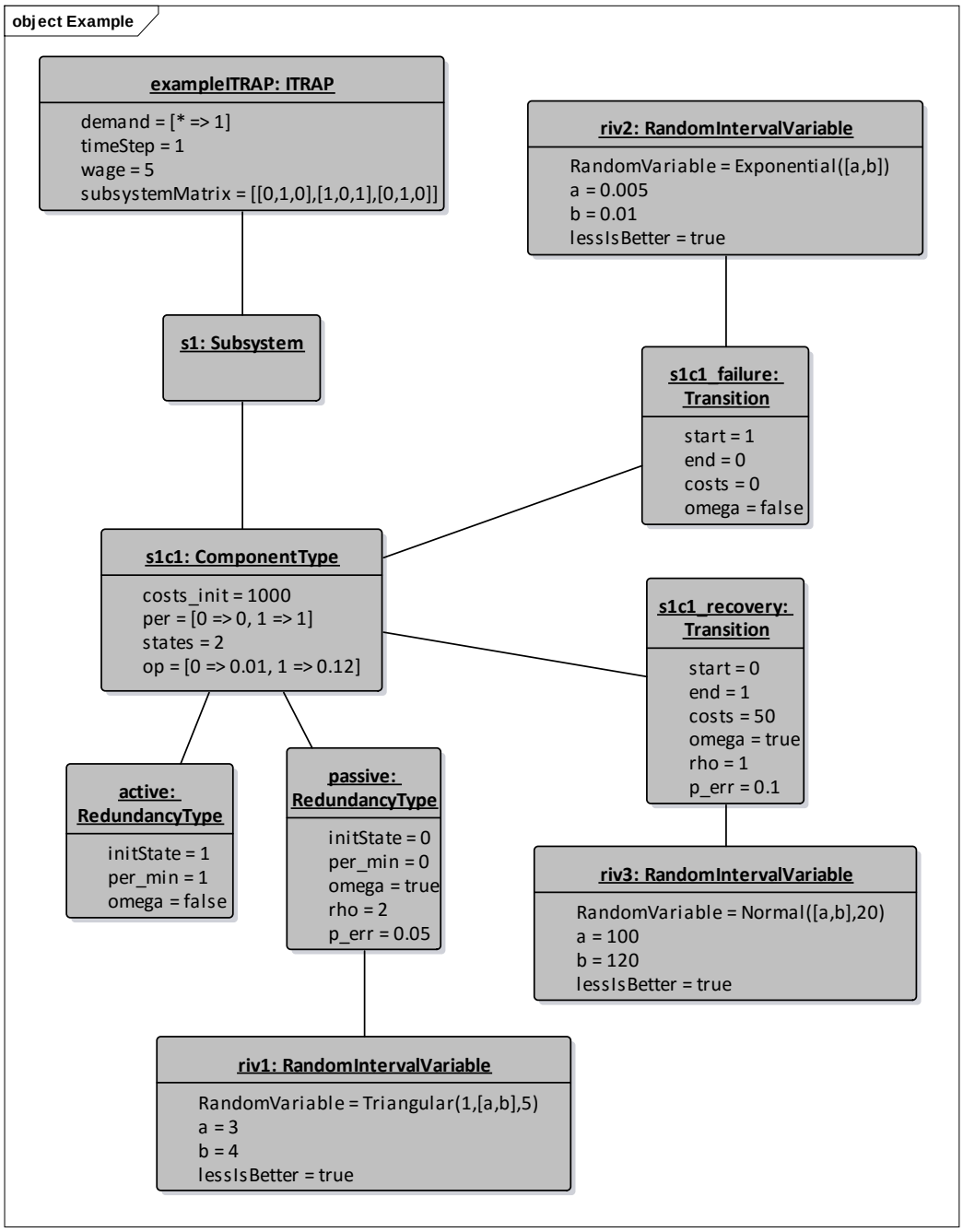

Abbildung 3.2: Beispiel für die Instanziierung des Meta-Modells zur Definition eines ITRAP

3.3 Evaluation eines Designs

Nachdem mögliche IT-Service-Designs in einem ITRAP formal beschrieben werden können, müssen Verfügbarkeit und Kosten dieser Designs vorhergesagt werden. Dazu wird zunächst der Zusammenhang zwischen Serviceverfügbarkeit und der Zustandsverteilungen der Komponenten definiert. Daraufhin wird ein Ansatz basierend auf Petri-Netzen vorgestellt, um diese Zustandsverteilungen abzuschätzen.

3.3.1 Beschreibung der Kosten und Verfügbarkeit eines IT-Service-Designs

Durch die Betrachtung laufender Kosten wie Energie- und Personalkosten steigen die Gesamtkosten eines IT-Service-Designs mit laufender Zeit immer weiter. Daher ist eine Betrachtung des stabilen Zustands für Kosten und Verfügbarkeit nicht zielführend. Wie bereits in der ITRAP-Definition angedeutet, können die drei Optimierungsprobleme daher nur mit Angabe eines Zeitparameters t adressiert werden. Dafür müssen die Intervall-Verfügbarkeit sowie die Kosten, abhängig von der Zustandsverteilung der Komponenten $Z_x(\tau), \tau \in [0, t]$, vorhergesagt werden. Die linearen Nebenbedingungen sind von t unabhängig.

Die Kosten eines Designs X sind dabei die Summe aus Initialkosten, Betriebskosten der Komponenten, Personalkosten sowie Transitionskosten:

$$C(X,t) = C_{init}(X) + C_{op}(X,t) + C_{pers}(X,t) + C_{tr}(X,t)$$

Die Initialkosten ergeben sich linear aus den Akquisitionskosten der in X genutzten Komponenten und sind damit von der betrachteten Zeitspanne unabhängig:

$$C_{init}(X) = \sum_{(x,r)\in X} c_{init_x}$$

Die Betriebskosten können ebenfalls linear aus den Betriebskosten der Komponenten berechnet werden. Diese Kosten sind abhängig von der Zustandsverteilung der Komponenten bis zum Zeitpunkt t und der definierten Betriebskostenfunktion dieser Komponenten:

$$C_{op}(X,t) = \frac{1}{\Delta t} \sum_{(x,r)\in X} \int_0^t op_x(z_x(\tau))\mathrm{d}\tau$$

Für die Personalkosten werden ausschließlich die Kosten der Administratoren betrachtet, die über die Anzahl an Zeitschritten Δt definiert ist:

$$C_{pers}(X,t) = o_X \cdot wage \cdot \frac{t}{\Delta t}$$

Die Transitionskosten sind abhängig von der Anzahl der Transitionen bis zum Zeitpunkt t für die einzelnen Komponenten, die wiederum mit der Zustandsverteilung $z_x(\tau)$ zusam-

menhängt:

$$C_{tr}(X,t) = \sum_{(x,r)\in X} \sum_{tr\in T_\chi} c_{tr} \cdot \#(tr,t)$$

Die Verfügbarkeit eines Designs ist als die Intervall-Verfügbarkeit $A(X,0,t) = A(X,t)$ definiert. Damit gilt:

$$A(X,t) = \frac{1}{t}\int_0^t Z_X(\tau)\mathrm{d}\tau$$

Dabei ist Z_X die Zufallsvariable, die die Verfügbarkeit des Gesamtsystems zu einem Zeitpunkt ausdrückt. Diese wird für einen Zeitpunkt als Verhältnis der aktuellen Service-Performance zu der erwarteten Last berechnet:

$$Z_X(t) = \begin{cases} \frac{per_X(t)}{demand(t)}, \text{ wenn } demand(t) > per_X(t) \\ 1, \text{ sonst} \end{cases}$$

Das Leistungsniveau des Designs im Sinne der Verfügbarkeit wird durch die Funktion $per_X(t)$ repräsentiert. Die Berechnung erfolgt aus den Leistungsniveaus der Einzelkomponenten, jedoch abhängig von der Subsystem-Matrix M. Daher müssen zunächst die möglichen Pfade durch den Subsystem-Graphen bestimmt werden. Dies sind alle Sequenzen von Subsystemen, die den Start- mit dem Endknoten verbinden:

$$\begin{aligned} Paths = \{(s)_{k=1,\ldots,\kappa} : s_k \in S \wedge M(0,s_1) = 1 \wedge M(s_k,s_{k+1}) = 1 \\ \text{für } 1 \leq k < \kappa \wedge M(s_\kappa, n+2) = 1 \wedge s_i \neq s_j \text{ für } i \neq j\} \end{aligned}$$

Die Leistung eines Subsystempfades ist die Performance des leistungsschwächsten Subsystems, da dieses einen Flaschenhals darstellt [AAMY13]. Die Leistungsfunktion ist dann die maximale Pfadleistung über alle möglichen Pfade:

$$per_X(t) = \max_{p\in Paths}\left(\min_{s\in p} per(s,t)\right)$$

Dabei berechnet sich die Performance eines einzelnen Subsystems über die Summe der Komponentenleistungen, da diese parallel betrieben werden [AAMY13]. Diese Größe hängt von der Zustandsverteilung der jeweiligen Komponente ab:

$$per(s,t) = \sum_{(x,r)\in\psi_i} per(z_x(t))$$

Die Werte für die Nebenbedingungen können linear aus den Komponentenwerten errechnet werden:

$$L_i(X) = \sum_{(x,r)\in X} \mathbf{l}_x(i)$$

Mit der Möglichkeit Kosten, Verfügbarkeit und Nebenbedingungen eines Designs zu berechnen, können die Optimierungsprobleme adressiert werden. Um diese Berechnung jedoch durchzuführen, muss die Zustandsverteilungen der im Design genutzten Komponen-

ten geschätzt werden.

3.3.2 Modellierung und Schätzung der Zustandsverteilung mittels Petri-Netzen

Die Zustandsverteilung einer ausgewählten Komponente hängt nicht nur von den definierten Transitionen sowie dem ausgewählten Redundanztyp ab, sondern kann auch von der Anzahl verfügbarer Administratoren (oder deren Fehlern) sowie definierten Abhängigkeiten bestimmt werden. Somit können die Zustandsverteilungen der einzelnen Komponenten nicht unabhängig voneinander betrachtet werden. Dies macht die Bestimmung eines analytischen Ausdrucks für die Zustandsverteilung praktisch unmöglich [LD09].

Wie bereits in Kapitel 2 festgestellt, kann ein zustandsraumbasierter Ansatz genutzt werden, um die Zustandsverteilung zu berechnen. Die explizite Bestimmung der Zustandsverteilung wie z.B. in Markov-Ketten hat jedoch den Nachteil der Zustandsraumexplosion. Aus diesem Grund wird in dieser Arbeit ein Generalisiertes Stochastisches Petri-Netz (GSPN) modelliert und mittels Monte-Carlo-Simulation gelöst (vgl. Abschnitt 2.1.3).

Um die definierten Optimierungsprobleme zu adressieren, müssen die Verfügbarkeiten vieler Designs vorhergesagt werden. Daher muss eine Methode entwickelt werden, Petri-Netze automatisch aus den Designinformationen zu erstellen und zu lösen. Dies wird einmal für die optimistischen und einmal für die pessimistischen Verteilungen durchgeführt. Da einerseits beliebige Verteilungstypen zulässig sind und andererseits unterbrochene Transitionen von z.B. verschiedenen Fehlerarten nicht zurückgesetzt werden sollten, wird *race age* als Rennstrategie aller Ausfalltransitionen gewählt. Für Wiederherstellungs- und Übernahmetransitionen gilt *race enabled*, da diese i.d.R. nach einer Zustandsänderung neu begonnen werden.

Dabei werden zunächst die einzelnen ausgewählten Komponenten x bearbeitet. Für jeden Zustand der Komponente aus Z_x wird eine Stelle erzeugt. Entsprechend der definierten Zustandsübergänge in T_x werden zeitbehaftete Transitionen zwischen diesen Zuständen modelliert. Diesen wird eine Zufallsvariable zugeordnet, die die Zeit zum Zustandsübergang repräsentiert. Für das optimistische Modell wird also eine Zufallsvariable TT_α erzeugt, wenn $TT_\alpha \triangleright TT_\beta$, und TT_β, wenn dies die optimistische Verteilung ist.

Benötigt ein Zustandsübergang Administratorinteraktion, wird eine Stelle generiert, die sich gleichzeitig im Vor- und Nachbereich der entsprechenden Transition befindet (Admin-Stelle). Dies bedeutet, der Zustandsübergang findet nur statt, wenn ein Administrator der Stelle zugewiesen wurde.

Abbildung 3.3 illustriert das Ergebnis dieses Vorgangs an einem Beispiel. Die modellierte Komponente besteht aus drei Zuständen, die jeweils voneinander durch Ausfall-Wiederherstellungs-Zyklen erreichbar sind. Der Übergang von Zustand 0 zu Zustand 2 erfordert dabei Administratorinteraktion.

Für jeden Zustandsübergang tr mit $c_{tr} \neq 0$ wird ein Impulse-Reward eingerichtet, der die Transitionskosten dokumentiert.

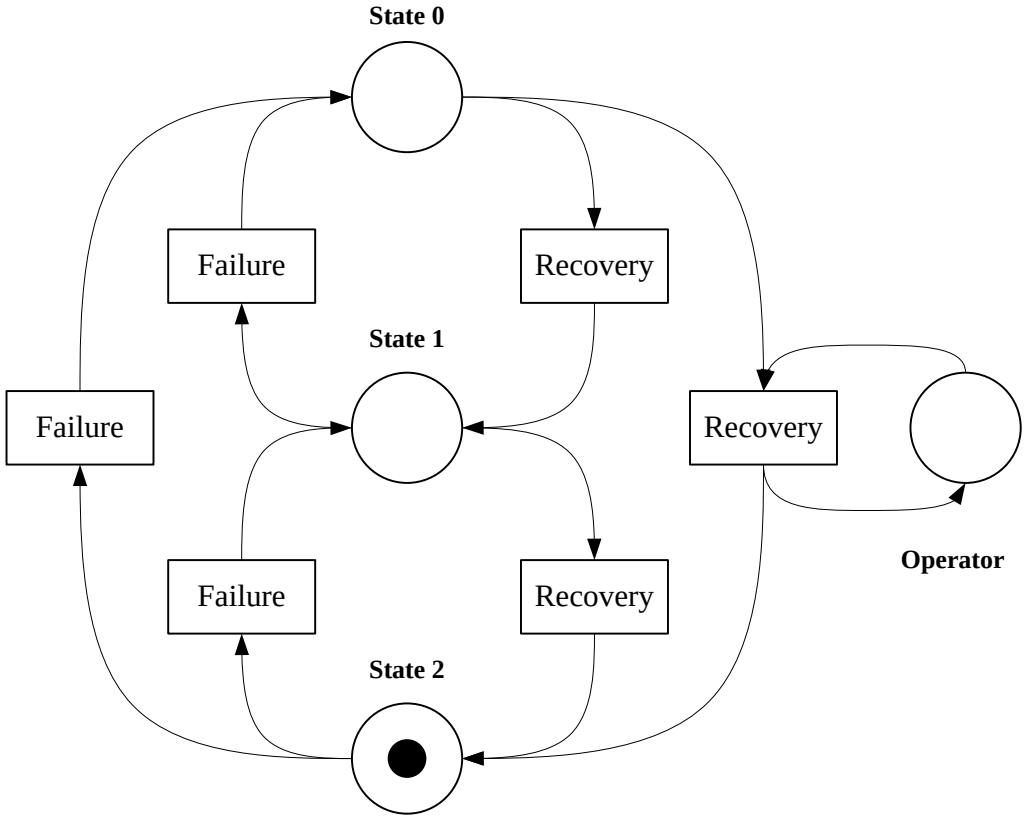

Abbildung 3.3: GSPN-Modell einer einzelnen Komponente

Als nächstes wird der für eine Komponente x ausgewählte Redundanztyp r verarbeitet. Dazu wird eine Marke in der Stelle erzeugt, die den definierten Initialzustand z_r repräsentiert. Ist die Redundanz passiv (Initialzustand ist nicht Zustand höchster Leistung) wird eine Standby-Stelle generiert, die die Wiederherstellung der Komponente aus diesem Zustand in einen Zustand höherer Performance mittels einer Inhibitorkante deaktiviert.

Zwischen dem Initialzustand und dem Zustand höchster Performance wird außerdem eine zeitbehaftete Übernahme-Transition mit der dem Redundanztyp entsprechenden Verteilung TTA platziert, die auch die Standby-Marke vernichtet. Gegebenenfalls kann auch hier eine Administrator-Stelle im Vor- und Nachbereich eingerichtet werden. Zusätzlich wird eine zeitlose Standby-Transition erstellt, die die Komponente wieder in den Initialzustand zurückführt und die Standby-Marke wieder erzeugt, wenn die Performance des Subsystems auch ohne die Standby-Komponente wieder über per_{min} steigt.

Um dieses Verhalten zu gewährleisten, werden der Übernahme- und der Standby-Transition eine Aktivierungsfunktion zugeordnet, die diese Transition genau dann aktiviert, wenn per_{min} im Subsystem erreicht bzw. überschritten wird. Dafür werden alle Zustandskombinationen im Subsystem identifiziert, in denen diese Fälle eintreten.

Ein Beispiel für eine Komponente in Standby-Redundanz ist in Abbildung 3.4 dargestellt. Dabei besitzt die modellierte Komponente zwei Zustände, wobei *State 0* der Initialzustand und *State 1* der Zustand höchster Performance ist. Ist $per(s,t) \leq per_{min}$ wird

die Übernahme-Transition aktiviert, sobald ein Administrator zugewiesen wurde. Ist die Komponente funktionstüchtig und $per(s,t) - per(x) > per_{min}$, wird die Standby-Transition aktiviert.

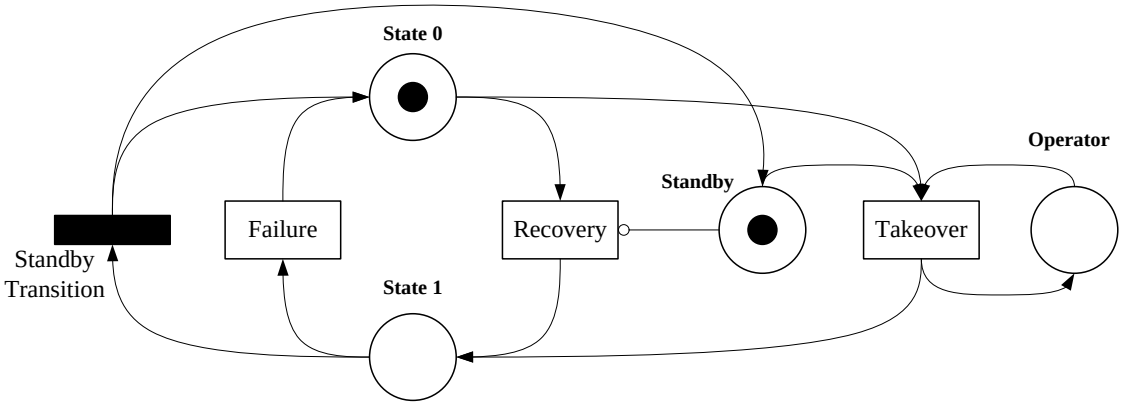

Abbildung 3.4: GSPN-Modell einer Komponente in Standby-Redundanz

Nach der Bearbeitung der Zustände und Redundanzmechanismen werden die Abhängigkeiten prozessiert. Dazu wird für jede Abhängigkeit (d, x, p, TTE) eines Übergangs eine Stelle im Nachbereich der jeweiligen Transition erstellt. Die nach dem Feuern der Transition erstellte Abhängigkeitsmarke kann dann von zwei unmittelbaren Transitionen verbraucht werden. Die erstere feuert mit Wahrscheinlichkeit $1-p$ und hat keine weiteren Verbindungen. Die andere Transition wird mit einem Zustandsübergang in x entsprechend der Funktion d assoziiert und feuert mit Wahrscheinlichkeit p_d. Ist eine Verteilung TTE angegeben, werden im Nachbereich dieser unmittelbaren Transition eine Stelle mit anschließender Transition mit Verteilung TTE erstellt.

In Abbildung 3.5 ist ein Beispiel für die Modellierung einer Abhängigkeit in einem GSPN dargestellt. Dabei kann die Wiederherstellung einer Komponente zum Ausfall einer anderen führen. Dies kann z.B. das Verhalten abbilden, dass die Ausfallwahrscheinlichkeit elektronischer Komponenten nach Wiederherstellung eines Stromausfalls erhöht sein kann.

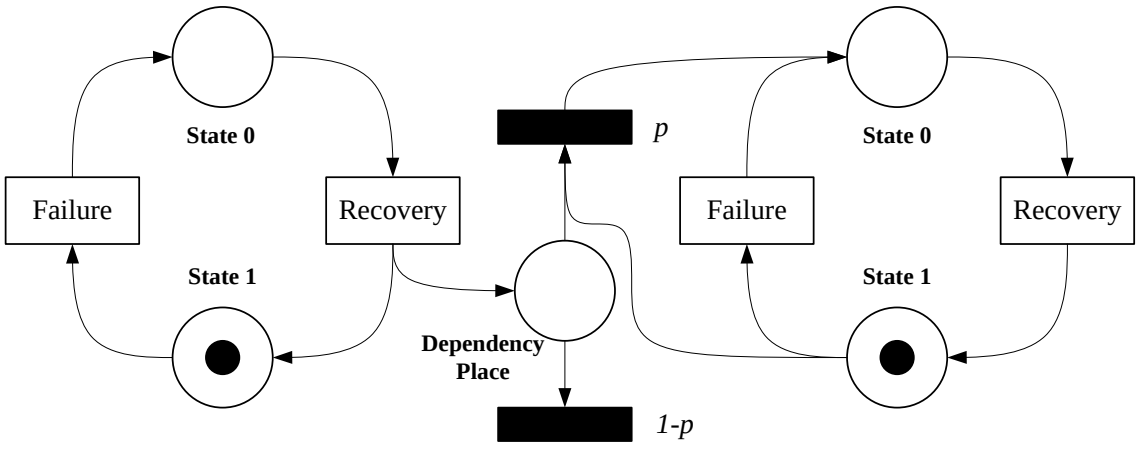

Abbildung 3.5: GSPN-Modell einer Abhängigkeit zwischen zwei Komponenten

Für die Abbildung der Administratorinteraktion und -kapazitäten wird eine Stelle generiert, die als Administratorpool bezeichnet wird. Entsprechend der in einem Design definierten Anzahl an Administratoren o werden Marken in dieser Stelle erzeugt. Zwischen dem Pool und den oben beschriebenen Administratorstellen wird eine unmittelbare Transition modelliert, der eine Aktivierungsfunktion zugeordnet wird. Eine Marke aus dem Pool wird dabei nur dann überführt, wenn die nach Interaktion verlangende Transition durch diese Marke aktiviert und keine Interaktion höherer Priorität verlangt würde.

Im Nachbereich der Administratorstelle werden weiterhin zwei unmittelbare Transitionen erzeugt, die einen erfolgreichen Abschluss der Interaktion sowie einen Interaktionsfehler repräsentieren. Beide unmittelbare Transitionen können nur aktiviert sein, wenn die nach Interaktion verlangende Transition nicht aktiviert ist (das heißt die Interaktion und der damit verbundene Zustandsübergang haben stattgefunden). Dabei feuert die Transition, die die Administratormarke wieder in den Pool überführt (erfolgreiche Interaktion), mit Wahrscheinlichkeit $1 - p_{err}$. Mit Wahrscheinlichkeit p_{err} wird die durch den Administrator gefeuerte Transition wieder zurückgesetzt und aktiviert (Interaktionsfehler führt zu Wiederholung der Interaktion).

Ein Beispiel für diese Modellierung ist in Abbildung 3.6 dargestellt. Dabei ist eine Interaktion eines Administrators aus dem Pool für die Wiederherstellung der Komponente vonnöten. Die Zuordnung des Administrators wird dabei nur vorgenommen, wenn sich die Komponente in *State 0* befindet. Ist die Wiederherstellung einmal abgeschlossen, kann der Zustand unmittelbar zurückgesetzt werden, sodass sie wiederholt werden muss.

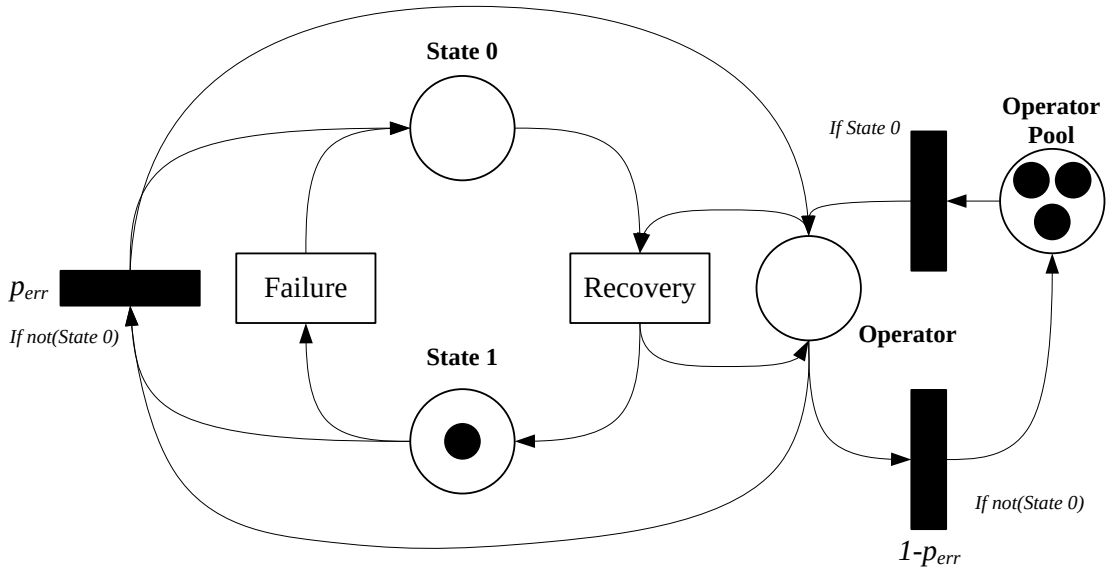

Abbildung 3.6: GSPN-Modell der Administrator-Interaktion

Wird die hier beschriebene Prozedur für alle Komponenten eines Designs durchgeführt, kann die Zustandsentwicklung des Systems simuliert werden, indem fortlaufend Werte für die jeweiligen Zufallsgrößen ermittelt werden. Daraus kann eine Reihenfolge von Ereignissen abgeleitet werden, die den Verlauf der Simulation beschreibt.

Initial- und Personalkosten können unabhängig von der Simulation berechnet werden. Die Wiederherstellungskosten sind in einer globalen Variable der Simulation gespeichert, die durch die definierten Impulse-Rewards verändert wird. Die Berechnung der Betriebskosten erfolgt sequentiell bei jeder Zustandsänderung zum Zeitpunkt bzw. am Ende der Simulation t_1 abhängig vom letzten Zustand zum Zeitpunkt t_0 als Rate-Reward:

$$\begin{aligned} c_{op}(x,0) &= 0 \\ c_{op}(x,t_1) &= c_{op}(x,t_0) + \frac{t_1 - t_0}{\Delta t} op_x(z_x(t_0)) \end{aligned}$$

Analog dazu kann die Intervall-Verfügbarkeit nach jeder Zustandsänderung bzw. am Ende der Simulation berechnet werden:

$$\begin{aligned} A(X,0) &= Z_X(0) \\ A(X,t_1) &= \frac{t_0 \cdot A(X,t_0) + (t_1 - t_0) \cdot \bar{Z}_X(t_0,t_1)}{t_1} \end{aligned}$$

Aufgrund der kontinuierlichen Lastkurve kann sich der Wert für Z_X zwischen t_0 und $t_1 - \epsilon$ ändern. Daher wird die Größe $\bar{Z}_X$ verwendet, die auf der durchschnittlichen Last in einem Intervall,

$$demand(t_0,t_1) = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} demand(\tau) \mathrm{d}\tau,$$

aufbaut. Dieser Wert kann durch verschiedene Approximationsverfahren (z.B. Rechteck-, Trapez- oder Simpsonregel) angenähert werden. Die mittlere Verfügbarkeit im Intervall $[t_0, t_1]$ kann dann wie folgt berechnet werden:

$$\bar{Z}_X(t_0,t_1) = \begin{cases} \frac{per_X(t_0)}{demand(t_0,t_1)}, \text{ wenn } demand(t_0,t_1) > per_X(t_0) \\ 1, \text{ sonst} \end{cases}$$

Mittels der beschriebenen Gleichungen ist es also möglich, die Kosten und die Verfügbarkeit eines IT-Service-Design in einem möglichen Szenario für eine Zeit t zu berechnen. Diese sollten jedoch nicht als Basis für Entscheidungen dienen, da sie nur einen möglichen Ereignisverlauf abbilden. Daher muss die Simulation für verschiedene Zufallswerte so oft wiederholt werden, bis statistisch signifikante Aussagen getroffen werden können (Monte-Carlo-Simulation).

Dazu werden in n Simulationsläufen Werte für Kosten $C^i(X,t)$ und Verfügbarkeit $A^i(X,t)$ berechnet mit $1 \leq i \leq n$. Aus diesen Werten kann jeweils ein α-Konfidenzintervall berechnet werden. Für die Weiterverarbeitung sind dabei Mittelwert und Grenzen des Konfidenzintervalls geeignet. Da für das ITRAP jeweils zwei Konfidenzintervalle für die pessimistischen sowie die optimistischen Werte berechnet werden können, werden im Folgenden nur die Mittelwerte für Verfügbarkeit und Kosten betrachtet. Das Ergebnis einer Evaluierung ist dann eine Intervall-Größe für Kosten sowie Verfügbarkeit des simulierten Designs.

3.4 Lösungsverfahren

Nachdem in den vorangegangenen Abschnitten die Beschreibung des Lösungsraumes sowie die Evaluierung einzelner Kandidaten für das ITRAP erörtert wurde, werden nun geeignete Methoden vorgestellt den Suchraum effizient zu durchforsten und (sub)optimale Lösungen zu identifizieren. Dies baut auf der in Abschnitt 2.2.2 geführten Diskussion auf.

Dort wurde festgestellt, dass Meta-Heuristiken die nötige Flexibilität und Effizienz bieten, um komplexe Probleme zu adressieren. Wie bereits an dieser Stelle diskutiert, werden für die Lösung des ITRAP ein Genetischer Algorithmus, Tabu-Suche sowie eine Bienenkolonie- und zwei Partikelschwarmoptimierungen adaptiert.

Zunächst werden im Folgenden Informationen präsentiert, die für alle Algorithmen von Bedeutung sind, wie die Berechnung der Fitness oder die Kodierung der Individuen. Daraufhin werden die spezifischen Charakteristika der adaptierten Algorithmen vorgestellt.

3.4.1 Fitnessevaluation

Um die Qualität eines Lösungskandidaten im Sinne der adressieren Optimierungsprobleme zu bestimmen, wird eine so genannte Fitnessfunktion verwendet. Diese bildet die Qualität auf einen zu maximierenden Funktionswert (bzw. Tupel im mehrkriteriellen Fall) ab. Die Fitnessfunktion ist dafür entscheidend, welche Ergebnisse produziert werden. Dabei sollten unzulässige Lösungen, in denen Nebenbedingungen verletzt werden, vermieden werden. Dennoch sollte der Bereich unzulässiger Lösungen durchforstet werden, um die frühzeitige Konvergenz zu verhindern [CS96b]. Daher wird eine Anwendung von Straftermen gegenüber Reparaturmechanismen bevorzugt.

Im Allgemeinen lässt sich eine solche Fitnessfunktion wie folgt beschreiben:

$$f(X) = g(X) - \sum_i \delta_i p_i(X)$$

Dabei repräsentiert g die reine Zielfunktion, die für die i Nebenbedingungen N_i ggf. bestraft wird. Die δ_i kontrollieren dabei das Gewicht der einzelnen Strafterme. Dabei sollten nach [CS96a] folgendes gelten:

$$\begin{aligned} N_i(X) \leq N_i \Rightarrow p_i(X) = 0, \\ p_i(X) \sim N_i(X) - N_i \text{ sowie} \\ gen_1 \leq gen_2 \Rightarrow p_i^{gen_1}(X) \leq p_i^{gen_2}(X) \end{aligned}$$

Dies bedeutet, dass der Strafterm bei Erfüllung einer Bedingung 0 ist und sich proportional zum Grad der Verletzung erhöht. Der Strafterm sollte dabei mit steigender Generationenanzahl monoton wachsen. Dies soll garantieren, dass in frühen Generationen unzulässige Lösungen betrachtet werden während dies in späteren Generationen verhindert wird.

Kulturel-Konak et al. nutzen für diesen Strafterm das folgende Schema [KKSC03]:

$$p(X) = \delta \sum_i \left(\frac{\max(0, N_i(X) - N_i)}{NFT_i} \right)^{\kappa_i}$$
$$\text{mit } \delta = g(X_{all}) - g(X_{feas})$$

Dabei wird der Strafwert als Verhältnis des Verletzungsgrads zu einem Grenzwert (NFT von engl. *near feasible threshold*) definiert. Dieses Verhältnis wird mit einem Formfaktor κ potenziert, der in der Literatur meist quadratisch definiert ist [CS96a, KKSC03]. Der NFT hat dabei den Effekt, dass die Lösungen, deren Werte für die Nebenbedingungen um weniger als NFT abweichen, weniger stark bestraft werden (eine Erhöhung von κ verstärkt diesen Effekt). So können Bereiche definiert werden, in denen unzulässige Lösungen auch noch in höheren Generationen betrachtet werden. Die Strafterme für alle Nebenbedingungen werden anschließend mit der Differenz zwischen dem besten Zielwert aller betrachteten ($g(X_{all})$) und dem besten aller zulässigen Lösungen ($g(X_{feas})$) multipliziert, um eine angemessene Gewichtung in der Zielwertdimension zu erhalten.

Diese Straffunktion erfüllt die ersten beiden Kriterien von Coit et al. vollständig (Strafe nur bei Verletzungen, Proportionalität zum Ausmaß der Verletzung), aber das dritte nur in der schwächsten Form (monotone Steigerung der Strafe in höheren Generationen). Daher wird der Strafterm in dieser Arbeit zusätzlich mit der normierten Generationszahl skaliert. Für die drei definierten Optimierungsprobleme (vgl. Abschnitt 3.2.2) werden die folgenden Zielfunktionen genutzt:

$$f^{(1)}(X, t, gen) = A(X, t) - \frac{gen}{maxGen} \cdot (A_{all} - A_{feas}) \cdot p^{(1)}(X, t)$$
$$\text{mit } p^{(1)}(X, t) = \left(\frac{\max(0, C(X, t) - C)}{NFT_C} \right)^2 + \sum_{i=1}^{\nu} \left(\frac{\max(0, L_i(X) - L_i)}{NFT_i} \right)^2$$

$$f^{(2)}(X, t, gen) = -C(X, t) - \frac{gen}{maxGen} \cdot (C_{feas} - C_{all}) \cdot p^{(2)}(X, t)$$
$$\text{mit } p^{(2)}(X, t) = \left(\frac{\max(0, A - A(X, t))}{NFT_A} \right)^2 + \sum_{i=1}^{\nu} \left(\frac{\max(0, L_i(X) - L_i)}{NFT_i} \right)^2$$

$$f_A^{(3)}(X, t, gen) = A(X, t) - \frac{gen}{maxGen} \cdot (A_{all} - A_{feas}) \cdot p^{(3)}(X, t)$$
$$f_C^{(3)}(X, t, gen) = -C(X, t) - \frac{gen}{maxGen} \cdot (C_{feas} - C_{all}) \cdot p^{(3)}(X, t)$$
$$\text{mit } p^{(3)}(X, t) = \sum_{i=1}^{\nu} \left(\frac{\max(0, L_i(X) - L_i)}{NFT_i} \right)^2$$

Für die Verfügbarkeit und die Kosten werden die Mittelwerte $\bar{A}(X, t)$ und $\bar{C}(X, t)$ nach *replications* Simulationsläufen für die Fitnessevaluation gewählt. Daher ist das Ergebnis ein Fitnessintervall mit den Grenzen entsprechend der Fitness für die pessimistische sowie die optimistische Verteilung.

3.4.2 Kodierung

Für die Anwendung der Meta-Heuristiken werden die Lösungskandidaten kodiert, um die zu instanziierenden Operationen einfach und generalisierbar definieren zu können. Dabei sollte die Kodierung so gewählt werden, dass eine Änderung im Genotyp (der Kodierung) auch zu einer vergleichbaren Änderung im Phänotyp (dem Design) führt [PC95].

Wie in Kapitel 2 festgestellt, wurden in bisherigen Arbeiten oft eine obere Schranke für die Anzahl an Komponenten in einem Subsystem genutzt, um eine Kodierung fester Länge zu garantieren (z.B. [CS96a, CRNK12, LD09]). Da für das ITRAP eine solche Schranke nicht definiert ist, um eine Begrenzung des Suchraumes zu vermeiden, ist dieses Verfahren nicht anwendbar.

Alternativ könnte, ähnlich wie z.B. in [ZHA14] oder [LD09], eine Kodierung der Länge $\sum_{i=1}^{n} |\Psi_i|$ genutzt werden. Dabei wird für jede mögliche Komponentenselektion χ angegeben, wie oft sie in der Lösung verwendet wird. Dazu wird die Anzahl der Administratoren angegeben. Sei z.B. $n = 3$, $m_i = 3$ und $|R_{ij}| = 2$, hat die Kodierung die Länge $3{\cdot}3{\cdot}2{+}1 = 19$ für die möglichen Komponentenselektionen sowie die Anzahl an Administratoren und kann so aussehen (Komponenten und Redundanzen sind nur aus Gründen der Übersichtlichkeit dargestellt):

1	1	1	0	1	0	0	0	0	1	1	1	0	0	0	0	0	1	0
o	x_{11}	x_{11}	x_{12}	x_{12}	x_{13}	x_{13}	x_{21}	x_{21}	x_{22}	x_{22}	x_{23}	x_{23}	x_{31}	x_{31}	x_{32}	x_{32}	x_{33}	x_{33}
	r_1	r_2	r_1	r_2	r_1	r_2	r_1	r_2	r_1	r_2	r_1	r_2	r_1	r_2	r_1	r_2	r_1	r_2

Als weitere Möglichkeit kann eine Matrix-Kodierung genutzt werden. Diese besteht aus drei Dimensionen: Subsysteme, Komponenten und Redundanzmechanismen. Die erste Dimension hat die feste Länge $n{+}1$ für die Anzahl an Administratoren sowie die Subsysteme. Die Anzahl von Komponenten ist variabel, aber nur ein Redundanzmechanismus kann pro Komponente genutzt werden. Daher kann ein Design als Matrix von Tupeln aus Komponente und Redundanztypen interpretiert werden, wobei die erste Spalte nur die Anzahl an Administratoren enthält. Dabei repräsentiert der Eintrag (j, k) in der Spalte i die Verwendung von (x_{ij}, r_{ijk}). Das oben verwendet Beispiel wäre dann wie folgt kodiert:

$$\begin{pmatrix} 1 & (1,1) & (2,1) & (3,1) & (1,1) \\ & (1,2) & (3,1) & & (1,1) \\ & (2,2) & (2,2) & & (1,2) \\ & & (2,2) & & \end{pmatrix}$$

Die Kodierungsgröße hängt dabei von den betrachteten Lösungskandidaten ab: Besteht das ITRAP aus vielen Komponenten und Redundanzmechanismen, von denen nur wenige verwendet werden, besteht die erstere Kodierung aus vielen Nullen. Dies ist anders, wenn viele Komponenten genutzt werden und wenig Möglichkeiten bestehen, diese auszuwählen. In diesem Falle wird die letztere Kodierung unhandlich.

Im Folgenden werden die für die Lösungsalgorithmen zu instanziierenden Operatoren für die Matrix-Kodierung definiert. Dies hat den Vorteil, dass die Kodierung den Fokus

auf Subsysteme statt auf einzelne Komponenten legt. Da die Service-Verfügbarkeit direkt aus der Performance der Subsysteme berechnet wird, sollten diese auch in der Kodierung als zusammenhängende Einheiten betrachtet werden. Nichtsdestotrotz ist auch die Anwendung der Vektorkodierung möglich und eine dahingehende Anpassung der Operatoren unproblematisch.

3.4.3 Initialisierung – Zufallssuche

Die zufällige Initialisierung einer Menge von Lösungskandidaten ist Bestandteil jeder genutzten Meta-Heuristik. Die zufällige Erstellung von Kandidaten kann aber auch im Suchverlauf, wie z.B. bei der Bienenkolonieoptimierung, angewendet werden.

Da der Lösungsraum für ein ITRAP theoretisch unbeschränkt ist, ist eine gleichverteilte zufällige Erstellung nicht möglich. Daher sollte die Wahrscheinlichkeit der Erstellung in bestimmte Regionen verzerrt werden, ohne dabei den Erstellungsraum per Definition zu beschränken. Um dies zu erreichen, kann z.B. eine Exponentialverteilung eingesetzt werden. Die Parameter $subSize$ und $opNum$ steuern dabei die Erstellung. Sie beschreiben den Mittelwert einer Exponentialverteilung für die Anzahl von Komponenten in den n Subsystemen und die Anzahl an Administratoren:

$$o \sim \left\lceil \exp(\frac{1}{opNum}) \right\rceil$$
$$|\psi_i| \sim \left\lceil \exp(\frac{1}{subSize}) \right\rceil \text{ für alle } 1 \leq i \leq n$$

Um die Anzahl an Komponenten im Subsystem i zu erreichen, werden gleichverteilt zufällige Kombinationen aus Komponenten und Redundanztypen aus Ψ_i ausgewählt. In der Matrixkodierung des zu erstellenden Kandidaten werden dann die Indizes der ausgewählten Komponente-Redundanztyp-Paare der jeweiligen Subsystem-Spalte hinzugefügt.

Außerdem wird ein Reparaturmechanismus eingeführt, der sicherstellt, dass in jedem Subsystem eine Komponente in aktiver Redundanz betrieben wird. Sollte dies nicht der Fall sein, wird für eine zufällige Komponentenauswahl aktive Redundanz ausgewählt. Dies kann durch Veränderung des Redundanzindizes eines beliebigen Tupels geschehen.

Mit diesem Vorgehen können Lösungskandidaten $X = (o, \psi_1, \ldots, \psi_n)$ erstellt werden.

3.4.4 Mutation – Nachbarschaftssuche

Außer den Schwarmoptimierungen basieren alle genutzten Algorithmen u.a. auf einer Form der Nachbarschaftssuche. Im Genetischen Algorithmus wird dies als Mutation bezeichnet. Dabei wird ein Lösungskandidat in seinen Merkmalen leicht verändert. Die Effekte auf die Fitness eines Individuums sollten dabei überschaubar sein, da große Fitnessveränderungen durch Mutation die Steuerung der Optimierung erschweren.

Für das ITRAP wird die Nachbarschaft eines Lösungskandidaten als die Lösungen definiert, die durch eine einzige Operation erreichbar sind. Gültige Operationen sind dabei die De- bzw. Inkrementierung der Anzahl der Administratoren sowie das Entfernen bzw.

Hinzufügen einer Komponentenauswahl. Dekrementierung und Entfernung sind nur dann möglich, wenn mindestens ein Administrator bzw. mindestens eine Komponentenauswahl in einem Subsystem vorhanden ist.

Dabei kann in der Kodierung eines Kandidaten der Wert für die Anzahl an Administratoren o erhöht bzw. verringert werden, basierend auf einem Zufallswert $r \in [0; 1)$:

$$o_X = \begin{cases} o_X - 1, \text{ wenn } r < 0,5 \wedge o_X > 0 \\ o_X + 1, \text{ sonst} \end{cases}$$

Bei der Entfernung von Komponenten wird ein zufälliges Element der Komponentenmatrix entfernt. Für das Hinzufügen wird ein zufälliges Paar aus Komponente und zulässigem Redundanztyp eingefügt.

3.4.5 Optimierungsalgorithmen

Nachdem in den vorangegangenen Abschnitten die gemeinsam genutzten Operationen der adaptierten Meta-Heuristiken vorgestellt wurden, werden in diesem Abschnitt die spezifischen Eigenschaften der genutzten Algorithmen diskutiert.

Genetischer Algorithmus Der Genetische Algorithmus (GA) verlangt zusätzlich zu den gemeinsamen Operatoren nach Definition einer Rekombinationsoperation sowie der Auswahl einer Selektionsmethode und eines Terminierungskriteriums. Allgemeine Steuerparameter sind μ, λ und p_{mut}.

Zunächst wird mit dem Parameter μ die Populationsgröße angegeben, also wie viele Lösungen gleichzeitig betrachtet werden sollen. Eine Vergrößerung erhöht den Berechnungsaufwand, aber auch die Lösungsqualität. Der Parameter λ gibt an, wie viele Nachkommen pro Generation durch Rekombination erstellt werden. Es gilt i.d.R. $\lambda > \mu$.

Um die λ Nachkommen zu erzeugen, werden zunächst jeweils zwei Individuen aus der Population ausgewählt. Diese Auswahl erfolgt fitnessproportional, sodass Individuen hoher Fitness mehr Nachkommen erzeugen. Im mehrkriteriellen Optimierungsproblem (3) wird dazu die Kostenfitness betrachtet. Sind zwei Individuen ausgewählt, wird ein uniformes Crossover angewandt. In der Matrixkodierung werden dabei ganze Subsystemkonfigurationen bzw. die Anzahl an Administratoren zwischen den Individuen zufällig ausgetauscht.

Nach der Erstellung der λ Nachkommen werden ausgewählte Individuen mutiert. Diese Auswahl wird vom Parameter p_{mut} gesteuert, der die Wahrscheinlichkeit der Mutation für ein Individuum angibt. Aus den ggf. mutierten Kandidaten müssen anschließend μ Individuen für die nächste Generation selektiert werden.

Für die einkriteriellen Optimierungsprobleme (1) und (2) sind prinzipiell alle in Kapitel 2 vorgestellten fitnessbasierten Selektionsmethoden geeignet. In dieser Arbeit wird die Turnierselektion bevorzugt, da sie einerseits nur qualitative Vergleiche zwischen den Individuen erfordert und andererseits aufgrund der zufälligen Auswahl der Turnierteilnehmer einen gewissen Schutz vor Crowding, also der unerwünschten Homogenisierung der Population, bietet. Nachteilig ist dabei die Einführung zweier zusätzlicher Parameter, $tSize$ für

die Turniergröße und p_t für die Auswahlwahrscheinlichkeit.

Das mehrkriterielle Problem (3) wird wie in verwandten Arbeiten (z.B. [CRNK12, LD09, WCTY09] und [AHA15]) durch den Pareto-basierten NSGA-II adressiert. Dieser wählt Individuen nach Dominanzrang und Distanz aus (vgl. [DAPM00]). Für die Anwendung des NSGA-II muss die Plus-Selektion verwendet werden (vgl. Algorithmus 1), um nicht-dominierte Lösungen zu erhalten. Die Probleme (1) und (2) werden durch eine Komma-Selektion mit Elitismus adressiert (vgl. Algorithmus 2). Die Komma-Selektion ermöglicht eine bessere Exploration des Suchraums und verhindert die vorzeitige Konvergenz zu lokalen Optima. Dennoch kann durch Elitismus eine Verschlechterung des Ergebnisses über die Generationen ausgeschlossen werden.

Der Vorgang von Rekombination, Mutation und Selektion wird solange wiederholt, bis die vorgegebene Anzahl an Generationen $maxGen$ erreicht ist. Für die Probleme (1) und (2) wird das Individuum höchster Fitness (da Elitismus betrieben wird, ist dies das globale Optimum) ausgegeben, während die Lösung des Problems (3) eine Pareto-Front, also eine Menge nicht-dominierter Lösungen, darstellt. In Tabelle 3.4 sind die zu instanziierenden Parameter angegeben.

Tabelle 3.4: Zu definierende Parameter für den Genetischen Algorithmus (GA)

Symbol	Anmerkung
μ	Populationsgröße
λ	Anzahl der Nachkommen mit $\lambda \geq \mu$
$maxGen$	Maximale Anzahl an Generationen
p_{mut}	Mutationswahrscheinlichkeit
$tSize$	Turniergröße für (1) und (2)
p_t	Auswahlwahrscheinlichkeit für (1) und (2)

Tabu-Suche Die von Kulturel-Konak et al. adaptierten Tabu-Suchalgorithmen für ein- und mehrkriterielle RAP werden für das ITRAP angepasst. Dazu werden, neben der bereits definierten Nachbarschaftssuche, noch die Definition der Zielwertauswahl sowie der Tabus verlangt.

Die Zielwertauswahl entfällt dabei für die Probleme (1) und (2), da hier der Fitnesswert zum Vergleich der Lösungskandidaten herangezogen wird (vgl. Algorithmus 3). Für das Optimierungsproblem (3) wird wie in [KKSN06] und [KKCB08] die gleichverteilte zufällige Auswahl des Wertes in jeder Iteration der Aggregation in einem Wert sowie der abwechselnden Auswahl vorgezogen (vgl. Algorithmus 4).

Nach der Initialisierung eines zufälligen Lösungskandidaten wird ein Teil von dessen Nachbarschaft (der Größe $neighborhoodSize$) durch Mutationen untersucht. Im Sinne des ausgewählten Fitnesswertes wird der beste Nachbar als neuer Kandidat ausgewählt. Die Differenz zwischen altem und neuem Kandidaten wird in einem Tabu erfasst.

Dieses Tabu ist für das ITRAP als das veränderte Subsystem definiert, analog zu [KKSN06]. Dies bedeutet, dass eine Nachbarlösung nur dann ausgewählt werden sollte, wenn keine ihrer Subsystemkonfigurationen (als Vektor von Komponente-Redundanztyp-

Tupeln) in der Tabu-Liste gespeichert ist oder die Lösung alle anderen dominiert. Dabei werden die ältesten Tabus entfernt, sollte die Liste die Länge $tabuSize$ überschreiten.

Die für die Tabu-Suche zu instanziierenden Steuerparameter sind in Tabelle 3.5 dargestellt.

Tabelle 3.5: Zu definierende Parameter für die Tabu-Suche (TS)

Symbol	Anmerkung
$maxIt$	Maximale Iterationen ohne Verbesserung
$neighborhoodSize$	Anzahl der berücksichtigten Nachbarschaftslösungen
$tabuSize$	Maximale Länge der Tabu-Liste

Partikelschwarmoptimierung Im Rahmen dieser Arbeit werden die Partikelschwarmoptimierung (PSO) sowie die Vereinfachte Schwarmoptimierung (SSO) für das ITRAP adaptiert. Dafür ist die spezielle Rekombination aus aktueller und letzter Position sowie lokalem und globalem Optimum für PSO und SSO zu definieren.

Um die SSO anzuwenden, muss eine Lösung in Teilmerkmale zerlegt werden, die dann ausgetauscht werden können. Für das ITRAP wird eine Lösung bzw. Position in $n + 1$ Merkmale zerlegt. Dabei bestehen n Merkmale aus ausgewählten Komponenten für die jeweiligen Subsysteme sowie ein Merkmal aus der Anzahl an Administratoren. Basierend auf einem Zufallswert und den definierten Schranken c, l und g wird für die neue Position entweder das originale Subsystem, das des lokalen Optimums, das des globalen Optimums oder ein zufälliges Subsystem (vgl. Abschnitt 3.4.3) ausgewählt.

Für die Implementierung der PSO muss eine kompliziertere Rekombination auf Basis der Differenz und der Aufsummierung der Merkmale definiert werden (vgl. Algorithmus 5). Die Differenz sollte dabei dem Abstand zwischen zwei Lösungen entsprechen und wird hier als die Menge an Operationen (vgl. Abschnitt 3.4.4) definiert, die einen Kandidaten in einen anderen überführt. Bei der Rekombination wird dann eine bestimmte Anzahl von zufällig ausgewählten Operationen auf die alte Position für die drei Bewegungsrichtungen angewendet. Diese Anzahl hängt von den definierten Faktoren sowie im Falle der Bewegung zum lokalen und globalen Optimum auch von einem zufälligen Wert ab.

Tabelle 3.6: Zu definierende Parameter für die Partikel- bzw. Vereinfachte Schwarmoptimierung (PSO/SSO)

Symbol	Anmerkung
$maxIt$	Maximale Iterationen
$size$	Anzahl der betrachteten Kandidaten
c	Faktor für bisherige Bewegung
l	Faktor für Bewegung zum lokalen Optimum
g	Faktor für Bewegung zum globalen Optimum
	für PSO $c + l + g = 1$, für SSO $c + l + g < 1$

Für die Matrixkodierung kann eine Differenz als Tupel $X_1 - X_2 = (o_1 - o_2, ((x, r, \pm)_k))$

mit $\pm \in \{+, -\}$ und $0 \geq k \geq K$ definiert werden. Jeder Eintrag $(x, r, \pm)$ besteht dabei aus einer Komponente x sowie einem Redundanztyp r und einem Operator, der angibt, ob eine Komponentenauswahl hinzugefügt $(+)$ oder entfernt $(-)$ werden muss, um den Kandidaten X_2 in den Kandidaten X_1 zu überführen.

Für die zufällige Aufsummierung zweier Differenzen $r_1(X_1 - X_2) + r_2(X_1 - X_3)$ mit $r1, r2 \in [0; 1]$ werden aus jeder Differenz $r_i \cdot (K_i + 1)$ Elemente zufällig selektiert und vereinigt. Diese können dann auf eine Lösung/Position angewendet, um eine neue Position für die nächste Iteration zu berechnen.

Die benötigten Steuerparameter werden in Tabelle 3.6 präsentiert.

Künstliche Bienenkolonie Die Künstliche Bienenkolonie (ABC) basiert vor allem auf der Nachbarschaftssuche, deren Intensität sich nach der Lösungsqualität richtet. Im einkriteriellen Fall wird dazu eine fitnessproportionale Selektion für die Zuschauer angewendet. Wird eine Arbeiterinlösung auch nach *limit* Iterationen nicht durch eine Nachbarlösung abgelöst, wird die Zufallssuche angewendet. Die drei Steuerparameter der ABC sind in Tabelle 3.7 dargestellt (vgl. Algorithmus 6).

Tabelle 3.7: Zu definierende Parameter für die Künstliche Bienenkolonie (ABC)

Symbol	Anmerkung
maxIt	Maximale Iterationen
size	Anzahl von Arbeiterinnen und Zuschauern
limit	Maximale Anzahl an Iterationen ohne Verbesserung eines Kandidaten

4 Evaluierung

Im vorangegangenen Kapitel wurden auf Basis von Anforderungen ein RAP für IT-Service-Design, das ITRAP, definiert sowie Lösungsmöglichkeiten entwickelt. In diesem Kapitel soll dieses Artefakt evaluiert werden. Dazu werden ein entwickelter Prototyp sowie dessen Verifikation als Proof-of-Concept anhand der in Abschnitt 3.1 aufgestellten Anforderungen präsentiert. Anschließend wird der Prototyp in einem realen Anwendungsfall eingesetzt, um das IT-Service-Design eines Application Service Provider zu optimieren. Somit können die verschiedenen Lösungsalgorithmen verglichen und der Unterschied zu klassischen RAP herausgestellt werden. Dies demonstriert die Anwendbarkeit des ITRAP und damit die Erreichung der Ziele der Arbeit.

4.1 Prototyp und Verifikation

In diesem Abschnitt wird zunächst der Prototyp beschrieben, der das ITRAP praktisch umsetzt. Auf Basis dieser Implementierung kann die Erfüllung der Anforderungen analysiert und auf Korrektheit geprüft werden. Damit wird die Umsetzung eines RAP für IT-Service-Design verifiziert.

4.1.1 Prototyp

Der Prototyp eines Programms zur Definition und Lösung eines ITRAP ist in der objektorientierten Programmiersprache *Java* geschrieben und in das Simulationsframework *AnyLogic University 6.8.1* von *XJ Technologies*[1] eingebettet. Dieses bietet Funktionen zur nebenläufigen Ereignisabarbeitung mit Zeitstempeln sowie für die Einbindung von externem Code. Daher ist das Framework für die Umsetzung der Petri-Netz-Simulation geeignet.

Der Prototyp wird durch die beiden Komponenten *Simulation* und *EvolAlg* gebildet, die den AnyLogic-internen sowie externen Code beschreiben. Die beiden Komponenten sind in Abbildung 4.1 als UML-Komponentendiagramm dargestellt. Dabei enthält die Komponente *EvolAlg* die Implementierung der Meta-Heuristiken sowie die ITRAP-Definition. In der Komponente *Simulation* befinden sich die Java-Klassen, die für die simulationsbasierte Auswertung eines Lösungskandidaten benötigt werden.

Komponente *EvolAlg* Für die Implementierung der Meta-Heuristiken konnten die Gemeinsamkeiten der ausgewählten Algorithmen ausgenutzt werden. Dazu wurden Schnitt-

[1] `http://www.anylogic.com/`

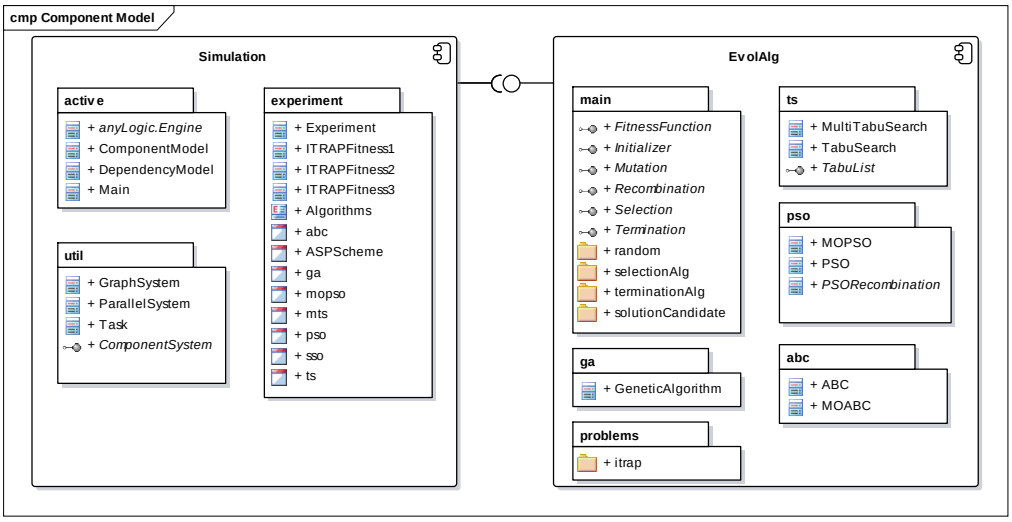

Abbildung 4.1: UML-Komponentendiagramm des Prototypen

stellen erstellt, die von allen implementierten Lösungsmethoden genutzt werden. Diese sind im Paket *main* gesammelt, welches in Abbildung 4.2 als UML-Klassendiagramm dargestellt ist. Aus Gründen der Übersichtlichkeit wurden Klassen, Schnittstellen und Objekte unterschiedlich eingefärbt.

Die beiden Schnittstellen *SolutionCandidate* und *SolutionCandidateMulti* beschreiben dabei Lösungskandidaten. Für die einkriterielle Optimierung enthält *SolutionCandidate* daher eine zu maximierende Zielfunktion (*getFitness()*) und eine Zähl-Hilfsvariable

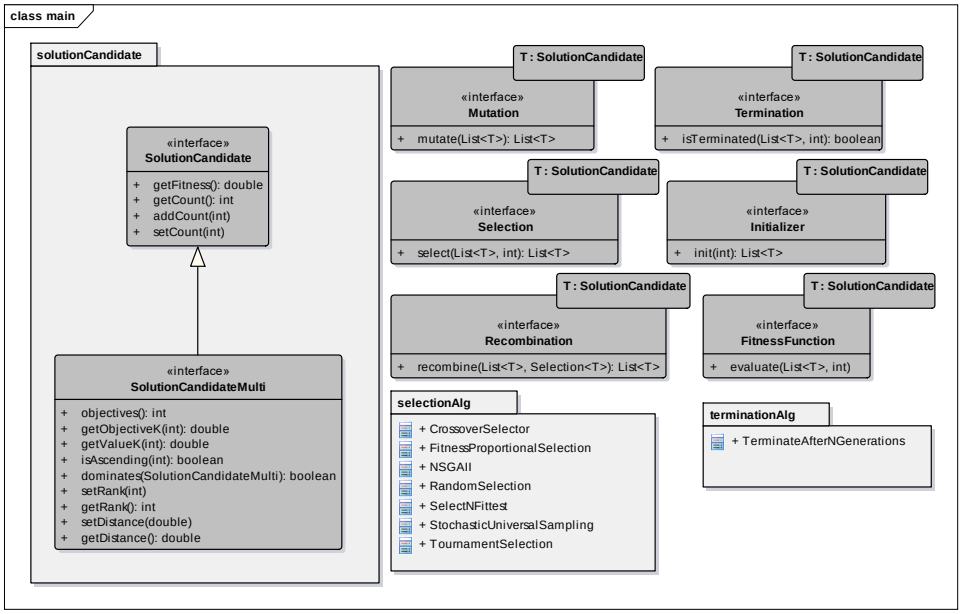

Abbildung 4.2: UML-Klassendiagramm des Pakets *main* in der Komponente *EvolAlg*

(*count*). Bei der mehrkriteriellen Optimierung werden zusätzlich Hilfsvariablen für Rang (*rank*) und Distanz (*distance*) gekapselt. Die Kernmethoden dieser Schnittstelle sind jedoch Funktionen zur Ausgabe der verschiedenen Zielwerte (*getObjectiveK(int)*) sowie deren Anzahl (*objectives()*) und die Vergleichsmethode *dominates(another)*.

Die Interfaces des Paketes *main* repräsentieren die Grundoperatoren jeder ausgewählten Meta-Heuristik: Initialisierung (*Initializer*), Bewertung (*FitnessFunction*), Nachbarschaftssuche (*Mutation*), Rekombination (*Recombination*), Selektionsmethoden (*Selection*) sowie Terminierungsbedingungen (*Termination*). Für letztere ist standardmäßig die Terminierung nach *maxGen* Generationen bzw. nach *maxIt* Iterationen vorgesehen.

Da die fitnessbasierte Selektion in der Regel problemunabhängig ist, wurden im Paket *main.selectionAlg* mehrere Selektionsmethoden umgesetzt. Darunter befinden unter anderem sich die Turnierselektion (*TournamentSelection*), die fitnessproportionale Selektion (*FitnessProportionalSelection*) und der NSGA-II (*NSGAII*).

Auf Basis dieser Schnittstellen konnten die Lösungsalgorithmen in den Paketen *ga*, *ts*, *pso* und *abc* implementiert werden. Der Genetische Algorithmus im Paket *ga* ist in Abbildung 4.3 als UML-Klassendiagramm dargestellt. Die Klasse *GeneticAlgorithm* implementiert dabei die Plus-Selektion, die Komma-Selektion sowie die Komma-Selektion mit n-Elitismus (vgl. Algorithmen 1 und 2). Um ein Objekt dieser Klasse zu erzeugen, müssen problemspezifische Instanziierungen aller Schnittstellen des Paketes *main* übergeben werden.

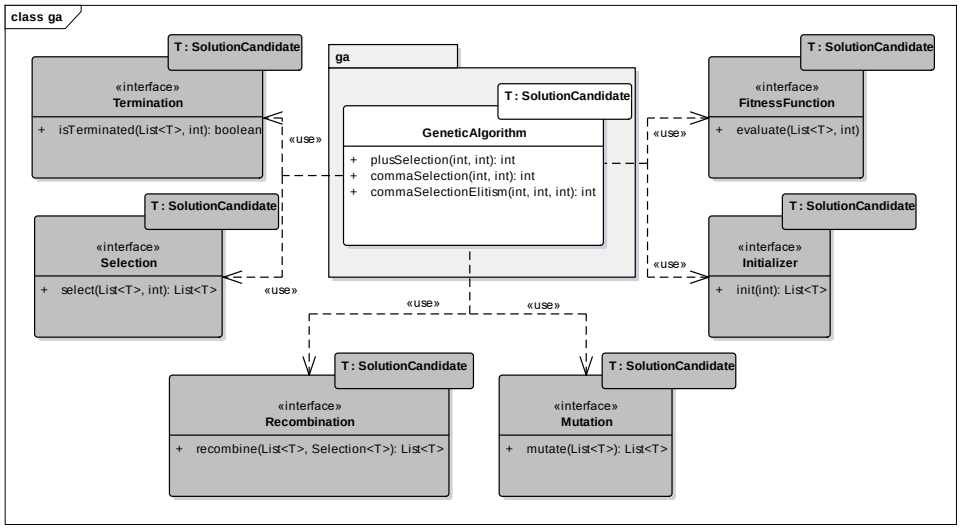

Abbildung 4.3: UML-Klassendiagramm des Paketes *ga* in der Komponente *EvolAlg*

Für die Tabu-Suche im Paket *ts* sind die unterschiedlichen Algorithmen für einkriterielle (*TabuSearch*) sowie mehrkriterielle Tabu-Suche (*MultiTabuSearch*) implementiert (vgl. Algorithmen 3 und 4). Dabei setzt das Mutationsinterface die Funktion MOVE aus der Funktion SEARCHNEIGHBORHOOD für die Tabu-Suche um. Für die Instanziierung der Funktionen ISTABU und UPDATETABULIST muss das Interface *TabuList* umgesetzt wer-

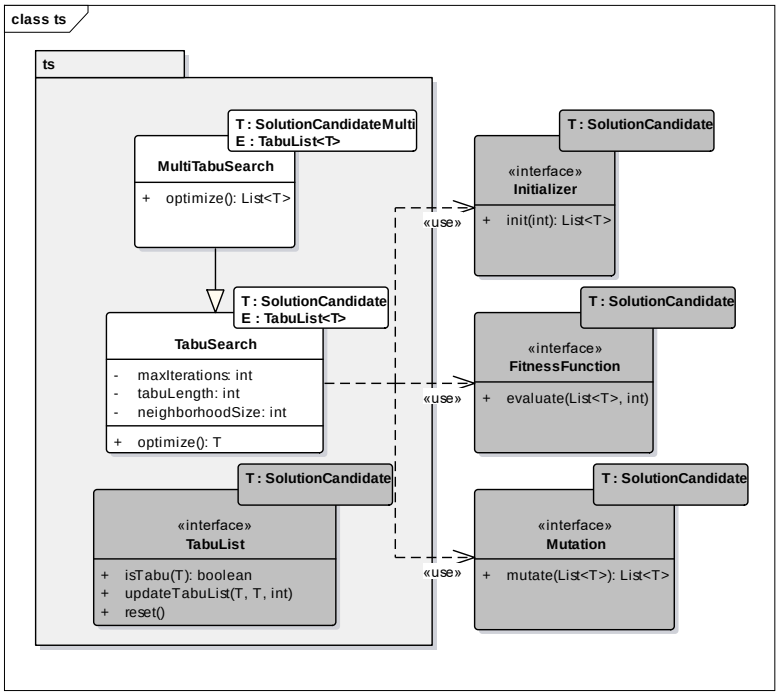

Abbildung 4.4: UML-Klassendiagramm des Paketes *ts* in der Komponente *EvolAlg*

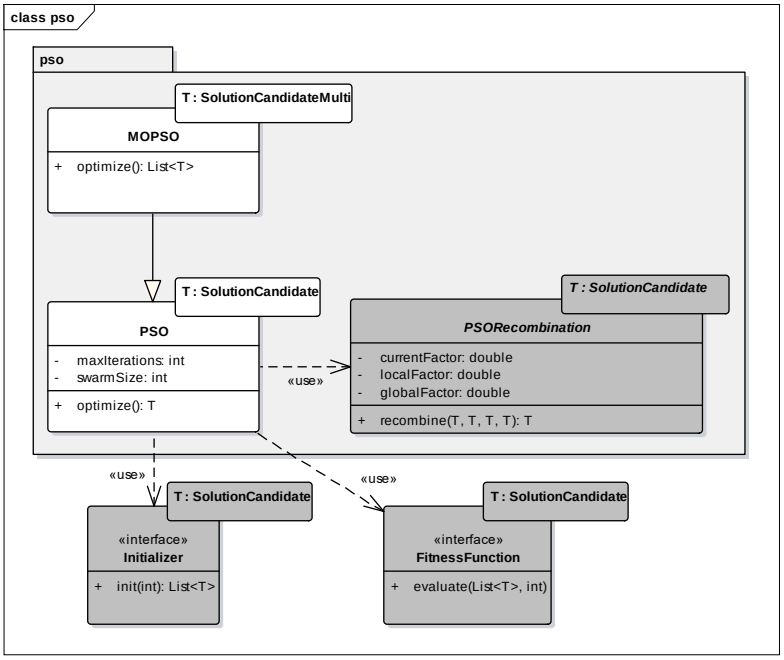

Abbildung 4.5: UML-Klassendiagramm des Paketes *pso* in der Komponente *EvolAlg*

den. Dieses kapselt die Methoden *isTabu(candidate)* zur Prüfung eines Lösungskandidaten und *updateTabuList(current, next, iteration)* für die Aktualisierung der Tabu-Liste. Das Paket ist in Abbildung 4.4 in einem UML-Klassendiagramm visualisiert.

Die Partikelschwarmoptimierung ist im Paket *pso* umgesetzt. Auch hier werden Klassen für die einkriterielle (*PSO*) und mehrkriterielle Optimierung (*MOPSO*) unterschieden. Für die Initialisierung und Evaluation des Schwarms werden Interfaces aus dem Paket *main* genutzt.

Das Herzstück der Partikelschwarmoptimierung, die Bewegung der Lösungspartikel durch den Suchraum, ist in der abstrakten Klasse *PSORecombination* implementiert. Diese Klasse wird problemabhängig instanziiert, wobei die drei Faktoren für die Gewichtung der Bewegungsrichtungen c, l und g übergeben werden. Die Methode *recombine(current, last, local, global)* errechnet dabei die neue Position eines Partikels *current* und kann sowohl für eine PSO als auch für eine SSO instanziiert werden. Das UML-Klassendiagramm für das Paket *pso* ist in Abbildung 4.5 dargestellt.

Die Umsetzung der Künstlichen Bienenkolonie ist im Paket *abc* beschrieben. Dort werden ebenfalls Klassen für die einkriterielle (*ABC*) und mehrkriterielle Optimierung (*MOABC*) unterschieden. Neben der Fitness- und Initialisierungfunktion wird ebenso auf das Interface *Mutation* für die Funktion GETNEIGHBOR aus Algorithmus 6 Bezug genommen. Zusätzlich wird eine Selektion übergeben, um die Arbeiterinnenauswahl der Zuschauer zu realisieren. In Abbildung 4.6 ist das Paket als UML-Klassendiagramm dargestellt.

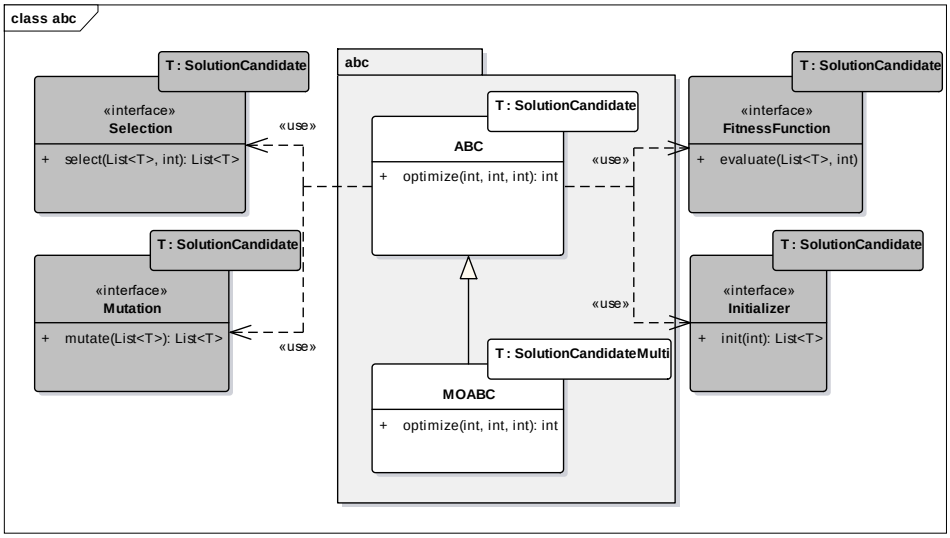

Abbildung 4.6: UML-Klassendiagramm des Paketes *abc* in der Komponente *EvolAlg*

Im Paket *random* wird das Konzept der Zufälligen Intervallgröße implementiert. Dieses ist in der Klasse *RandomInterval* umgesetzt, die für verschiedene Verteilungsarten (*DistributionType*) konstruiert werden kann. Für jede Verteilungsart ist dabei ein Parameter als Intervallgröße definiert.

Dabei gilt

- für eine deterministische Verteilung $DETERMINISTIC(\boldsymbol{x}) : \boldsymbol{x} = [a, b]$,
- für eine Exponentialverteilung $EXPONENTIAL(\boldsymbol{\lambda}) : \boldsymbol{\lambda} = [a, b]$,
- für eine Normalverteilung $NORMAL(\boldsymbol{\mu}, \sigma) : \boldsymbol{\mu} = [a, b]$,
- für eine Dreiecksverteilung $TRIANGULAR(x, \boldsymbol{y}, z) : \boldsymbol{y} = [a, b]$ mit $a > x \wedge b < z$ sowie
- für eine Gleichverteilung $UNIFORM(x, \boldsymbol{y}) : \boldsymbol{y} = [a, b]$ mit $a > x$.

Basierend auf dem Verteilungstyp, dem Intervall und dem Vergleichsoperator (durch *lessIsOptimistic*) können sowohl für einen optimistischen, als auch für einen pessimistischen Durchlauf Verteilungen (*RandomDistribution*) generiert werden, aus denen wiederum die entsprechende Elemente in der Simulationsumgebung erstellt werden. Das Paket *random* ist in Abbildung 4.7 als UML-Klassendiagramm dargestellt.

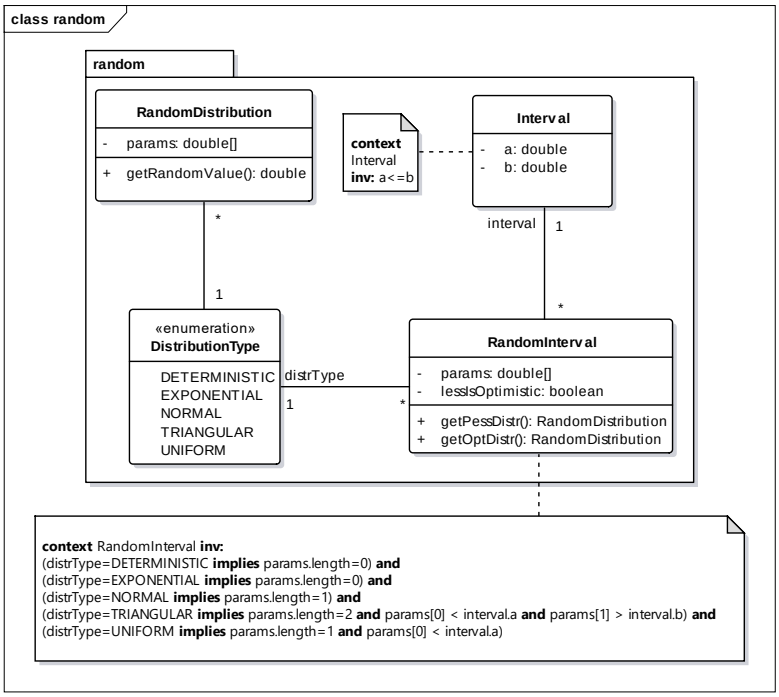

Abbildung 4.7: UML-Klassendiagramm des Paketes *main.random* in der Komponente *EvolAlg*

Das Paket *problems* enthält problemspezifische Kodierungen und Operatoren. So enthält das Paket *problems.itrap* alle relevanten Informationen zur Definition und Lösung eines ITRAP. Dieses Paket ist in Abbildung 4.8 als UML-Klassendiagramm visualisiert.

Die Klasse *ITRAPScheme* definiert dabei ein ITRAP, u.a. durch die Definition eines Subsystemgraphen (*adjacency*) und den Nebenbedingungen. Weiterhin wird jedem ITRAP

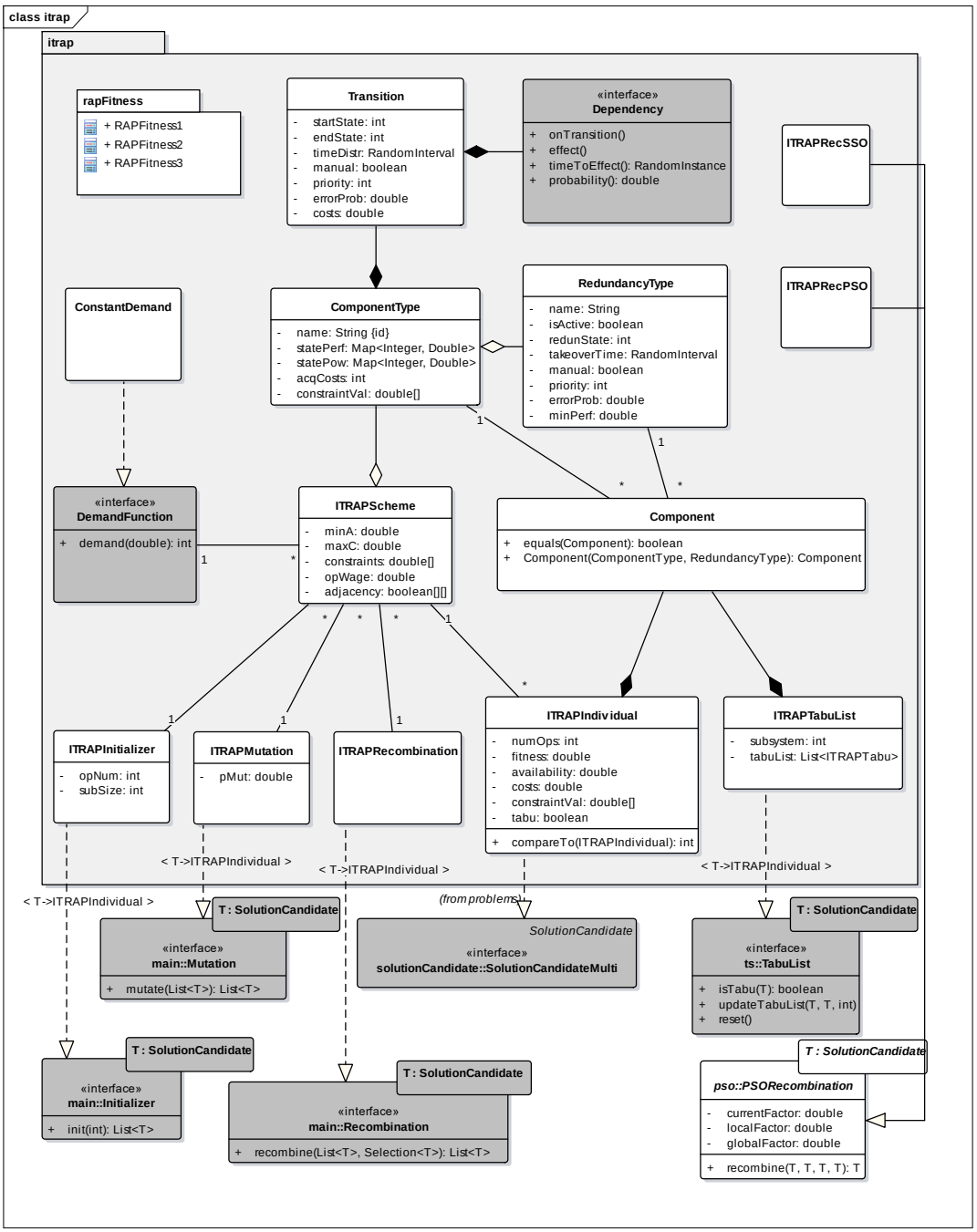

Abbildung 4.8: UML-Klassendiagramm des Paketes *problems.itrap* in der Komponente *EvolAlg*

eine Klasse zugeordnet, die das Interface *DemandFunction* implementiert. Diese Klasse gibt dann die Last eines Services zu einem Zeitpunkt an. Als einfachste Umsetzung dieser Schnittstelle repräsentiert die Klasse *ConstantDemand* eine konstante Last.

Die einzelnen Subsysteme eines ITRAP sind als Listen von Instanziierungen der Klasse *ComponentType* definiert. Jedem Objekt der Klasse *ComponentType* sind dabei insbesondere eine Betriebskosten- und Performancefunktion (*stateOp* und *statePerf*) sowie eine Menge von Instanziierungen von *Transition* und *RedundancyType* zugeordnet. Die Klasse *Transition* beschreibt mögliche Zustandsübergänge zwischen *startState* und *endState*. Instanziierungen von *RedundancyType* definieren mögliche Redundanzmodi für die Komponente.

Zu jeder Transition kann eine Menge von *Dependency*-Instanziierungen assoziiert werden. Dabei wird eine Abhängigkeit durch Funktionen für den direkten Effekt von Zustandsübergängen (*onTransition()*) und für einen Effekt nach einer Zeitspanne (*effect()*) sowie durch die Beschreibung der Dauer dieser Zeitspanne (*timeToEffect()*) und durch die Eintrittswahrscheinlichkeit der Abhängigkeit (*probability()*) charakterisiert.

Diese Klassen setzen somit das Meta-Modell für ITRAP aus Abschnitt 3.2.4 um. Die weiteren Klassen definieren Kodierung und Operatoren für die Lösung eines ITRAP. Die Kodierung repräsentiert dabei die Klasse *ITRAPIndividual*, die die Matrixkodierung abhängig von einem *ITRAPScheme* umsetzt. Neben der Anzahl an Administratoren (*numOps*) enthält diese Klasse also eine Liste aus Listen von Komponentenselektionen (*Component*), die aus Tupeln von *ComponentType* und *RedundancyType* gebildet werden. *ITRAPIndividual* setzt dabei das Interface *SolutionCandidateMulti* um. Die Fitness eines *ITRAPIndividual* soll durch Simulation bestimmt werden (vgl. Abschnitt 3.4.1). Um einen späteren Vergleich zwischen dieser und einer klassischen RAP-Fitnessevaluation durchführen zu können, wurde zusätzlich auch eine kombinatorische Fitnessevaluierung implementiert (Paket *problems.itrap.rapFitness*).

Für die Umsetzung der genetischen Operatoren wurden die Klassen *ITRAPInitializer*, *ITRAPMutation* und *ITRAPRecombination* implementiert. Für die Partikelschwarmoptimierung wurde für die Rekombination die normale (*ITRAPRecPSO*) und die vereinfachte Rekombination (*ITRAPRecSSO*) umgesetzt. Die Klasse *ITRAPTabuList* setzt das Interface *TabuList* um.

Komponente *Simulation* In der Komponente Simulation werden Klassen mit Ereignissteuerung (nach den so genannten *ActiveClasses* in *AnyLogic* im Paket *active* gruppiert), Experimentklassen (Paket *experiments*) sowie Hilfsklassen (Paket *util*) unterschieden.

Die Klassen des Paketes *active* werden in Abbildung 4.9 als UML-Klassendiagramm präsentiert. Diese repräsentieren die für die Zustandsverteilung relevanten Ereignisse, also die Zustandsverteilung der Komponenten (*ComponentModel*), die Abhängigkeiten (*DependencyModel*) sowie die Zustandsberechnung des Services und die Verwaltung der Administratoren (*Main*). Diese Klassen sind außerdem für die automatische Modellerstellung aus Lösungskandidaten verantwortlich. Für die Berechnung der Service-Zustandsverteilung aus den Zustandsverteilungen der Komponenten wird das Hilfsinterface *ComponentSys-*

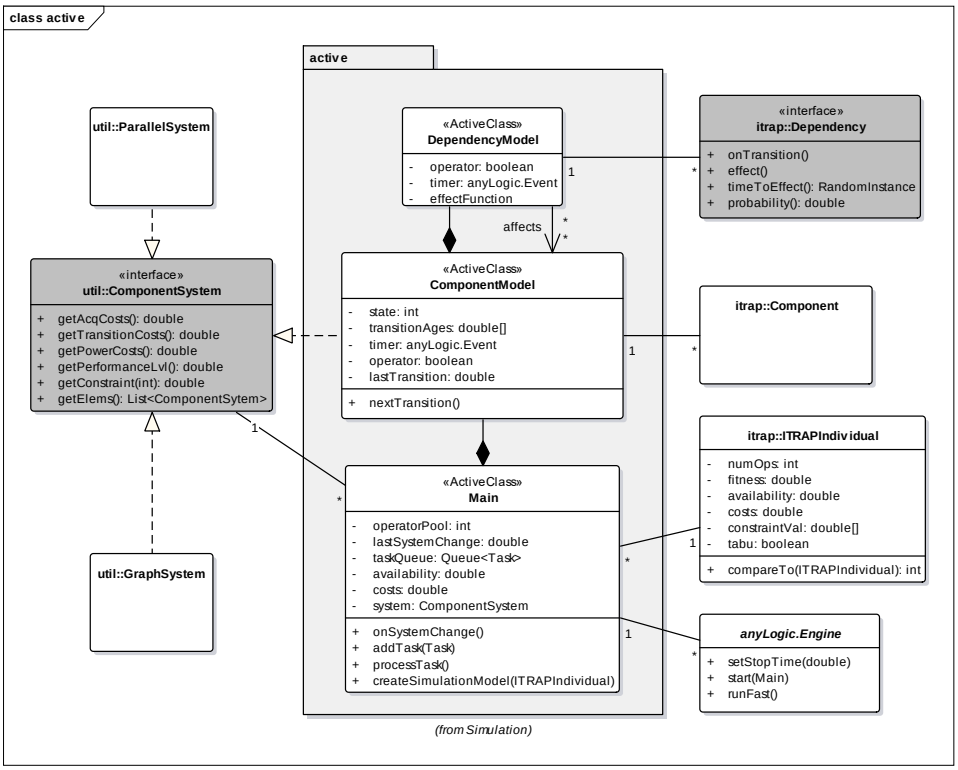

Abbildung 4.9: UML-Klassendiagramm der ActiveClasses in der Komponente *Simulation*

tem verwendet. Dieses wird von einzelnen Komponenten, aber auch von den Subsystemen (*ParallelSystem*) sowie dem System aus Subsystemen (*GraphSystem*) umgesetzt.

Herzstück des Paketes *experiments* ist die Klasse *Experiment*, die einen konkreten Optimierungslauf anstößt. Dazu wird eines der drei Optimierungsprobleme (*problem*) für ein *ITRAPScheme* adressiert. Für die später beschriebenen Experimente im Anwendungsfall eines Application Service Providers (ASP) instanziiert die Klasse *ASPScheme* ein *ITRAPScheme*. Weiterhin wird eine Lösungsmethode ausgewählt (*algorithm*), die in *iterations* Iterationen ausgeführt wird. Die Klassen *ITRAPFitness1*, *ITRAPFitness2* und *ITRAPFitness3* kapseln dann für die Evaluation eines Individuums den Aufruf der Monte-Carlo-Simulation, die von den Klassen des Paketes *active* gebildet wird. Dabei wird die Optimierung jeweils einmal optimistisch und pessimistisch durchgeführt.

Durchführung einer Optimierung Die Verteilung der Komponenten des Prototyps sowie dessen Benutzerschnittstelle zur Durchführung eines Optimierungslaufes sind in Abbildung 4.11 als UML-Deployment-Diagramm dargestellt. Ein Nutzer stößt dabei die Optimierung durch Aufruf der Klasse *Experiment* in der Komponente *Simulation* an, die in *AnyLogic* ausgeführt wird. Um die Meta-Heuristiken zur Optimierung aufzurufen, werden Informationen aus der Komponente *EvolAlg* extrahiert, die in der Datei *EvolAlg.jar* gespeichert sind.

Die Ergebnisse eines Optimierungslaufes werden sowohl für die optimistische als auch

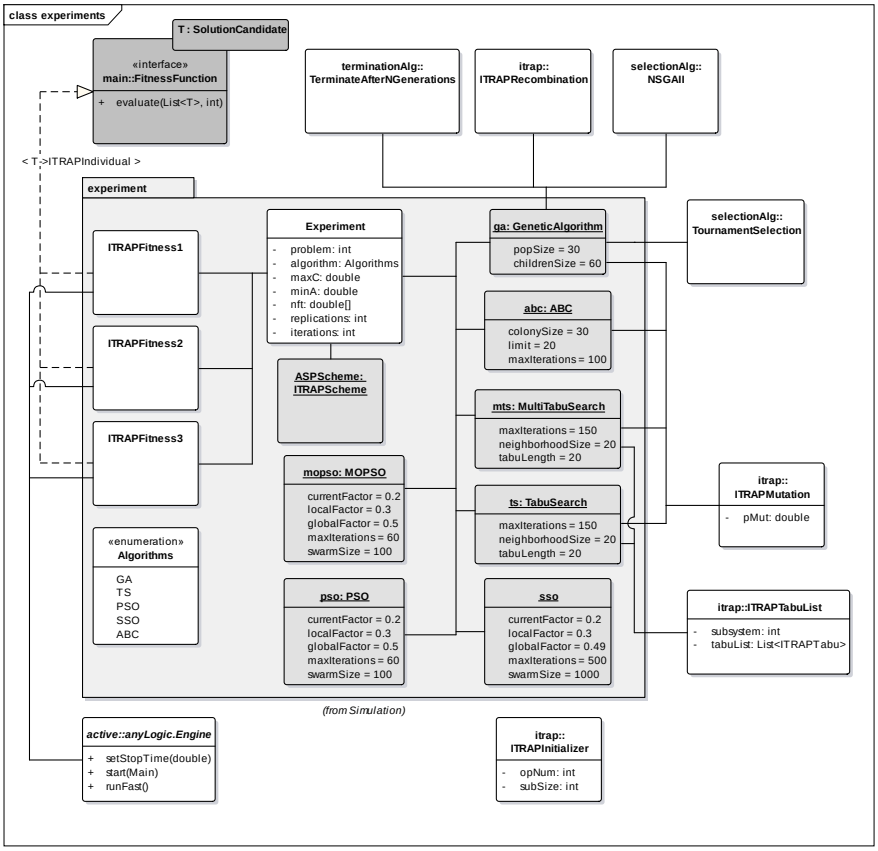

Abbildung 4.10: UML-Klassendiagramm der Experimentklassen in der Komponente *Simulation*

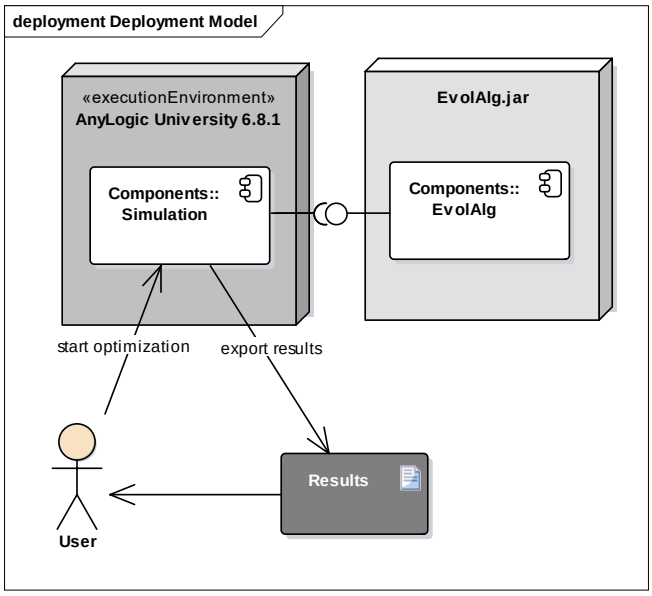

Abbildung 4.11: UML-Deployment-Diagramm für den Prototypen

für die pessimistische Zustandsverteilung ausgegeben. Diese Ergebnisse bilden sich dabei aus den Ergebnissen jeder Iteration eines Lösungsverfahrens. So werden für die Probleme (1) und (2) für jede Iteration zwei Ergebnisse (optimistisch und pessimistisch) berechnet. Für das Optimierungsproblem (3) werden die Ergebnisse jeder Iteration in zwei Mengen gesammelt, sodass am Ende des Optimierungslaufes eine Menge nicht-dominierter Lösungen für beide Zustandsverteilungen ausgegeben werden kann.

Die Ergebnisse werden nach Ablauf der Optimierung in die Datei *results.log* exportiert, von wo aus sie vom Nutzer weiterverarbeitet werden können. Ein Beispiel für die Ausgabe der Ergebnisse der einkriteriellen Optimierungsprobleme in dieser Datei ist in Abbildung 4.12 dargestellt. Dabei wurden für die Zufälligen Intervallgrößen Intervalle der Länge 0 ausgewählt, sodass die optimistische und die pessimistische Zustandsverteilung zusammenfallen. Ein Lösungsverfahren wurde hier in zehn Durchläufen angewendet, für die die globale Optima ausgegeben werden.

```
1  Pessimistic:
2  [1; ({C1};{C2|C2:p};{C3|C3:p};{C4};{C5|C5:p};{C6|C6:p|C6:p};{C7|C7:p};{C8};{C9|C9};{C10};{C11});0.996503806;524373.18;0.996503806
3  , 1; ({C1};{C2|C2:p};{C3|C3:p};{C4};{C5|C5:p};{C6|C6:p|C6:p};{C7|C7:p};{C8};{C9|C9};{C10};{C11});0.996472868;523548.38;0.996472868
4  , 1; ({C1};{C2|C2:p};{C3|C3};{C4};{C5|C5:p};{C6|C6:p};{C7|C7:p};{C8};{C9|C9};{C10};{C11});0.996402613;524601.61;0.996402613
5  , 1; ({C1};{C2|C2:p};{C3|C3};{C4};{C5|C5:p};{C6|C6:p};{C7|C7:p};{C8};{C9|C9};{C10};{C11});0.99637912;523381.27;0.99637912
6  , 1; ({C1};{C2|C2:p};{C3|C3};{C4};{C5|C5:p};{C6|C6:p};{C7|C7:p};{C8};{C9|C9};{C10};{C11});0.996327561;523571.03;0.996327561
7  , 1; ({C1};{C2|C2:p};{C3|C3:p};{C4};{C5|C5:p};{C6|C6};{C7|C7:p};{C8};{C9|C9};{C10};{C12});0.996084455;524715.91;0.996084455
8  , 1; ({C1};{C2|C2:p};{C3};{C4};{C5|C5};{C6|C6:p};{C7|C7:p};{C8};{C9|C9};{C10};{C11});0.995719095;523567.95;0.995719095
9  , 1; ({C1};{C2|C2};{C3};{C4};{C5|C5:p};{C6|C6:p};{C7|C7:p};{C8};{C9|C9};{C10};{C11});0.995242006;524754.94;0.995242006
10 , 1; ({C1};{C2|C2:p};{C3|C3|C3:p};{C4};{C5|C5:p};{C6|C6:p};{C7};{C8};{C9|C9};{C10};{C12});0.994625741;523389.96;0.994625741
11 , 1; ({C1};{C2|C2:p};{C3|C3:p};{C4};{C5|C5:p|C5:p};{C6|C6:p};{C7|C7};{C8};{C9};{C10};{C11});0.99333291;522720.75;0.99333291
12 ]
13 Optimistic:
14 []
15 computed in 745.041 seconds
```

Abbildung 4.12: Beispielhafte Ausgabe in der Results.log

Für jeden (sub)optimalen Lösungskandidaten werden dabei die Anzahl an Administratoren, das Servicedesign (Subsysteme getrennt per Semikolon), Verfügbarkeit, Kosten sowie Fitness ausgegeben. Für die ausgewählten Komponenten wird der definierte Name eingetragen, hier „C1“ bis „C11“. Hinter diesem Namen wird die Kennung des jeweiligen Redundanzmechanismus angegeben. In diesem Fall wird aktive Redundanz mit einer leeren Zeichenfolge und passive Redundanz mit „:p“ gekennzeichnet. Wie an den Fitnesswerten erkannt werden kann, wurde hier das Optimierungsproblem (1) adressiert, da die Verfügbarkeit die Fitness maßgeblich bestimmt.

Um die Ergebnisse weiterzuverarbeiten, kann die Lösung mit der höchsten Fitness isoliert betrachtet werden (hier Zeile 3). Für den Vergleich der Performance der Lösungsalgorithmen bietet sich jedoch eine Betrachtung mehrerer Iterationen an, um die Effekte der Zufallskomponente abschätzen zu können.

Für den Fall der mehrkriteriellen Optimierung für das Problem (3) werden statt einem Einzelergebnis für jeden Durchlauf des Optimierungsverfahrens die Liste nicht-dominierter Lösungen ausgegeben.

4.1.2 Verifikation

Die Verifikation des Prototypen hat zum Ziel, die Erfüllung der Anforderungen zu analysieren. Diese Anforderungen wurden in Abschnitt 3.1 aufgestellt, um ein RAP für IT-Service-Design zu spezifizieren (vgl. Tabellen 3.1, 3.2 und 3.3). Die Überprüfung der Anforderungsgruppe F1, der Optimierung von Service-Designs, kann dabei erst nach der Prüfung der weiteren funktionalen Anforderungen erfolgen, da diese in Abhängigkeit stehen. Zunächst sollen daher die Anforderungen geprüft werden, die an die Modellierung möglicher Designs, also des Suchraums, gestellt worden sind (F2).

Dazu wird das Meta-Modell des ITRAP untersucht (vgl. Abbildung 3.1). Ein ITRAP wird durch eine Menge von Subsystemen gekennzeichnet, deren Verbindung in einer Adjazenzmatrix beschrieben wird. Somit wird die Anforderung, ein System aus Subsystemen modellieren zu können (F2.1) erfüllt.

Die zweite Anforderung in diesem Bereich betrifft die Modellierung möglicher Komponenten und Redundanzmechanismen (F2.2). Im Meta-Modell des ITRAP besteht ein Subsystem aus mehreren Komponententypen, zu denen jeweils mehrere Redundanztypen zugeordnet werden können. Ein Lösungskandidat wird dann durch eine konkrete Auswahl von Komponenten- und Redundanztypen in jedem Subsystem gebildet. Damit ist diese Anforderung erfüllt.

Die Modellierung von Administratoren im Suchraum ist als natürliche Zahl in jedem Lösungskandidaten abgebildet. Die Anforderung F2.3 ist somit erfüllt. Anforderung F2.4 verlangt die Unbeschränktheit des Suchraumes. Da ein Lösungskandidat theoretisch aus unendlich vielen Komponenten bestehen kann, ist auch diese Anforderung verwirklicht. Es kann festgestellt werden, dass das ITRAP einen Suchraum möglicher Designs beschreibt, der für ein RAP für IT-Service-Design gefordert wird.

Als Nächstes werden die Anforderungen untersucht, die an die Verfügbarkeitsvorhersage gestellt wurden (F3). Ob und inwiefern das ITRAP diese Anforderungen erfüllt, wird in diesem Abschnitt mit verschiedenen Testverfahren untersucht. Dazu gehören im wesentlichen der Vergleich der (Simulations-)Ergebnisse mit anderen Modellen und Validitätstests mit Festwerten (vgl. [Sar10]).

Um die Verfügbarkeit von isolierten Komponenten vorherzusagen (F3.1), muss zunächst deren (multipler) Zustandsraum abbildbar sein (F3.1.1). Dies wird einerseits für einen binären sowie für einen mehrwertigen Zustandsraum untersucht. Um die Verifikation des Prototypen besser nachvollziehen zu können, werden diese Tests im Detail beschrieben.

Für eine Komponente mit binärem Zustand kann die Verfügbarkeit kombinatorisch nach $A_0 = \frac{MTTF}{MTTF+MTTR}$ berechnet werden. Dies wurde für verschiedene Verteilungen untersucht, um gleichzeitig die Anforderung F3.1.3 nach beliebigen Zeitverteilungen zu

überprüfen. Dafür wurden jeweils 1000 Replikationen der Simulation (Laufzeit ein Jahr) ausgeführt und ein 0,99-Konfidenzintervall berechnet. Die Ergebnisse sind in Tabelle 4.1 dargestellt, alle Zeitangaben sind in Stunden angegeben. Wie zu erkennen ist, beinhaltet das Konfidenzintervall den Erwartungswert in jedem Fall.

Tabelle 4.1: Verifikationsergebnisse für den Test des binären Zustandsraums

TTF	TTR	A_0	Konfidenzintervall
0,5	0,5	0,5	$[0,5; 0,5]$
11	1	$0,91\overline{6}$	$[0,91\overline{6}; 0,91\overline{6}]$
$\sim$ exp(0,5)	$\sim$ exp(0,5)	0,5	$[0,4997; 0,5004]$
$\sim$ exp(1000)	$\sim$ exp(1)	0,999	$[0,9989; 0,9991]$
$\sim$ N(10; 2)	$\sim$ N(0,1; 0,02)	0,9901	$[0,9901; 0,9901]$
$\sim$ U(8; 12)	$\sim$ U(0,02; 0,18)	0,9901	$[0,9901; 0,9901]$
$\sim$ T(8; 10; 12)	$\sim$ T(0,02; 0,1; 0,18)	0,9901	$[0,9901; 0,9901]$

Für den Test multipler Zustände wurden die Simulationsergebnisse mit der mathematischen Lösung von zwei zeitkontiniuerlichen Markov-Ketten (CTMC) verglichen. Dafür wurde eine Komponente mit drei Zuständen (Abbildung 4.13) und eine mit vier Zuständen (Abbildung 4.14) mit verschiedenen Performance-Levels modelliert. Da die verschiedenen Verteilungstypen schon im vorangegangenen Test untersucht wurden und in AnyLogic integriert sind, wird sich im folgenden auf exponentielle Verteilungen beschränkt, um diese einfachen CTMC lösen zu können.

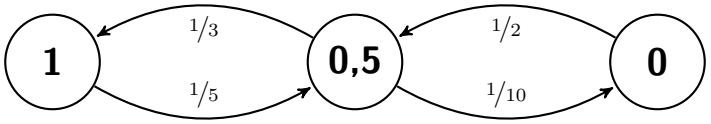

Abbildung 4.13: CTMC einer Komponente mit drei Zuständen

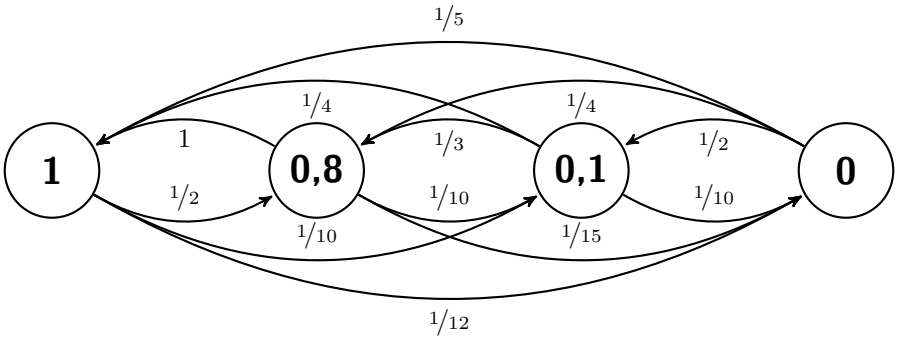

Abbildung 4.14: CTMC einer Komponente mit vier Zuständen

Mit einer konstanten Last vom Niveau 1 kann die Verfügbarkeit durch die stationäre Lösung der CTMC beschrieben werden. Die CTMC-Verfügbarkeit sowie das 0,99-Konfidenzintervall der Simulation nach 10.000 Replikationen ist in Tabelle 4.2 dargestellt.

Da der Erwartungswert innerhalb der Konfidenzintervalle liegt, kann der Zustandsraum einer isolierten Komponente modelliert werden. Damit sind die Anforderungen F3.1.1 und F3.1.3 nach Modellierung eines mehrwertigen Zustandsraums mit beliebigen Übergangsverteilungen erfüllt.

Tabelle 4.2: Verifikationsergebnisse für den Test des multiplen Zustandsraums

Zustände	A_0	Konfidenzintervall
3	0,7558	$[0,7557; 0,7561]$
4	0,7156	$[0,7154; 0,7157]$

Zur Modellierung verschiedener Fehlertypen wurde ein Test mit zwei Fehlertypen durchgeführt. Dafür wurde eine CTMC modelliert, die beide Ausfallzustände sowie einen Zustand für die Verfügbarkeit abbildet (siehe Abbildung 4.15). Das Konfidenzintervall der Simulation $[0,666; 0,667]$ beinhaltet den Verfügbarkeitswert von $2/3$, der durch Lösung der CTMC identifiziert wurde. Damit kann festgehalten werden, dass die Anforderung F3.1.2 nach verschiedenen Fehlerarten durch die Verfügbarkeitsvorhersage realisiert wird.

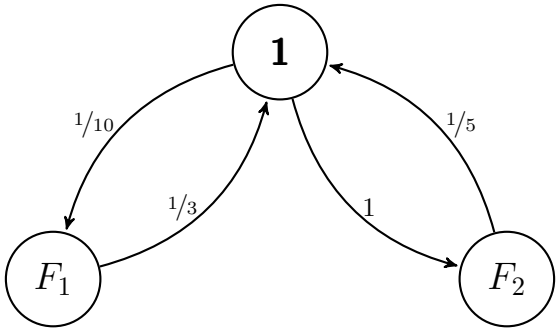

Abbildung 4.15: CTMC einer binären Komponente mit zwei Fehlerarten

Für die Abbildung ungenauer Modellparameter wurde das Konzept der Zufälligen Intervallgröße eingeführt. Um die korrekte Berechnung der ungenauen Verfügbarkeit aus den Parametern zu überprüfen, wurde der binäre Zustandstest für Zufällige Intervallgrößen verschiedenen Typs durchgeführt. Die Ergebnisse werden in Tabelle 4.3 präsentiert und weisen die Erfüllung der Anforderung nach Ungenauigkeit in der RAP-Definition nach (F3.1.4). Damit wurden alle Anforderungen an die Modellierung isolierter Komponenten (F3.1) realisiert.

Tabelle 4.3: Verifikationsergebnisse für den Test der Zufälligen Intervallgröße

TTF	TTR	A_0	Konfidenzintervall-Intervall
$[5; 10]$	$[1; 2]$	$[0,714; 0,909]$	$[0,714; 0,909]$
N($[5; 10]$; 1)	N($[1; 2]$; 0,25)	$[0,714; 0,909]$	$[0,714 \pm 2,2E\text{-}4; 0,909 \pm 1,3E\text{-}4]$
exp($[5; 10]$)	exp($[1; 2]$)	$[0,714; 0,909]$	$[0,714 \pm 1,3E\text{-}3; 0,909 \pm 6,8E\text{-}4]$
U(3; $[7; 17]$)	U(0; $[2; 4]$)	$[0,714; 0,909]$	$[0,714 \pm 5,8E\text{-}4; 0,909 \pm 3,5E\text{-}4]$
T(1; $[2; 11]$; 15)	T(0; $[1; 4]$; 5)	$[0,667; 0,818]$	$[0,667 \pm 7,3E\text{-}4; 0,818 \pm 5,4E\text{-}4]$

Als Nächstes werden die Anforderungen an die Abhängigkeiten zwischen den Komponenten untersucht (F3.2). Dabei wird zunächst auf die Anforderung nach heterogenen Subsystemen eingegangen (F3.2.2). Dazu wird einerseits die kombinatorische Formel für aktive Redundanz, $A = 1 - \prod_i (1 - A_i)$, für binäre Komponenten angewendet. Andererseits wird für Komponenten mit multiplen Zuständen die Formel $per(s) = \sum per(c)$ für die beiden oben eingeführten Beispiele verifiziert (vgl. Abbildungen 4.13 und 4.14).

Tabelle 4.4: Verifikationsergebnisse für den Test der heterogenen, aktiven Redundanz

Test	Zustände	A_c	A_0	Konfidenzintervall
Binär	2	0,9091		
	2	0,9302	0,9937	$[0,99357; 0,99373]$
	2	0,7143	0,9982	$[0,99817; 0,99824]$
Multipel	3	0,7558		
	4	0,7156	0,9429	$[0,9426; 0,9431]$

Die Ergebnisse dieser Tests sind in Tabelle 4.4 dargestellt. Dabei wurden für den binären Test zwei Subsysteme aus zwei bzw. drei Komponenten definiert. Für den Test der korrekten Performanceberechnung bei mehreren Zuständen wurde ein Subsystem aus den Komponenten mit multiplem Zustand der obigen Tests genutzt. Durch die korrekten Ergebnisse kann die Erfüllung der Anforderung nach heterogenen Subsystemen (F3.2.2) bestätigt werden.

Auf Basis der korrekten Berechnung der Subsystemperformance aus den Komponentenzuständen kann die Anforderung F3.2.1 nach Modellierung komplexer Systeme getestet werden. Diese Anforderung wird im ITRAP durch den Subsystemgraphen adressiert. Um die korrekte Funktionsweise des Graphen sowie der damit verbundenen Berechnung der Performance des Systems zu überprüfen, wurden komplexe Systeme aus der RAP-Literatur modelliert. Zunächst wurde jedoch ein parallel-serielles System getestet.

Dabei werden in n Subsystemen jeweils m unabhängige Komponenten in aktiver Redundanz parallel betrieben. Daher gilt:

$$A_0 = (1 - (1 - A)^m)^n$$

Jede Komponente hat einen binären Zustandsraum mit $A = 0.83\overline{3}$. Die Ergebnisse eines Tests für verschiedene Werte von n und m werden in Tabelle 4.5 präsentiert.

Das erste Beispiel eines komplexen Systems ist das so genannte Überbrückungssystem. Der Subsystemgraph dieses Systems ist in Abbildung 4.16 dargestellt. Dabei fungiert Subsystem 3 als Überbrückung, falls beide parallele Pfade ($1 \rightarrow 4$ und $2 \rightarrow 5$) von Ausfällen betroffen sind.

Wie in [RMR97, Che06] und [SBR10] festgestellt, lässt sich die Verfügbarkeit des

Tabelle 4.5: Verifikationsergebnisse für den Test des seriell-parallelen Systems

n	m	A_0	Konfidenzintervall
1	1	0,8333	$[0,8327; 0,8339]$
2	1	0,6944	$[0,6940; 0,6954]$
3	1	0,5787	$[0,5784; 0,5800]$
1	2	0,9722	$[0,9721; 0,9725]$
2	2	0,9452	$[0,9449; 0,9455]$
3	2	0,9190	$[0,9184; 0,9191]$
1	3	0,99537	$[0,99529; 0,99543]$
2	3	0,9908	$[0,9907; 0,9909]$
3	3	0,9862	$[0,9861; 0,9863]$

Systems mit gegebenen Subsystemverfügbarkeiten wie folgt berechnen:

$$A_0 = A_1A_4 + A_2A_5 + A_2A_3A_4 + A_1A_3A_5 + 2A_1A_2A_3A_4A_5 - A_2A_3A_4A_5 \\ -A_1A_3A_4A_5 - A_1A_2A_4A_5 - A_1A_2A_3A_5 - A_1A_2A_3A_4$$

Besteht das System aus Subsystemen mit gleicher Verfügbarkeit lautet die Gleichung also $A_0 = 2A^2 + 2A^3 + 2A^5 - 5A^4$.

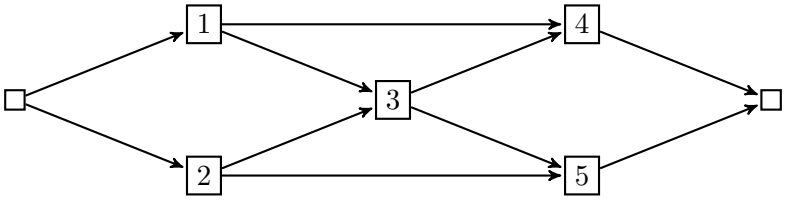

Abbildung 4.16: Graph des Überbrückungssystems

In [Che06] wird ein weiteres Beispiel für ein komplexes System eingeführt, welches sich als hierarchisches seriell-paralleles System beschreiben lässt. Der Subsystemgraph dieses Systems wird in Abbildung 4.17 präsentiert. Die Abhängigkeit der System- von den Subsystemverfügbarkeiten lässt sich wie folgt beschreiben:

$$A_0 = 1 - (1 - A_1A_2)(1 - (1 - (1 - A_3)(1 - A_4))A_5)$$

Für gleiche Subsystemverfügbarkeiten also $A_0 = 1 - (1 - A^2)(1 - (1 - (1 - A)^2)A)$.

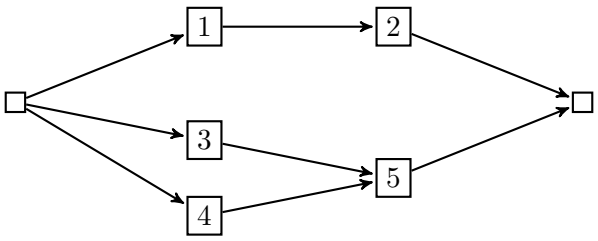

Abbildung 4.17: Graph des hierarchischen Systems aus [CY05]

Ein weiteres Beispiel für ein hierarchisches System stammt aus [SBR10] und ist in Abbildung 4.18 als Subsystemgraph visualisiert. Die Verfügbarkeit lässt sich wie folgt berechnen:

$$A_0 = (1-(1-(1-(1-A_3)(1-A_1A_2))A_4)(1-A_5A_6))(1-(1-A_7)(1-A_8)(1-A_9))A_{10}$$

Für gleiche Verfügbarkeiten gilt hier:

$$A_0 = (1-(1-(1-(1-A)(1-A^2))A)(1-A^2))(1-(1-A)^3)A$$

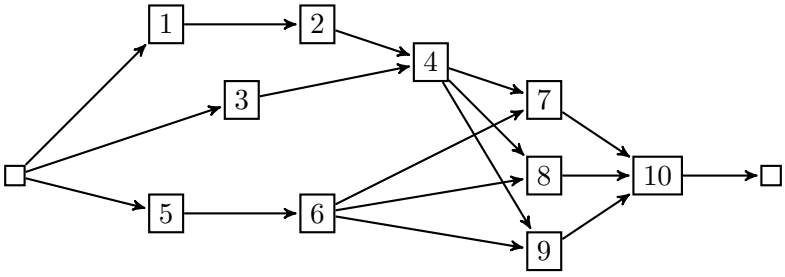

Abbildung 4.18: Graph des hierarchischen Systems aus [SBR10]

Diese drei Beispiele und die gegebenen Gleichungen können bei binären Subsystemen (bestehend aus einer binären Komponente) zur Verifikation komplexer Systeme genutzt werden. Für verschiedene Performancelevel wurde das Überbrückungssystem als CTMC mit unabhängigem Komponentenausfall in 1024 Zuständen modelliert (vier Zustände für jedes Subsystem aus einer Komponente, vgl. Abbildung 4.14). Die Ergebnisse dieser Tests sind in Tabelle 4.6 dargestellt.

Tabelle 4.6: Verifikationsergebnisse für den Test der komplexen Designs

Graph	Zustände	A	A_0	Konfidenzintervall
4.16	binär	0,8333	0,9388	$[0,9385; 0,9391]$
4.16	multipel	0,7156	0,7980	$[0,7978; 0,7985]$
4.17	binär	0,8333	0,9420	$[0,9417; 0,9422]$
4.18	binär	0,8333	0,7765	$[0,7758; 0,7771]$

Da die Konfidenzintervalle die erwarteten Ergebnisse beinhalten, können damit die in der Literatur verwendete Beispiele für hierarchische bzw. komplexe Designs im ITRAP abgebildet werden. Durch die Adjazenzmatrix erfüllt das ITRAP somit die Anforderung an die Modellierung beliebiger komplexer Designs (F3.2.1). Damit können die erwähnten Vorarbeiten, in denen nur einzelne komplexe Designs betrachtet worden, im ITRAP generalisiert werden.

Eine weitere Anforderung in dem Bereich der Abhängigkeiten ist die Modellierung passiver Redundanzmechanismen (F3.2.3). Um diese Anforderung zu testen, wurden zwei Testfälle definiert. Im ersten Fall wurde ein homogenes 1+1-System modelliert, also ein System mit einer aktiven und einer passiv-redundanten Komponente. Eine CTMC dieses

Systems ist in Abbildung 4.19 dargestellt.

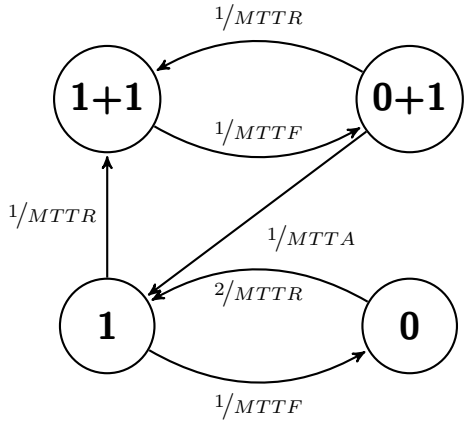

Abbildung 4.19: CTMC eines homogenen 1+1-Standby-Systems

Als weiteren Testfall für die passive Redundanz wurde ein 2+1-System modelliert. Die CTMC dieses Beispiels wird in Abbildung 4.20 präsentiert. Hierbei sind für alle Aufgaben genügend Administratoren vorhanden, sodass beispielsweise die Wiederherstellung von Komponenten parallel ablaufen kann. Dies wird in der CTMC durch die Faktoren der Raten ausgedrückt, so z.B. wenn alle drei Komponenten des Systems ausgefallen sind (zwischen Zustand 0 und 1). Der Ausfall von Komponenten läuft in jedem Fall für aktive Komponenten parallel ab (z.B. zwischen Zustand 2+1 und 1+1). Für beide Systeme hat jede Komponente ein Leistungsniveau von 1. Während im ersten System eine Last von 1 erzeugt wird, wird für das 2+1-System eine Last von 2 definiert.

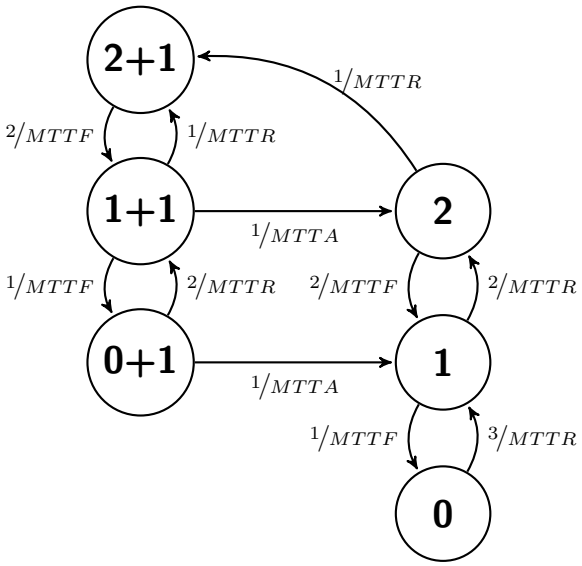

Abbildung 4.20: CTMC eines homogenen 2+1-Systems

Die Ergebnisse der Tests der passiven Redundanz sind in Tabelle 4.7 dargestellt. Durch die erfolgreiche Durchführung beider Tests kann festgehalten werden, dass die Anforderung F3.2.3 nach der Abbildung mehrerer Redundanztypen durch das ITRAP erfüllt

wird.

Tabelle 4.7: Verifikationsergebnisse für den Test der passiven Redundanz

System	MTT F/R/A	Last	A_0	Konfidenzintervall
$1+1$	10/2/0,5	1	0,9539	$[0,9537; 0,9541]$
$2+1$	10/2/0,5	2	0,9452	$[0,9449; 0,9453]$

Um die Anforderung nach generischen Abhängigkeiten (F3.2.4) zu testen, wurde die so genannte Ausfallabhängigkeit definiert. Diese Abhängigkeit kann zwischen zwei Komponententypen c_1 und c_2 definiert werden und führt bei Ausfall von c_1 mit Wahrscheinlichkeit p zum Ausfall von c_2. Eine CTMC dieser Ausfallabhängigkeit für heterogene, binäre Komponenten ist in Abbildung 4.21 gegeben (vgl. Abbildung 2.2).

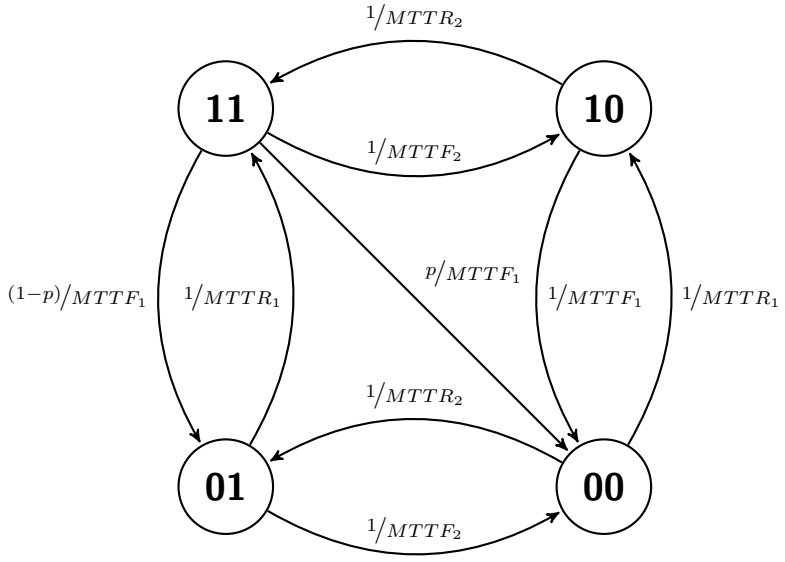

Abbildung 4.21: CTMC von zwei ausfall-abhängigen Komponenten

Die Ergebnisse dieses Tests sind in Tabelle 4.8 visualisiert. Die Ergebnisse für die Ausfallabhängigkeit sind schlüssig, daher wird die Erfüllung der Anforderung F3.2.4 nach generischen Abhängigkeiten bejaht. Damit konnten alle Anforderungen an Abhängigkeiten zwischen Komponenten (F3.2) realisiert werden.

Tabelle 4.8: Verifikationsergebnisse für den Test der Ausfallabhängigkeit

MTTF/R_1	MTTF/R_2	p	A_0	Konfidenzintervall
10/2	20/1,5	1	0,92466	$[0,92415; 0,92473]$
10/2	20/1,5	0,5	0,95569	$[0,95538; 0,95584]$

Die dritte Gruppe von Anforderungen an die Verfügbarkeitsvorhersage eines RAP für IT-Service-Design F3.3 behandelt den Einfluss von Administratorinteraktion auf die Verfügbarkeit. Dafür werden zunächst zwei Typen von Administratoraufgaben (F3.3.1) unterschieden: Transitions- und Übernahmeaufgaben. Die Erfüllung dieser Aufgaben kann mit

Interaktionsfehlern (F3.3.3) verbunden sein, die zur Wiederholung einer Aufgabe führen. In den im folgenden beschriebenen Testfällen werden diese beiden Anforderungen getestet.

Eine CTMC einer binären Komponente mit manueller Wiederherstellung ist in Abbildung 4.22 dargestellt. Dabei gilt

$$\sigma = \begin{cases} 0, \text{ falls } o = 0 \\ 1, \text{ sonst} \end{cases}$$

Die Wiederherstellung kann in diesem Fall also nur erfolgen, wenn im Design mindestens ein Administrator vorgesehen ist. Die mittlere Wiederherstellungszeit verlängert sich mit steigender Fehlerwahrscheinlichkeit p_{err}. Die durchschnittliche Auslastung der Administratorressourcen kann mit π_0/o berechnet werden (falls $o > 0$).

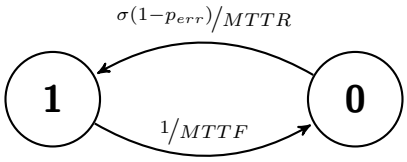

Abbildung 4.22: CTMC einer Komponente mit zwei Zuständen und fehlerbehafteter, manueller Wiederherstellung

Für den Test der Transitionsaufgabe wurden die Werte von o und p_{err} variiert. Neben der Verfügbarkeit des Systems wurde auch die Auslastung der Operatoren U mit einem theoretischen Wert verglichen, der durch die Lösung der CTMC bestimmt wurde. Die Ergebnisse dieses Tests sind in Tabelle 4.9 aufgelistet.

Tabelle 4.9: Verifikationsergebnisse für den Test der Transitionsaufgabe

o	p_{err}	A_0	Konfidenzintervall (A)	U_0	Konfidenzintervall (U)
0	0	0	$[0,0010; 0,0012]$	0	$[0; 0]$
1	0	0,8333	$[0,8331; 0,8343]$	0,1667	$[0,1663; 0,1675]$
2	0	0,8333	$[0,8331; 0,8343]$	0,0833	$[0,0829; 0,0835]$
1	0,1	0,8182	$[0,8175; 0,8188]$	0,1818	$[0,1814; 0,1827]$
1	0,5	0,7143	$[0,7131; 0,7149]$	0,2857	$[0,2851; 0,2861]$
1	1	0	$[0,0011; 0,0013]$	1	$[0,9988; 0,9990]$

Für diesen Test wurde ebenfalls eine Simulationszeit von einem Jahr gewählt. Da alle Systeme initial verfügbar sind, wird der theoretische Wert des stabilen Zustands nur asymptotisch in der Simulation erreicht. Dies erklärt die Differenz zwischen theoretischem Wert und Konfidenzintervall in den Extremfällen. Dennoch entsprechen diese Ergebnisse eher der Realität als der Wert des stabilen Zustands. Dies und die Tatsache, dass die Konfidenzintervalle in den anderen Fällen die theoretischen Werte beinhalten, verdeutlichen, dass das Verhalten der Simulation für die Transitionsaufgaben korrekt ist.

Um die Korrektheit der Übernahmeaufgaben zu zeigen, wurde das oben genutzte 1+1-System mit Interaktion für die Übernahme modelliert (siehe Abbildung 4.23).

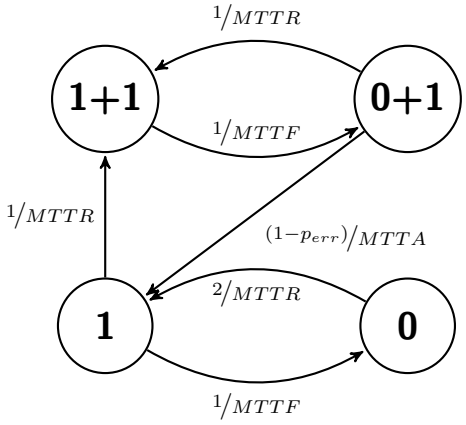

Abbildung 4.23: CTMC eines homogenen 1+1-Standby-Systems mit fehlerbehafteter Übernahme

In Tabelle 4.10 sind die Ergebnisse dieses Tests dargestellt. Auffällig ist hier im Vergleich zum vorangegangenen Test, dass auch in den Extremfällen die Zielwerte von den Konfidenzintervallen eingeschlossen werden. Dies liegt in der Tatsache begründet, dass die Übernahme bei Wiederherstellung der Primärkomponente abgebrochen wird. Mit diesen Ergebnissen können die Anforderungen nach Administratoraufgaben (F3.3.1) und Interaktionsfehlern (F3.3.3) als erfüllt angesehen werden.

Tabelle 4.10: Verifikationsergebnisse für den Test der Übernahmeaufgabe

p_{err}	A_0	Konfidenzintervall (A)	U_0	Konfidenzintervall (U)
0	0,9540	$[0,9537; 0,9612]$	0,0330	$[0,0329; 0,0331]$
0,1	0,9514	$[0,9511; 0,9515]$	0,0358	$[0,0357; 0,360]$
0,5	0,9341	$[0,9338; 0,9343]$	0,0549	$[0,0548; 0,0553]$
1	0,8333	$[0,8329; 0,8341]$	0,1667	$[0,1657; 0,1670]$

Der Test der Anforderung F3.3.2, der Abbildung von Prioritäten, wird auf Basis des 2+1-Systems aus Abbildung 4.20 durchgeführt. Dabei wird Administratorinteraktion für die Aufgaben der Wiederherstellung sowie der Übernahme gefordert. Dies führt zu vier Aufgaben, von denen maximal drei gleichzeitig anstehen können (in Zustand 0). Wenn die Anzahl an Administratoren o in diesem System also kleiner drei ist, müssen Prioritäten definiert werden. In diesem Fall wird der Übernahme eine höhere Priorität als den Wiederherstellungsaufgaben zugeordnet.

Eine CTMC für diese Fälle kann in Abhängigkeit der Anzahl an Administratoren o definiert werden, so z.B. in Abbildung 4.24 für $o = 1$. Dabei kann in jedem Zustand nur eine Aufgabe durchgeführt werden, die anhand der Priorität bestimmt wird. Im Zustand 0 kann nur eine von drei möglichen Wiederherstellungen parallel durchgeführt werden, daher ist die theoretische Rate aus Abbildung 4.20 gedrittelt. Ähnlich verhält es sich in Zustand 0+1, wo keine Wiederherstellung, sondern nur die Übernahme durchgeführt wird.

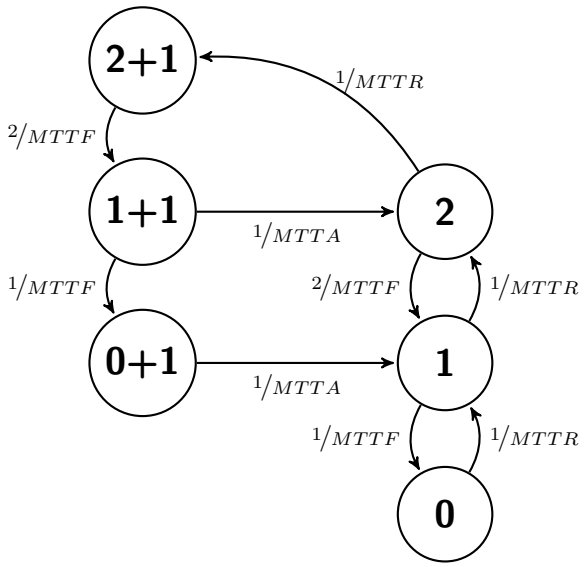

Abbildung 4.24: CTMC eines homogenen 2+1-Systems mit einem Administrator

In Tabelle 4.11 sind die Ergebnisse dieses Tests notiert. Dabei gilt für alle Komponenten $MTTF = 10, MTTR = 2$ und $MTTA = 0,5$. Mit steigender Admin-Anzahl nähert sich die Verfügbarkeit dem theoretischen Maximum in diesem Fall an. Die Ergebnisse der Simulation stimmen mit den Werten des stabilen Zustands überein. Daher kann die Erfüllung der Anforderung nach Prioritäten (F3.3.2), und somit auch die Anforderung nach der Modellierung von Administratoren (F3.3) bestätigt werden. Damit konnten alle gestellten Anforderungen an die Verfügbarkeitsvorhersage eines RAP für IT-Service-Design (F3) vom ITRAP erfüllt werden.

Tabelle 4.11: Verifikationsergebnisse für den Test der Administratorprioritäten

o	A_0	Konfidenzintervall (A)	U_0	Konfidenzintervall (U)
1	0,8964	$[0,8957; 0,8966]$	0,4169	$[0,4164; 0,6183]$
2	0,9431	$[0,9430; 0,9435]$	0,2158	$[0,2150; 0,2160]$
3	0,9452	$[0,9450; 0,9453]$	0,1442	$[0,1438; 0,1445]$
4	0,9452	$[0,9450; 0,9453]$	0,1081	$[0,1080; 0,1084]$

Die Anforderungsgruppe F4 besteht aus Anforderungen an die Kostenvorhersage eines RAP für IT-Service-Design. Dazu sollen Akquisitions-, Betriebs-, Personal- und Wiederherstellungskosten abbildbar sein.

Für die Anforderung nach Akquisitionskosten (F4.1) wurde der Test des seriell-parallelen Systems (vgl. Tabelle 4.5) mit Akquisitionskosten für die Komponenten wiederholt. So konnte die Erwartung, dass die Initialkosten der Summe der Komponenten-Akquisitionskosten entsprechen, für verschiedene Werte von n und m verifiziert werden.

Der Zusammenhang zwischen Systemdesign und Personalkosten ist in Abhängigkeit des Stundenlohns *wage* sowie der Anzahl an Administratoren o definiert. Um die korrekte Berechnung zu verifizieren, wurde der Prioritätstest (vgl. Tabelle 4.11) mit Fokus auf den Personalkosten erneut durchgeführt. Auch hier konnten die zu erwartenden Ergeb-

nisse erreicht werden. Daher wird die Anforderung F4.3 nach Personalkosten als erfüllt angesehen.

Um die Vorhersage der dynamischen Wiederherstellungs- (F4.4) und Betriebskosten (F4.2) zu verifizieren, wurden Komponenten mit zwei, drei und vier Zuständen getestet. Für die binäre Komponente gilt $MTTF = MTTR = 0,5$, die Komponenten mit multiplen Zuständen sind aus den obigen Tests entnommen (vgl. Abbildungen 4.13 und 4.14). Dabei wurden jeder Transition Wiederherstellungskosten der Höhe 1 zugeordnet. Die Betriebskostenwerte der einzelnen Zustände beginnen im Zustand niedrigster Performance bei 1 und steigen inkrementell in Zuständen höherer Performance an. So gilt beispielsweise für die Komponente mit drei Zuständen aus Abbildung 4.13 $op(0) = 1, op(0,5) = 2$ und $op(1) = 3$.

Die Simulationsergebnisse für die dynamischen Kostenarten sind in Tabelle 4.12 den theoretischen Werten gegenübergestellt. Durch die erfolgreiche Durchführung dieses Tests erfüllt das ITRAP die Anforderungen nach der Abbildung von Wiederherstellungs- und Betriebskosten (F4.2 und F4.4). Somit sind alle Anforderungen im Bereich F4 für die Vorhersage der Kosten von IT-Services erfüllt.

Tabelle 4.12: Verifikationsergebnisse für den Test der dynamischen Kosten

Zustände	A_0	C_{tr_0}	Konfidenzint. (C_{tr})	C_{op_0}	Konfidenzint. (C_{op})
2	0,5	8.760	[8.755; 8.766]	13.140	[13.136; 13.142]
3	0,756	1.324	[1.321; 1.325]	22.001	[21.990; 22.011]
4	0,716	3.893	[3.889, 8; 3.897]	49.646	[49.622; 49.654]

Neben den funktionalen Anforderungen nach der Vorhersage von Verfügbarkeit und Kosten eines IT-Service-Designs sollen auch Werte für Nebenbedingungen berechnet werden (F5). Die ITRAP-Definition lässt eine beliebige Anzahl von Nebenbedingungen zu, wie es in Anforderung F5.1 festgehalten wurde. Dazu sollen die entsprechenden Werte für jede eingesetzte Komponente aufsummiert werden (F5.2). Diese Berechnung konnte für das ITRAP verifiziert werden, indem der Test des seriell-parallelen Systems (vgl. Tabelle 4.5) mit verschiedenen Nebenbedingungen wiederholt wurde. Damit sind die Anforderungen in diesem Bereich erfüllt.

Nachdem die Anforderungen aus den Bereichen F2 bis F5 geprüft wurden, wird im Folgenden auf die Anforderung nach Designoptimierung (F1) eingegangen, da diese von der Beschreibung des Suchraums (F2) sowie der Vorhersage der Verfügbarkeit (F3), der Kosten (F4) und der Werte für die Nebenbedingungen (F5) eines Designs abhängig ist. Um die drei geforderten Optimierungsprobleme adressieren zu können, wurden für das ITRAP fünf Lösungsalgorithmen adaptiert: ein Genetischer Algorithmus (GA), Tabu-Suche (TS), Partikel- und Vereinfachte Schwarmoptimierung (PSO und SSO) sowie eine Künstliche Bienenkolonie (ABC).

Da für alle Algorithmen gemeinsame Operationen definiert wurden, werden zunächst diese Operationen überprüft. Bei der Initialisierung von Lösungskandidaten sollte eine diverse und über den Suchraum angemessen verteilte Population erstellt werden. Dies wur-

de in einem Test für verschiedene Werte für $subSize$ und $opNum$ sowie unterschiedliche Problemdefinitionen für 1000 Kandidaten überprüft. Die Ergebnisse sind in Tabelle 4.13 aufgelistet. Die Problemdefinition wird dabei durch die Werte n für die Anzahl Subsysteme, m für die Anzahl möglicher Komponenten in jedem Subsystem sowie r für die Anzahl möglicher Redundanzmechanismen gekennzeichnet. Weiterhin sind für verschiedene Steuerparameter der Anteil einzigartiger Individuen sowie der höchste Anteil eines gleichen Individuums notiert. Es wird deutlich, wie die Diversität der Initialpopulation von der Definitionskomplexität sowie den Parametern abhängt. Dies zeigt, dass die Initialisierung einer geeigneten Population von Lösungskandidaten durch die implementierte Operation möglich ist.

Tabelle 4.13: Verifikationsergebnisse für den Test der Initialisierungsoperation

n	m	r	$subSize$	$opNum$	einzigartige Ind.	max. Anteil
2	2	2	1	1/3	27,5 %	9,6 %
			3	1	82,8 %	1,6 %
			10	10	99,8 %	0,2 %
3	3	2	1	1/3	72,1 %	1,3 %
			3	1	99,2 %	0,3 %
			10	10	100 %	0,1 %

Für die Umsetzung einer Nachbarschaftssuche wurde die Mutationsoperation definiert. Dabei wird ein zufälliges Element eines Designs hinzugefügt bzw. entfernt. Die Korrektheit dieser Operation konnte in Tests nachgewiesen werden. Dies gilt auch für die spezifischen, problemabhängigen Operationen der Lösungsalgorithmen: Das uniforme Crossover, das ITRAP-Tabu sowie die Rekombination der PSO und der SSO.

Um die korrekte Funktionsweise der Lösungsalgorithmen zu testen, wurden sie für ein einfaches Optimierungsproblem angewandt: $\max x$ mit $x \in \mathbb{R} \wedge x \leq 10$. Dabei wird als Fitnessfunktion definiert:

$$f(x) = \begin{cases} x, \text{ falls } x \leq 10 \\ 10 - x, \text{ sonst.} \end{cases}$$

Für dieses Problem konnte die korrekte Funktionsweise der einkriteriellen Algorithmen (GA mit Turnierselektion, TS, PSO und ABC) verifiziert werden, da sich die Fitness des gefundenen Optimums mit steigender Generationszahl dem theoretischen Wert annähert. Für die mehrkriteriellen Algorithmen (NSGA-II, MTS, MOPSO und MOABC) wurde folgendes Problem untersucht: $\max x, y$ mit $x + y \leq 10$. Auch hier näherten sich die Algorithmen sukzessive der optimalen Lösung an. Damit wurde die Korrektheit der implementierten Lösungsalgorithmen im Allgemeinen und ITRAP-spezifisch nachgewiesen.

Da alle durchgeführten Tests erfolgreich abgeschlossen wurden, ist die nicht-funktionale Anforderung NF1 nach Korrektheit für das ITRAP erfüllt. Der Forderung nach Änderbarkeit (NF3) wird durch das ITRAP insoweit nachgekommen, dass sowohl die Definition und der Transformationsprozess für das Simulationsmodell modular und mit loser Kopplung gestaltet worden, sodass Anpassungen ohne Nebeneffekte realisierbar sind. Die

Entscheidung, Meta-Heuristiken zur Lösung des ITRAP einzusetzen, garantiert eine hohe Änderbarkeit, da nur relativ wenig problemspezifische Informationen vonnöten sind und diese auch modular definiert sind. Zusätzlich ist die gewählte Matrix-Kodierung flexibel gegenüber weiteren Dimensionen, wobei die Effektivität der Subsystemoperationen jedoch verringert wird.

Ein wesentlicher Grund für die Auswahl eines Simulationsansatzes war die Sicherstellung der Skalierbarkeit (NF4). Da die Auswertung der einzelnen Individuen den Großteil des Zeitaufwandes eines Optimierungslaufes im Vergleich zu einem klassischen, kombinatorischen RAP-Ansatz darstellt, wurde die Skalierbarkeit der Vorhersage im Sinne der Simulationslaufzeit untersucht.

Die Laufzeit der Evaluation eines Lösungskandidaten wird dabei von der Anzahl verarbeiteter Ereignisse bestimmt. Diese Anzahl wird maßgeblich von zwei Aspekten bestimmt: Der Problemkomplexität und der gewünschten Ergebnisgenauigkeit. Je komplexer das Problem im Sinne der mittleren Ereignisanzahl im untersuchten Zeitintervall $[0; t]$ ist, desto länger dauert ein einzelner Simulationslauf. Tests mit verschiedenen Problemdefinitionen legen eine schwach quadratische Abhängigkeit der Simulationsdauer (inkl. Modellkonstruktion) von der mittleren Anzahl der Ereignisse nahe (Bestimmtheitsmaß $R^2 > 0,999$).

Eine Erhöhung der Genauigkeit kann durch Erhöhung der Replikationsanzahl erreicht werden. Zieht man die Definition des Konfidenzintervalls zurate, gilt ein quadratischer Zusammenhang zwischen Genauigkeit des Intervalls und Replikationsanzahl. Soll die Genauigkeit also verdoppelt werden, müssen vier Mal so viele Replikationen durchgeführt werden. Da diese Replikationen jedoch unabhängig sind, ist eine massive Parallelisierung möglich.

Es bleibt jedoch auch ohne eine Parallelisierung festzuhalten, dass eine steigende Problemkomplexität zu einem polynomial wachsenden Laufzeitverhalten führt. Die Modellkomplexität wächst dabei nur linear. Da exponentielle Abhängigkeiten vermieden werden, kann die Evaluation eines Individuums und damit die Optimierung als skalierbar betrachtet werden.

Für die finale Überprüfung der funktionalen Anforderungen im Bereich F1 sowie der nicht-funktionalen Anforderung nach Effizienz werden Experimente in einem realen Anwendungsfall durchgeführt. Diese werden im Folgenden beschrieben.

4.2 Setup und Ergebnisse der Experimente

In diesem Abschnitt der Arbeit wird ein Anwendungsfall genutzt, um die verbleibenden Anforderungen an ein RAP für IT-Service-Design zu überprüfen. Dafür wird zunächst die Effizienz der Lösungsmethoden für ein Teilproblem untersucht. Daraufhin werden die definierten Lösungsmethoden für den ITRAP- und einen RAP-Ansatz verglichen, um einerseits die Anwendbarkeit des ITRAP zu überprüfen und andererseits die Differenz zwischen klassischen Ergebnissen und denen des ITRAP herauszustellen.

4.2.1 Anwendungsfall: Ein international agierender Application Service Provider

Für die weitere Evaluierung des ITRAP wird ein realer Anwendungsfall herangezogen. Dabei soll ein Ausschnitt der Systemlandschaft eines Application Service Provider hinsichtlich Verfügbarkeit und Kosten optimiert werden. Durch die Abbildung der Spezifika des Anwendungsfalls auf eine ITRAP-Definition kann die prinzipielle Anwendbarkeit des Konzepts nachgewiesen werden. Außerdem kann ein Vergleich der verschiedenen Lösungsmethoden untereinander sowie des ITRAP mit einem klassischen RAP-Ansatz durchgeführt werden. Letzteres soll verdeutlichen, dass die Anwendung eines ITRAP zu anderen Designvorschlägen als die eines klassischen RAP führt. Dieser Anwendungsfall wurde ebenfalls in [BST15] und [BST16] zur Demonstration von Teilfunktionalitäten des ITRAP genutzt.

Ein Application Service Provider (ASP) ist ein IT-Service-Provider, der seinen Kunden Zugang zu Applikationen über das Internet bietet. Der betrachtete ASP betreibt und administriert dabei Infrastruktur-, Hardware- und Softwaresysteme, um Kunden im universitären Umfeld Zugang zu Anwendungssystemen zu ermöglichen. Diese werden von den Anwendern zum Zwecke der Ausbildung, Forschung und Innovationsförderung genutzt. Die zur Verfügung gestellten Anwendungssysteme stammen aus dem SAP®-Lösungsportfolio. Beispiele dafür sind die SAP Business Suite (u.a. Enterprise Resource Planning – ERP), SAP Business ByDesign® (eine Software-as-a-Service-Anwendung) oder SAP HANA® (In-Memory-Technologien). Der ASP bedient fast 500 Kunden in mehr als 60 Ländern und ermöglicht damit über 150.000 Endnutzern Zugang zu den angebotenen Diensten.

Besondere Bedeutung erfährt dabei der Zugang zu ERP-Systemen, der zur Vermittlung von Grundkenntnissen in der computergestützten Bearbeitung von integrierten Geschäftsprozessen genutzt wird. Daher wird im Folgenden der Ausschnitt der ASP-Systemlandschaft betrachtet, der zur Erbringung dieser Dienstleistung genutzt wird. Für diesen Ausschnitt sollen Kosten und Verfügbarkeit des ERP-Service optimiert werden.

Aus diesem Grund wurde der ERP-Service als ITRAP formuliert. Dieser IT-Service wird von elf Subsystemen erbracht, die in Abbildung 4.25 dargestellt sind. Dabei wird von einer konstanten Performanceerwartung von 1 und einem Administrator-Stundenlohn von 40 € ausgegangen.

Herzstück des Service ist dabei das ERP-Subsystem: Hier werden SAP ERP 6.0 Systeme betrieben, auf die die Endnutzer zugreifen. Dazu werden Nutzeranfragen über ein

Proxy-Subsystem zu einem ERP-System mittels SAP-Routern geleitet.

Für den Betrieb der ERP-Systeme werden Server in einem ERP-Server-Subsystem gehostet, für das die Blades HP® ProLiant BL620c G7 bzw. BL980 G7 genutzt werden können. Die Zuordnung der ERP-Instanzen zu den Serverressourcen wird in einem Load-Balancer-Subsystem durch SAP Adaptive Computing Controller verwaltet.

Zur Datenverwaltung werden die ERP-Systeme mit einem Datenbank-Subsystem verbunden. Dort werden IBM® DB2® Datenbanksysteme betrieben. Diese Systeme werden vom Datenbank-Server-Subsystem mit Rechnerressourcen ausgestattet, in dem die Blades HP ProLiant BL460c G7 bzw. BL360 G7 genutzt werden können. Für die Speicherung und den Transfer gespeicherter Daten wird ein Storage-Subsystem (Storage Area Network – SAN) zur Verfügung gestellt. Hier stehen HP Storage Works EVA 6400-Systeme zur Verfügung. Das Speichernetzwerk wird dabei in einem Storage-Management-Subsystem verwaltet, in dem HP Storage Essentials Software-Systeme benutzt werden können.

Neben der Speicherverwaltung werden noch weitere Infrastrukturdienste benötigt. So werden Ethernet-Verkabelungen genutzt, um die Kommunikation zwischen den einzelnen Servern zu ermöglichen (LAN-Subsystem). Dieses LAN ist über ein WAN-Subsystem mit dem Internet verbunden. Hier können redundante ISP-Anschlüsse zusammen mit benötigten Netzwerkkomponenten genutzt werden. Alle Komponenten des Service sind zusätzlich von einer durchgehenden Stromversorgung abhängig. Diese wird durch das Strom-Subsystem bereitgestellt. Dabei können neben Anschlüssen an das regionale Stromnetz außerdem unterbrechungsfreie Stromversorgungen (USV) genutzt werden.

Für den ASP soll auf Basis der obigen Informationen eine Komponentenauswahl für jedes Subsystem erfolgen, sodass (sub)optimale Designs im Sinne der Verfügbarkeit und Kosten identifiziert werden können. Dazu müssen zusätzlich mögliche Redundanztypen sowie die Transitionen und Abhängigkeiten der Komponenten modelliert und parametrisiert werden.

Für diese Informationen standen Beschreibungen seitens des ASP sowie historische Daten nur eingeschränkt zur Verfügung. Außerdem sind Herstellerangaben zu Verfügbarkeitsstatistiken oft überoptimistisch [PWB07]. Diese Probleme sind jedoch allgemeiner Natur und erschweren generell die Evaluierung von verfügbarkeitsbezogenen Modellen [BST14b].

Um die fehlenden oder unzureichenden Parameter zu identifizieren, wurden daher auch statistische Daten aus Literaturquellen z.B. aus [MM11] und [SPW11] herangezogen. Weiterhin wurde auf historische Daten des Los Alamos National Lab[2] zugegriffen. Dort sind Ausfalldaten eines Clusters für den Zeitraum zwischen 1995 und 2011 gesammelt.

Während die Definition der Kostenparameter für Akquisition und Wiederherstellung aus Händlerangaben durchgeführt werden konnte, wurden für die Betriebskosten der Server die Ergebnisse zweier Benchmarks für die Leistungsaufnahme von Servern genutzt: Der SPECpower Benchmark[3] und der HP Power Advisor[4].

[2] `http://institute.lanl.gov/data/fdata`
[3] `http://www.spec.org/power_ssj2008/results/power_ssj2008.html`
[4] `http://www8.hp.com/de/de/products/servers/solutions.html?compURI=1439951`

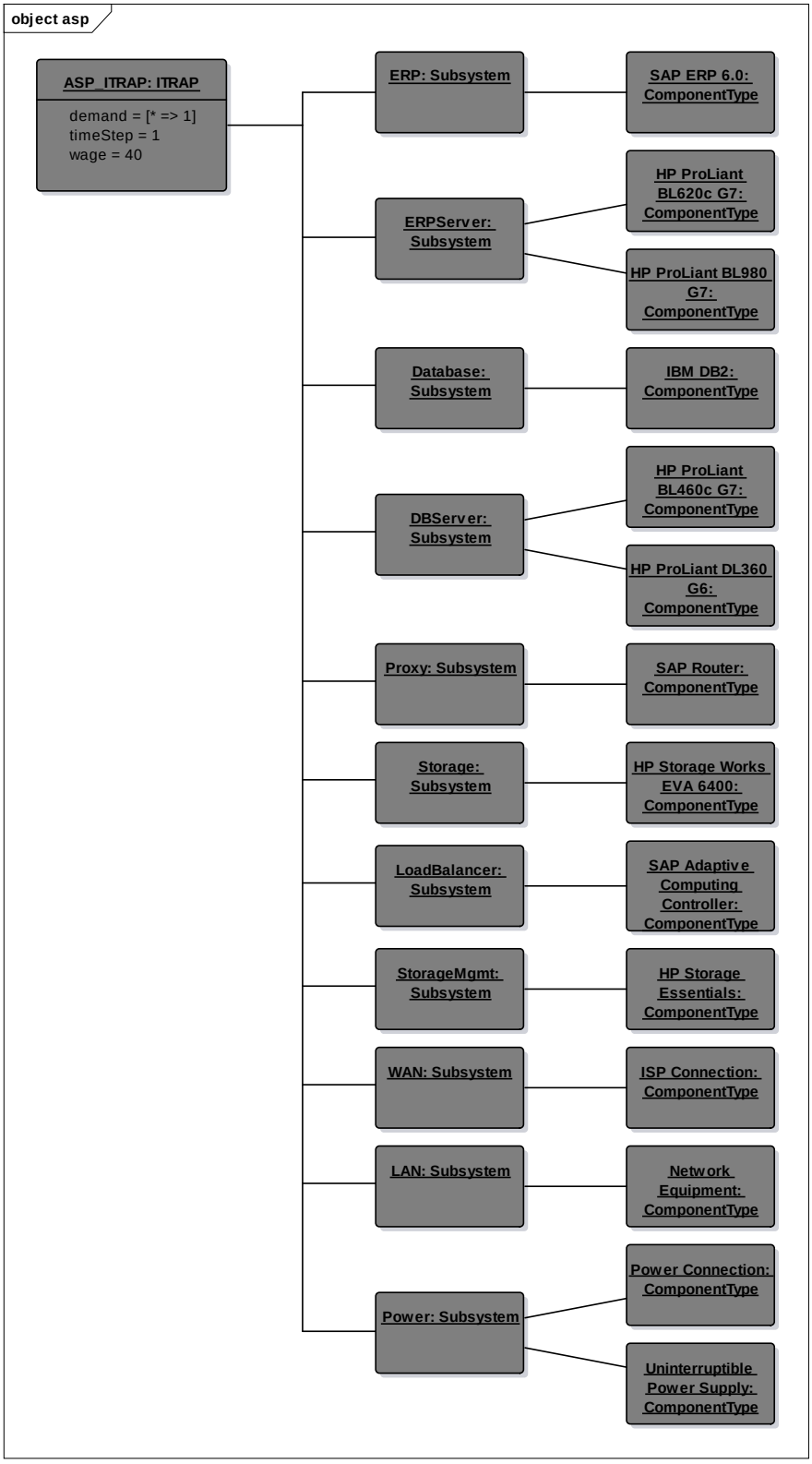

Abbildung 4.25: Benötigte Subsysteme sowie mögliche Komponenten für das ASP-Szenario

Aus den verschiedenen Informationsquellen konnten die oben modellierten Komponenten parametrisiert werden. Im Ergebnis wurden so Komponenten mit zwei bis vier möglichen Zuständen definiert. Für die meisten Komponenten wurde jeweils eine aktive und eine passive Redundanz modelliert. Auf Basis von Expertenbefragungen beim ASP konnten Abhängigkeiten wie imperfekte Übernahme und Ausfälle aufgrund gemeinsamer Ursachen definiert werden. So kann z.B. ein zusätzlicher WAN-Access-Point die Wahrscheinlichkeit eines Anschlussausfalls reduzieren. Eine andere Ausfallart beschreibt jedoch einen Ausfall des ISP, der durch redundante Anschlüsse nicht abgedeckt werden kann. Eine andere Abhängigkeit besteht zwischen Datenbank- und ERP-Systemen, wobei ein Datenbankausfall zu einem Absturz des ERP-Systems führen kann. Die ERP-Systeme selbst können zusätzlich von einem Ausfall aufgrund gemeinsamer Ursachen betroffen sein.

Auf die genaue Parametrisierung der einzelnen Komponenten wird aus Gründen der Übersichtlichkeit an dieser Stelle verzichtet. Eine Aufstellung dieser Daten ist jedoch in Anhang A gegeben. Beispielhaft ist die Parametrisierung in der grafischen Modellierspra-che in den Abbildungen 4.26 und 4.27 für zwei Komponenten dargestellt.

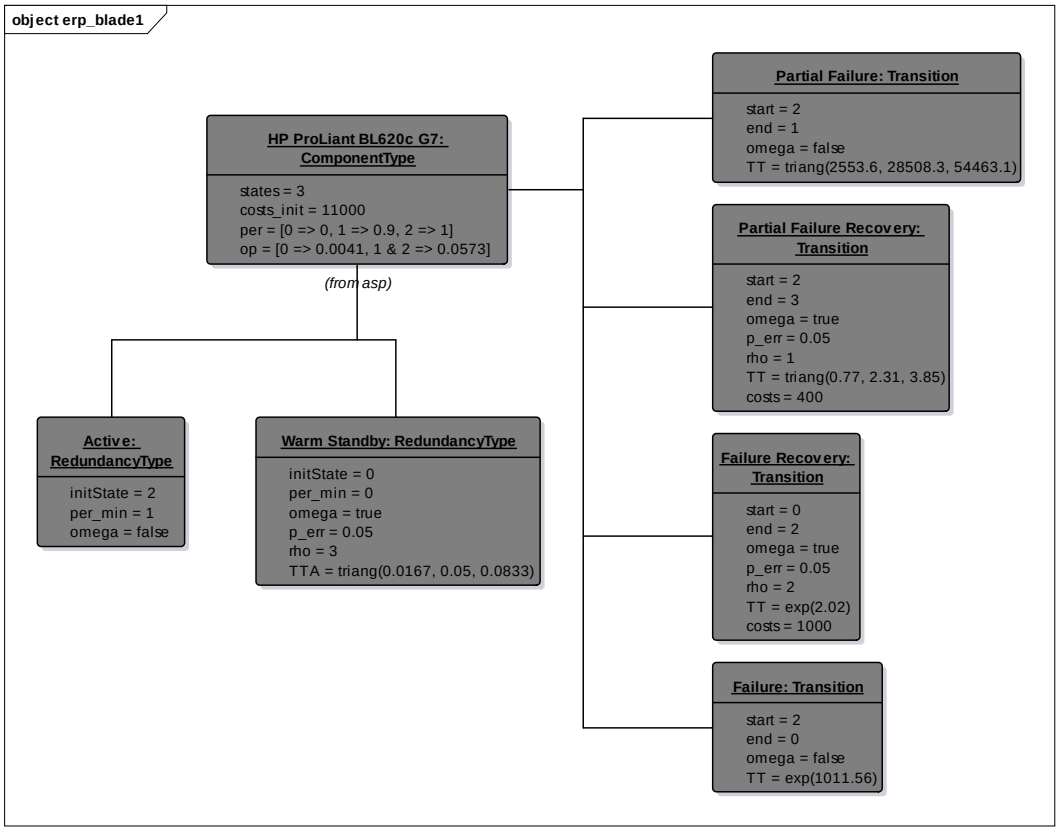

Abbildung 4.26: Parametrisierung der ersten Komponente im ERP-Subsystem

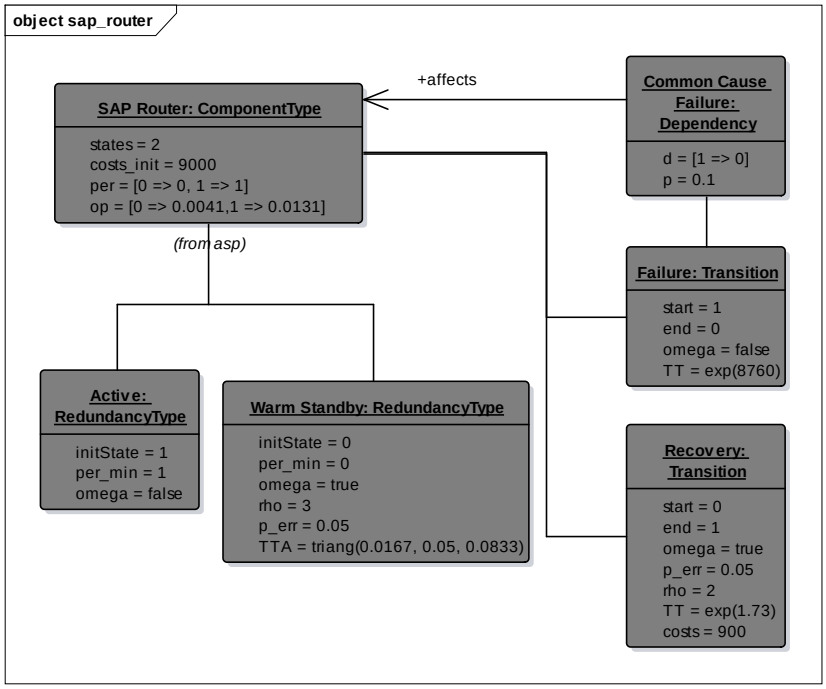

Abbildung 4.27: Parametrisierung der Komponente im Proxy-Subsystem

4.2.2 Analyse der Effizienz

Zunächst wird die Effizienz der implementierten Lösungsmethoden betrachtet. Diese Effizienz ist dann gegeben, wenn die Lösungsverfahren sich einer optimalen Lösung in einer vertretbaren Zeit annähern. Diese Zeit wird von der Problemkomplexität bestimmt, wobei die Ergebnisqualität der Verfahren schon bei geringer Durchforstung des Suchraumes akzeptabel sein sollte. Diese Qualität sollte außerdem mit steigendem Zeitaufwand besser werden. Als Vergleichswert und für die Identifikation des Optimums kann eine komplette Durchforstung des gesamten Suchraumes durchgeführt werden.

Diese Durchforstung ist jedoch nur für einen begrenzten Suchraum praktikabel. Daher wurden für das Effizienzexperiment die Anzahl an Komponenten, die in einem Subsystem verwendet werden dürfen, auf drei beschränkt. Weiterhin werden exemplarisch nur die Subsysteme der Gateways, der Load-Balancer, der ERP-Systeme und der ERP-Server betrachtet. Mit der Bedingung, dass die erste Komponente eines Subsystems in aktiver Redundanz betrieben wird, besteht der Suchraum damit aus 16.200 eindeutigen Lösungen.

Das Effizienzexperiment wird für das Optimierungsproblem (3), also der mehrkriteriellen Optimierung von Verfügbarkeit und Kosten, durchgeführt. Im Optimierungsproblem (3) soll eine Menge nicht-dominierter Lösungen identifiziert werden, die so genannte Pareto-Front. Dies sind – vereinfacht gesagt – alle Lösungen, für die keine andere Lösung mit geringeren Kosten und höherer Verfügbarkeit existiert. Diese Lösungen können in quadratischem Aufwand aus einer Menge von Kandidaten bestimmt werden (vgl. [DAPM00]). Jede verwendete Meta-Heuristik besitzt mindestens einen Parameter, um den

Zeitverbrauch der Suche zu bestimmen. Diese Parameter können so gewählt werden, dass der „quadratische“ Zeitverbrauch der Identifikation der Pareto-Front aus allen möglichen 16.200 Lösungen nicht erreicht wird. So können die Ergebnisqualität der Algorithmen für eine vertretbaren Zeitverbrauch und die Art der Annäherung an ein Optimum diskutiert werden.

Die Art der Ergebnisse als Menge nicht-dominierter Lösungen erschwert jedoch den direkten Vergleich, da Pareto-Fronten verglichen werden müssen. Nach [ZDT00] kann die Qualität einer Pareto-Front in drei Kriterien unterteilt werden, solang eine optimale Pareto-Front bekannt ist:

(i) Der Abstand zwischen der optimalen und der erhaltenden Pareto-Front soll minimiert werden.

(ii) Das Ausmaß der Pareto-Front in allen Zieldimensionen soll maximiert werden.

(iii) Die Verteilung der Elemente der Pareto-Front sollte möglichst gleichverteilt sein.

Die Vereinigung dieser drei Kriterien in einer einzelnen Metrik ist jedoch schwierig und kann zu missverständlichen Ergebnissen führen [SCC02]. Daher werden für die drei Kriterien jeweils einzelne Metriken betrachtet.

Für das Kriterium (i) kann die mittlere Idealdistanz herangezogen werden [SCC02]:

$$MID(\mathbf{X}, \mathbf{Opt}) = \frac{1}{|\mathbf{X}|} \sqrt{\sum_{X \in \mathbf{X}} \min_{Y \in \mathbf{Opt}} \|X - Y\|}$$

Dabei wird die mittlere minimale Euklidische Distanz zwischen den Punkten einer berechneten Front $\mathbf{X}$ und der optimalen Front $\mathbf{Opt}$ im Ergebnisraum berechnet. Dieser Wert sollte im besten Fall 0 annehmen. Dabei können jedoch Fronten mit sehr wenigen (optimalen) Elementen Fronten mit vielen Elementen vorgezogen werden, wenn einige davon nicht optimal sind. Daher wird als weitere Metrik die maximale Streuung betrachtet (engl. *maximum spread*) [CRNK12]:

$$MS(\mathbf{X}) = \frac{1}{\sqrt{2}} \sqrt{\left(\max_{X \in \mathbf{X}} A(X) - \min_{X \in \mathbf{X}} A(X) \right)^2 + \left(\max_{X \in \mathbf{X}} C(X) - \min_{X \in \mathbf{X}} C(X) \right)^2}$$

Damit ist dieser Wert ein Maß für die Länge der Diagonalen des minimalen Rechtecks im Ergebnisraum, welches alle Elemente der Pareto-Front enthält. Der Wert misst somit die Erfüllung des Kriteriums (ii) und sollte maximiert werden. Auch hier kann jedoch kritisiert werden, dass eine Front mit nur zwei Elementen optimal scheinen kann, solang diese den Grenzen der optimalen Pareto-Front entsprechen. Auch könnten Fronten, in denen z.B. der gleiche Verfügbarkeitswert durch viel höhere Kosten als in einer Vergleichsfront erreicht wird, missverständlich bevorzugt werden.

Um dieser Fehleinschätzung entgegen zu wirken, sollte zusätzlich das Kriterium (iii) herangezogen werden. Da die optimale Pareto-Front bekannt ist, wird diese als Idealverteilung definiert. Daher sollten einerseits möglichst viele Elemente aus der optimalen

Pareto-Front identifiziert werden, andererseits möglichst wenige Elemente als Ergebnis ausgegeben werden, die nicht Teil der optimalen Front sind. In Anlehnung an das Gebiet des Information Retrieval werden diese beiden Aspekte als Trefferquote (engl. *recall*) und Genauigkeit (engl. *precision*) bezeichnet. Das harmonische Mittel beider Werte wird als F1-Maß bezeichnet:

$$precision(\mathbf{X}, \mathbf{Opt}) = \frac{|\{X \in \mathbf{X} : X \in \mathbf{Opt}\}|}{|\mathbf{X}|}$$
$$recall(\mathbf{X}, \mathbf{Opt}) = \frac{|\{X \in \mathbf{X} : X \in \mathbf{Opt}\}|}{|\mathbf{Opt}|}$$
$$F_1 = 2\frac{precision \cdot recall}{precision + recall}$$

Im durchgeführten Experiment wurden die Verfügbarkeit und die Kosten jedes der 16.200 Designs per Simulation a priori bestimmt und in ein Repository gespeichert. Daraufhin wurde die erste Pareto-Front aus diesen Designs mit quadratischem Aufwand bestimmt (Durchforstung – DF).

Die drei Metriken MID, MS und F_1 werden dann als Performance-Indikatoren der jeweiligen Algorithmen berechnet. Dafür wurden eine Zufallssuche (ZS), der GA, die Tabu-Suche, die PSO, die SSO sowie die ABC mit verschiedenen Parametern eingesetzt, um den Zeitaufwand zu steuern. Zur Verhinderung von Verzerrungseffekten zwischen den Ergebnisdimensionen in den Metriken werden die Ergebniswerte auf Basis der optimalen Front normiert.

Einige Ergebnisse des Effizienzexperimentes sind in Tabelle 4.14 dargestellt, eine vollständige Auflistung ist in Anhang B gegeben (Tabelle B.1). Die Experimente wurden in AnyLogic auf einem Intel® Core™ i5-3570K @ 3,4 GHz ausgeführt. Alle Verfahren wurden dabei in 100 Iterationen durchgeführt, um 0,99-Konfidenzintervalle zu erhalten. Die Zufallssuche entspricht dabei der Durchforstung, wobei nur eine begrenzte Anzahl an zufälligen Lösungskandidaten (vgl. Abschnitt 3.4.3) betrachtet wird ($subSize = 1, opNum = 1$). Dies verdeutlicht die Qualität der Initialkandidaten für jeden Algorithmus. Für den Genetischen Algorithmus wurde der Parameter $\mu = \lambda$ ($maxGen = 100, p_{mut} = 0,3$) und für die Tabu-Suche $maxIt$ ($neighborhoodSize = 20$, $tabuSize = 30$) zur Steuerung des Zeitaufwands genutzt. Der Zeitverbrauch der PSO ($maxIt = 60, c = 0,08, l = 0,22, g = 0,7$), SSO ($maxIt = 60, c = 0,63, l = 0,22, g = 0,1$) sowie der ABC ($maxIt = 100$, $limit = 5$) wird über den Parameter $size$ beeinflusst (vgl. Abschnitt 3.4.5 für eine Übersicht der Steuerparameter).

Da die Tabu-Suche als lokale Suche definiert ist, reagiert der Algorithmus ab einer bestimmten Zeit (ca. ab $maxIt = 100$) robust gegen eine Erhöhung der maximalen Iterationen. In diesem Zustand scheint die Lösungsmethode in ein lokales Optimum konvergiert zu sein. Daher konnte für hohe Iterationszahlen keine zu den anderen Algorithmen vergleichbare Performance erreicht werden. Aus diesem Grund wurde die Tabu-Suche ab $maxIt = 100$ in mehreren Durchläufen statt für mehr Iterationen ausgeführt, wobei die resultierende Menge nicht-dominierter Lösungen aus den aggregierten Ergebnissen der ein-

Tabelle 4.14: Einige Ergebnisse des Effizienzexperiments

Alg.	Parameter	F_1	MID	MS	Zeit in s
DF		1,00	0,000	1,00	82,52
ZS	$n = 1000$	$[0,32; 0,35]$	$[0,025; 0,029]$	$[0,74; 0,79]$	$[1,08; 1,20]$
	$n = 5000$	$[0,43; 0,45]$	$[0,019; 0,022]$	$[0,77; 0,82]$	$[9,03; 9,15]$
	$n = 15000$	$[0,51; 0,54]$	$[0,016; 0,019]$	$[0,77; 0,82]$	$[77,00; 77,68]$
GA	$\mu = \lambda = 10$	$[0,25; 0,29]$	$[0,025; 0,035]$	$[0,72; 0,83]$	$[0,52; 0,54]$
	$\mu = \lambda = 100$	$[0,58; 0,62]$	$[0,012; 0,018]$	$[0,84; 0,95]$	$[5,55; 5,65]$
	$\mu = \lambda = 1000$	$[0,86; 0,89]$	$[0,005; 0,009]$	$[0,94; 1,01]$	$[84,53; 85,48]$
TS	$maxIt = 10$	$[0,12; 0,18]$	$[0,045; 0,061]$	$[0,51; 0,59]$	$[0,18; 0,23]$
	$maxIt = 100$	$[0,35; 0,43]$	$[0,021; 0,026]$	$[0,64; 0,73]$	$[2,98; 3,72]$
TS_2	$maxIt = 100$	$[0,51; 0,58]$	$[0,013; 0,017]$	$[0,73; 0,82]$	$[6,59; 7,73]$
TS_{20}	$maxIt = 100$	$[0,96; 0,98]$	$[0,000; 0,001]$	$[0,98; 1,00]$	$[66,35; 69,86]$
PSO	$size = 20$	$[0,23; 0,31]$	$[0,024; 0,033]$	$[0,70; 0,72]$	$[0,76; 0,80]$
	$size = 100$	$[0,58; 0,65]$	$[0,006; 0,008]$	$[0,71; 0,73]$	$[3,11; 3,15]$
	$size = 2000$	$[0,87; 0,89]$	$[0,003; 0,007]$	$[0,72; 0,75]$	$[92,30; 94,26]$
SSO	$size = 20$	$[0,34; 0,40]$	$[0,021; 0,029]$	$[0,71; 0,75]$	$[0,80; 0,82]$
	$size = 100$	$[0,69; 0,73]$	$[0,011; 0,015]$	$[0,74; 0,79]$	$[3,48; 3,56]$
	$size = 2000$	$[0,93; 0,96]$	$[0,004; 0,008]$	$[0,81; 0,88]$	$[78,70; 79,23]$
ABC	$size = 10$	$[0,12; 0,17]$	$[0,035; 0,041]$	$[0,71; 0,77]$	$[1,13; 1,18]$
	$size = 100$	$[0,45; 0,50]$	$[0,014; 0,016]$	$[0,81; 0,86]$	$[8,48; 8,58]$
	$size = 800$	$[0,82; 0,85]$	$[0,005; 0,007]$	$[0,86; 0,92]$	$[72,89; 74,34]$

zelnen Durchläufe gebildet wird. In Tabelle 4.14 ist dies durch die Beschriftung TS_k für k Durchläufe gekennzeichnet. Aufgrund der hohen Diversität der Resultate der einzelnen Durchläufe zeigte sich dabei, dass die Qualität der Ergebnisse durch dieses Vorgehen massiv erhöht werden konnte.

Auf Basis der Ergebnisse dieses Tests konnten logarithmische Regressionsfunktionen (Bestimmtheitsmaß $R^2 > 0,9$, Standardfehler $se < 0,05$) für die Performance der Lösungsalgorithmen abgeleitet werden. Diese sind in Abbildung 4.28 für die mittlere Idealdistanz, in Abbildung 4.29 für die maximale Streuung und in Abbildung 4.30 für das F1-Maß visualisiert. Alle Metriken sind dabei abhängig von NTC, dem normalisierten Zeitverbrauch auf Basis der Dauer der Durchforstung, dargestellt. Ein Wert von 1 repräsentiert dabei den Zeitverbrauch der Durchforstung. Dabei wurde die Skalierung der Algorithmen über die in Tabelle 4.14 beschriebenen Parameter gesteuert. Da ein Tuning zwischen den Parametern, die maßgeblich den Zeitverbrauch bestimmen (z.B. für den GA neben der Populationsgröße auch die Anzahl maximaler Generationen), nicht durchgeführt wurde, sind die Kurven als untere Schranken der Performance zu verstehen.

Wie in den Ergebnissen zu sehen, besitzt die definierte Zufallssuche die geringste Performance. Die Meta-Heuristiken nähern sich hingegen den optimalen Werten mit steigendem Zeitverbrauch an. Die Tabu-Suche zeigt in allen drei Metriken eine hohe Performance. Dies konnte jedoch nur durch Aggregation der Fronten mehrerer Läufe erreicht werden.

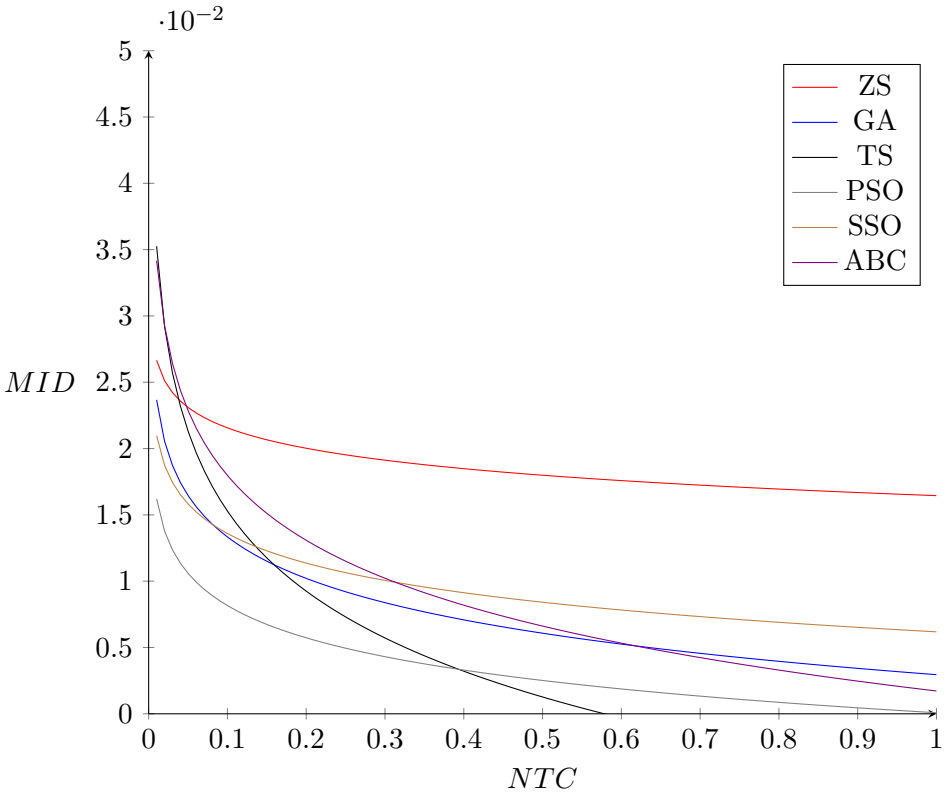

Abbildung 4.28: Mittlere Idealdistanz der verschiedenen Lösungsverfahren relativ zur Durchforstung (logarithmische Annäherung)

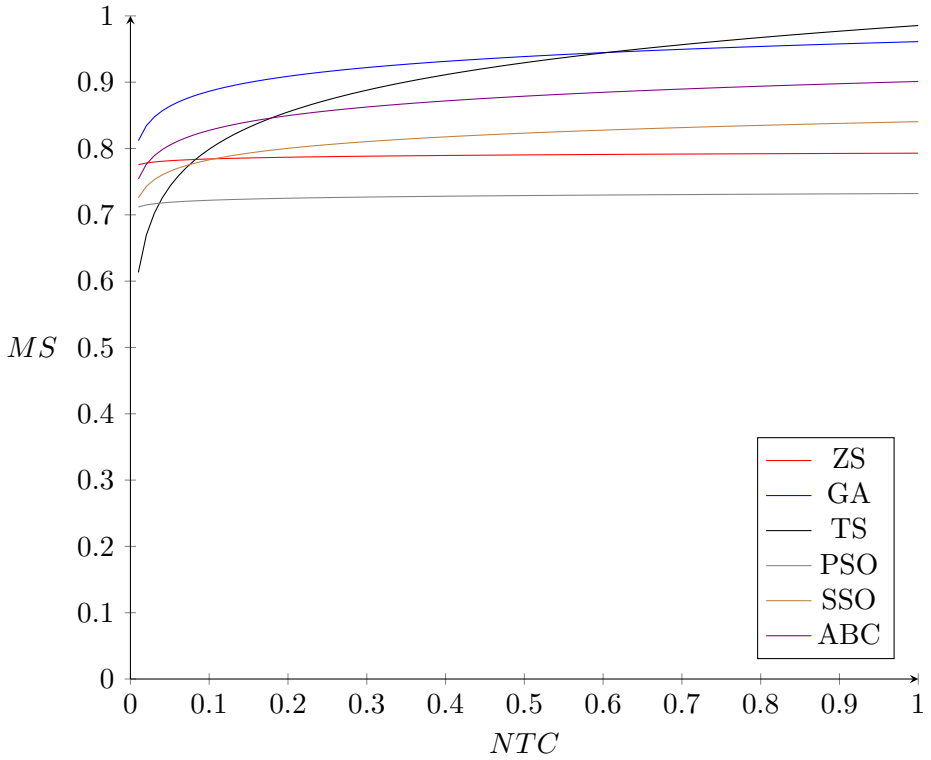

Abbildung 4.29: Maximale Streuung der verschiedenen Lösungsverfahren relativ zur Durchforstung (logarithmische Annäherung)

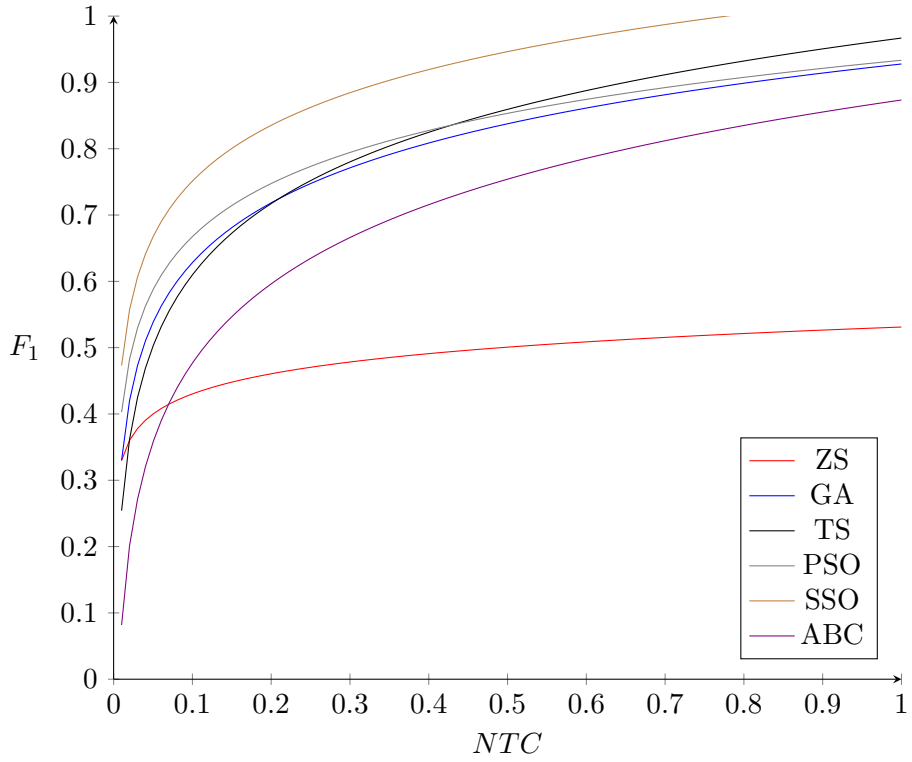

Abbildung 4.30: F1-Maß der verschiedenen Lösungsverfahren relativ zur Durchforstung (logarithmische Annäherung)

Die Partikelschwarmoptimierung zeigt sowohl beim F1-Maß als auch bei der mittleren Idealdistanz eine gute Leistung, fällt jedoch bei der maximalen Streuung deutlich hinter die anderen Algorithmen zurück. Dies zeigt, dass die PSO den Bereich hoher Verfügbarkeit kaum durchforsten kann, aber die Initialregion gut ausbeutet. Bei der SSO ist dieser Effekt in schwächerer Form ebenfalls zu beobachten. Dennoch entsprechen die Ergebnisse der SSO am ehesten den Individuen der optimalen Pareto-Front, wie das F1-Maß zeigt. Der Genetische Algorithmus und die ABC verhalten sich über die drei Kriterien unterdurchschnittlich, wobei der GA Vorteile gegenüber der Künstlichen Bienenkolonie besitzt.

Insgesamt sind alle Lösungsalgorithmen in der Lage, sich in vertretbarer Zeit einer optimalen Lösung anzunähern. Dies wird einerseits durch den Vergleich zu der definierten Zufallssuche deutlich, die die Initialpopulation aller Algorithmen bestimmt. Hier sind also alle Verfahren mehr oder weniger imstande, den Initalbereich auszubeuten und gleichzeitig andere Bereiche des Suchraums zu erkunden, da die Performance der Zufallssuche deutlich übertroffen wird. Andererseits zeigen alle Algorithmen im Bereich niedriger Zeitverbräuche eine signifikante Steigerung der Performance im Vergleich zu Erhöhung des Zeitaufwandes. Dies belegt, dass die Qualität der Optimierungsergebnisse auch akzeptabel ist, wenn nur eine zum Suchraum verglichen geringe Anzahl an Lösungskandidaten betrachtet wird.

Die Ergebnisse dieses Experiments legen damit nahe, dass die nicht-funktionale Anforderung nach Effizienz (NF2) durch das ITRAP erfüllt wird. Durch die Identifikation (sub)optimaler Lösungen können auch die funktionalen Anforderungen nach Optimierung

von Kosten und Verfügbarkeit (F1) als realisiert angesehen werden.

Damit konnte nachgewiesen werden, dass das ITRAP alle gestellten Anforderungen an ein RAP für IT-Service-Design erfüllt. In dem folgenden Teil dieses Kapitels wird die Anwendbarkeit des ITRAP im Beispielszenario untersucht. Dabei wird die Performance der Lösungsalgorithmen sowie die Differenz zu einem klassischen RAP analysiert.

4.2.3 Experimenteller Vergleich

Nachdem die Effizienz der Lösungsalgorithmen in einem Teilproblem des Anwendungsfalls getestet wurde, wird in diesem Abschnitt das komplette Problem des ASP behandelt. Dabei werden die drei definierten Optimierungsprobleme mit verschiedenen Nebenbedingungen adressiert. Der Vergleich der Ergebnisse mit einem klassischen RAP-Ansatz soll verdeutlichen, dass unterschiedliche Design-Empfehlungen vom ITRAP identifiziert werden. Eine ausführliche Diskussion der erhaltenden Ergebnisse findet im anschließenden Abschnitt 4.2.4 statt.

Der genutzte klassische Ansatz ist dabei an das RAP aus [CS96a] angelehnt, welches z.B. in [KKSC03] oder [Yeh14] für Vergleichswerte genutzt wird. Dabei können Subsysteme aus heterogenen Komponenten in aktiver Redundanz bestehen. Die Charakteristika der einzelnen Komponenten wie z.B. Verfügbarkeit, Betriebskosten für ein Jahr, etc. wurden aus den isolierten Zustandsverteilungen a priori errechnet. Damit werden unabhängige Ausfälle angenommen, sodass die Werte für Verfügbarkeit und Kosten eines Lösungskandidaten kombinatorisch berechnet werden können. Somit kann das klassische RAP weder Standby-Redundanz noch Abhängigkeiten oder (fehlerbehaftete) Administratorinteraktion abbilden. Um die Kosten von RAP- mit denen von ITRAP-Kandidaten vergleichen zu können, werden für jeden RAP-Kandidaten Personalkosten eines Administrators für ein Jahr addiert.

Wie bereits in Abschnitt 4.1.1 erläutert, werden die unterschiedlichen Ansätze durch verschiedene Fitness-Evaluationen abgebildet. Daher können alle definierten Lösungsalgorithmen auch für das klassische RAP eingesetzt werden. Dabei wurden die Parameter so gewählt, dass ca. 60.000 Individuen per Iteration evaluiert werden. Da alle Verfahren stochastische Elemente besitzen, werden zehn Iterationen jedes Algorithmus durchgeführt.

Tabelle 4.15: Parameterauswahl für die Optimierungsexperimente

Algorithmus	Parameter
Initialisierung	$subSize = 1$, $opNum = 1$
Simulation	$replications = 100, t = 8760$ (ein Jahr)
GA	$\mu = 30, \lambda = 60, maxGen = 100, p_{mut} = 0,3, tSize = 6, p_t = 0,98$
NSGAII	$\mu = \lambda = 60, maxGen = 100, p_{mut} = 0,3$
TS	$maxIt = 150, nSize = 20, tSize = 30$
PSO	$maxIt = 60, swarmSize = 100, c = 0,08, l = 0,22, g = 0,7$
SSO	$maxIt = 60, swarmSize = 100, c = 0,63, l = 0,22, g = 0,1$
ABC	$maxIt = 100, size = 30, limit = 5$

Wie beim Effizienzexperiment wird ein Repository aller simulierten Lösungen gespeichert, um bessere Vergleichbarkeit herzustellen. Die Parameter der Experimente und der jeweiligen Meta-Heuristiken sind in Tabelle 4.15 gegeben.

Optimierungsproblem (1) – Maximierung der Verfügbarkeit In den folgenden Szenarien soll die Verfügbarkeit des ERP-Service-Designs maximiert werden, wobei verschiedene Kostenbedingungen definiert werden. Diese wurden so gewählt, dass mit steigender Kostenschranke die Qualität der initial erstellten Lösungen, gesteuert über die konstanten Parameter $opNum$ und $subSize$, abnimmt:

- Szenario 1: Kostenschranke $C = 525.000$ €,
- Szenario 2: Kostenschranke $C = 550.000$ € und
- Szenario 3: Kostenschranke $C = 600.000$ €.

In den Tabellen 4.16, 4.17 und 4.18 werden Statistiken der erhaltenen Ergebnisse für die drei Szenarien präsentiert; für die Berechnung der Fitness gilt $NTF_C = 1.000$ €. Dargestellt sind die Fitness des Optimums der zehn Läufe, die mittlere Fitness, die Standardabweichung der Fitness sowie der Anteil unzulässiger Lösungen für die Ergebnisse der jeweiligen Algorithmen. In jeder Tabelle ist der beste Wert der Algorithmen mit hellgrau, der schlechteste mit dunkelgrau hinterlegt.

Tabelle 4.16: Vergleich der Optimierungsergebnisse im Szenario 1 des Optimierungsproblems (1)

	Bester	Mittelwert	Std.Abw.	Unzulässig
GA	0,9960234	0,3730106	1,346	20%
TS	0,9960234	-136,593	179,29	50%
SSO	0,9960234	0,9960220	0.00001	0%
PSO	0,9960234	0,9949752	0,00086	0%
ABC	0,9960234	0,9949982	0,00077	0%

Tabelle 4.17: Vergleich der Optimierungsergebnisse im Szenario 2 des Optimierungsproblems (1)

	Bester	Mittelwert	Std.Abw.	Unzulässig
GA	0,9982479	0,9962601	0,00332	40%
TS	0,9983715	0,9940778	0,00373	70%
SSO	0,9984140	0,9983161	0,00008	0%
PSO	0,9982235	0,9977528	0,00059	0%
ABC	0,9984000	0,9981458	0,00023	0%

Es wird deutlich, dass alle Algorithmen in der Lage sind, vergleichbare Bestwerte zu erzielen. Die Partikelschwarmoptimierung produziert die niedrigsten Bestwerte, ist jedoch stabiler als die Tabu-Suche, die starke Schwankungen in der Ergebnisqualität besitzt. Ein

Tabelle 4.18: Vergleich der Optimierungsergebnisse im Szenario 3 des Optimierungsproblems (1)

	Bester	Mittelwert	Std.Abw.	Unzulässig
GA	0,9997960	0,9992391	0,00062	30%
TS	0,9998468	0,9993592	0,00057	40%
SSO	0,9998341	0,9997964	0,00003	20%
PSO	0,9997910	0,9987571	0,00066	0%
ABC	0,9997961	0,9997573	0,00004	40%

zusätzlicher Beleg dafür ist der Anteil unzulässiger Lösungen und die niedrigen Mittelwerte. Die TS-Performance steigt jedoch mit höherer Kostenschranke. Die Vereinfachte Schwarmoptimierung zeigt das stabilste Verhalten und wird im Bestwert nur in Szenario 3 von der Tabu-Suche dominiert.

Die Künstliche Bienenkolonie besitzt eine stabile und hohe Ergebnisqualität und wird nur von der SSO im Mittelwert übertroffen. Die hohen Anteile unzulässiger Lösungen besonders in Szenario 3 entstehen durch Kandidaten mit minimaler Verletzung der Kostenschranke. Der Genetische Algorithmus ist im Bestwert vergleichbar mit der PSO (wenn auch leicht besser), erreicht aber nicht deren Stabilität.

Tabelle 4.19: Gegenüberstellung der fittesten Lösungskandidaten des ITRAP und des klassischen RAP für das Optimierungsproblem (1)

	Szenario 1 525.000 €		Szenario 2 550.000 €		Szenario 3 600.000 €	
	ITRAP	RAP	ITRAP	RAP	ITRAP	RAP
Admin	1		1		1	
Router	1	1	1+1	1	1+1	2
ACC	1+1	2	1+1	2	2+1	3
ERP	2	2	2+1	2	2+1	2
ERP Server	1	1	1+1	2	1+1	2
Storage-Mgmt	1+1	3	1+1	2	2	2
DB	1+1	4	1+1	1	3+2	3
DB Server	1+1	1	1+1	2	2	2
Storage	1	1	1	1	2	2
LAN	2	2	2	2	2	2
WAN	1	1	1	1	1	1
Strom	1	1	1	1	1+1	1
A	0,99602	0,99486	0,99841	0,99864	0,99985	0,99992
C	524.648	524.548	547.305	550.104	600.014	599.567

In Tabelle 4.19 sind die optimale Ergebnisse von ITRAP und klassischem RAP aus allen Lösungsmethoden gegenübergestellt. Dabei repräsentieren die Zahlen die Anzahl benutzter Komponenten pro Subsystem, ein Eintrag der Form $x + y$ beschreibt ein System mit x aktiven und y passiven Komponenten. Da passive Redundanz nur im ITRAP modelliert wurde, werden in den RAP-Lösungen alle Komponenten aktiv betrieben. In Szenario

2 ist die fitteste Lösung des RAP, in Szenario 3 die des ITRAP, eine unzulässige Lösung mit minimaler Verletzung der Kostenschranke (schwarz hinterlegt).

Zunächst kann bemerkt werden, dass die Anzahl an Komponenten mit steigender Kostenschranke immer stärker zwischen den ITRAP- und RAP-Lösungen differiert. Werden im ersten Szenario noch mit 17 Komponenten zwei weniger in der ITRAP-Lösung verwendet, steigt die Anzahl in den ITRAP-Lösungen stärker in den weiteren Szenarien an (21 ggü. 17 in Szenario 2 und 27 ggü. 22 in Szenario 3). Dies wird durch die Nutzung der Standby-Redundanz ermöglicht, die deutliche Einsparungen in den Betriebskosten erzielt. In Szenario 1 kann daher eine höhere Verfügbarkeit durch das ITRAP erreicht werden. Mit steigender Kostenschranke fällt die Fitness jedoch hinter die der RAP-Lösungen. Hier macht sich die Modellierung von fehlerbehafteter Interaktion und Abhängigkeiten bemerkbar.

Optimierungsproblem (2) – Minimierung der Kosten Im zweiten Optimierungsproblem sollen die Kosten eines Designs minimiert werden, wobei eine minimale Verfügbarkeit erreicht werden muss. Auch hier werden drei Szenarien mit verschiedenen Nebenbedingungen formuliert:

- Szenario 1: $A = 0.995$ ($NFT_A = 0.0001$),
- Szenario 2: $A = 0.999$ ($NFT_A = 0.0001$) und
- Szenario 3: $A = 0.9999$ ($NFT_A = 0.00001$).

In den Tabellen 4.20, 4.21 und 4.22 sind die Kennzahlen für die Ergebnisqualität der Lösungsalgorithmen im ITRAP für die drei Szenarien des Problems (3) dargestellt. Auch hier markieren die hellgrau hinterlegten Zahlen die besten Werte, während die dunkelgrau hinterlegten Zahlen die schlechtesten Werte unter den Algorithmen darstellen.

Tabelle 4.20: Vergleich der Optimierungsergebnisse im Szenario 1 des Optimierungsproblems (2)

	Bester	Mittelwert	Std.Abw.	Unzulässig
GA	-521641,98	-533940,20	10909,72	0%
TS	-521641,98	-545462,04	17577,65	30%
SSO	-521641,98	-521641,98	0	0%
PSO	-521641,98	-523673,03	3591,02	0%
ABC	-521641,98	-524564,18	1541,90	10%

Zunächst kann festgestellt werden, dass alle Algorithmen dasselbe Optimum in Szenario 1 identifizieren. Dabei zeigt die SSO das stabilste Verhalten und findet diese Lösung in jedem der zehn Läufe. Auch in Szenario 2 überzeugt die SSO mit der besten und stabilsten Ergebnisqualität. Die SSO-Performance sinkt jedoch stark ab in Szenario 3, wo nur die PSO schlechtere Ergebnisse produziert. Für letzteres Verfahren kann ebenfalls eine sinkende Performance mit steigender Nebenbedingung beobachtet werden, nur in noch

Tabelle 4.21: Vergleich der Optimierungsergebnisse im Szenario 2 des Optimierungsproblems (2)

	Bester	Mittelwert	Std.Abw.	Unzulässig
GA	-560320,32	-583676,19	20960,60	0%
TS	-568728,62	-608641,47	23749,77	10%
SSO	-560320,32	-560499,83	234,27	0%
PSO	-560759,30	-601543,77	24983,98	0%
ABC	-562825,68	-565980,38	3299,04	10%

Tabelle 4.22: Vergleich der Optimierungsergebnisse im Szenario 3 des Optimierungsproblems (2)

	Bester	Mittelwert	Std.Abw.	Unzulässig
GA	-609189,15	-718740,98	153467,97	10%
TS	-609189,15	-626156,01	18689,87	10%
SSO	-630364,07	-748878,81	166970,33	20%
PSO	-1114368,34	-1582790,56	518007,03	90%
ABC	-625379,91	-656332,53	18471,05	10%

stärkerem Ausmaß als bei der SSO. Im letzten Szenario produziert die PSO in neun von zehn Fällen unzulässige Ergebnisse mit im Vergleich sehr niedriger Fitness.

Die Tabu-Suche zeigt wie im ersten Optimierungsproblem zunächst das instabilste Verhalten. In Szenario 3 sind die TS-Ergebnisse jedoch mit großem Abstand dominant im Mittelwert. Auch der Bestwert kann dort nur von dem GA erreicht werden. Der Genetische Algorithmus zeigt eine deutlich stärkere Performance als in Optimierungsproblem (1), in keinem Szenario wird der GA-Bestwert übertroffen. Die Qualität der ABC-Ergebnisse ist stabil und solide, Mittelwert und Standardabweichung werden in Szenario 2 nur von der SSO, in Szenario 3 nur von der TS übertroffen bzw. erreicht. Dennoch erreicht der ABC-Bestwert die Qualität vom Genetischen Algorithmus in den höheren Szenarien nicht.

Eine Gegenüberstellung der fittesten Kandidaten von RAP und ITRAP ist in Tabelle 4.23 dargestellt.

Zunächst ist auffallend, dass in den RAP-Lösungen für Szenario 1 und 2 die Verfügbarkeitsbedingung vergleichsweise stark verletzt wird (schwarz hinterlegte Werte). Dennoch besitzen diese Individuen eine höhere Fitness als zulässige Ergebnisse derselben Algorithmen. Dies mag bei deutlich niedrigeren Kosten sinnvoll erscheinen, kann jedoch, falls vom Entscheidungsträger gewünscht, mit Verringerung des NFT-Werts in höheren Generationen verhindert werden. Weiterhin wird wieder ein unterschiedlicher Anstieg der Komponentenanzahl von RAP- und ITRAP-Lösungen deutlich (15 ggü. 16 Komponenten in Szenario 1, 19 ggü. 18 in Szenario 2 sowie 28 ggü. 20 in Szenario 3). Daher steigen auch die Kosten der ITRAP-Lösungen im Vergleich zu den RAP-Lösungen.

Auffällig ist dabei vor allem die hohe Nutzung von ERP- und DB-Komponenten in Szenario 3. Für diese beiden Komponenten wurden Abhängigkeiten definiert, die eine Erhöhung der Verfügbarkeit durch zusätzliche Komponenten limitiert.

Tabelle 4.23: Gegenüberstellung der fittesten Lösungskandidaten des ITRAP und des klassischen RAP für das Optimierungsproblem (2)

	Szenario 1 0,995		Szenario 2 0,999		Szenario 3 0,9999	
	ITRAP	RAP	ITRAP	RAP	ITRAP	RAP
Admin	1		1		1	
Router	1	1	1	1	1+1	2
ACC	1+1	2	1+1	2	2+1	2
ERP	1	2	2	2	5	2
ERP Server	1	1	1+1	2	2	2
Storage-Mgmt	1+1	2	1+1	2	2	2
DB	1	2	1+1	2	4+1	2
DB Server	1+1	1	1+1	2	2	2
Storage	1	1	1+1	1	2	2
LAN	2	2	2	2	2	2
WAN	1	1	1	1	1	1
Strom	1	1	1	1	1+1	1
A	0,99503	0,99486	0,99906	0,99884	0,99991	0,99991
C	521.642	520.744	560.320	550.756	609.189	594.254

Optimierungsproblem (3) – Mehrkriterielle Optimierung In diesen Experimenten wird – ähnlich zum Effizienzexperiment – die gleichzeitige Optimierung von Verfügbarkeit und Kosten angestrebt. Auch hier müssen Mengen von nicht-dominierten Ergebnissen verglichen werden. Im Gegensatz zum Effizienzexperiment wird dies jedoch zusätzlich durch die Tatsache erschwert, dass die optimale Lösung aufgrund des unbeschränkten Suchraums nicht bekannt ist. Daher können Metriken wie das F1-Maß oder die mittlere Idealdistanz nicht berechnet werden. Zusätzlich zur maximalen Streuung werden daher Metriken genutzt, die keine Kenntnis einer optimalen Front voraussetzen (vgl. [CRNK12]).

Zunächst soll die resultierende Pareto-Front eine möglichst uniforme Verteilung der Lösungen zeigen. Damit sollen einem Entscheidungsträger möglichst viele, aber unterschiedliche Lösungen geboten werden. Daher soll die Anzahl an Lösungen in der Pareto-Front NPS (engl. *number of Pareto solutions*) maximiert werden:

$$NPS(\mathbf{X}) = |\mathbf{X}|$$

Die schiere Anzahl der Lösungen sagt jedoch allein nichts über deren Verteilung aus. Daher wird zusätzlich der Abstand zwischen den einzelnen Lösungen betrachtet:

$$d_X = \min_{Y \in \mathbf{X}, Y \neq X} (|A(X) - A(Y)| + |C(X) - C(Y)|)$$
$$\text{mit } \bar{d} = \frac{1}{|\mathbf{X}|} \sum_{X \in \mathbf{X}} d_X$$

Auf Basis dieser Abstände und des Mittelwerts lässt sich ein Abstandsmaß S (engl. *spa-*

cing) definieren:

$$S = \sqrt{\frac{1}{|\mathbf{X}| - 1} \sum_{X \in \mathbf{X}} (d_X - \bar{d})^2}$$

Diese Metrik ist damit ein Maß für die Streuung der Abstände zur nächsten Nachbarlösung. Daher sollte sie in einer guten Verteilung minimal sein. Wie auch bei der maximalen Streuung werden hier nur normalisierte Werte betrachtet, um Verzerrungseffekten vorzubeugen.

Weiterhin soll der Abstand der resultierenden Front zu einer optimalen Front minimiert werden. Da diese Front jedoch nicht bekannt ist, können nur die resultierenden Fronten in Bezug auf diese Fragestellung verglichen werden. Aus diesem Grund wird das Abdeckungsmaß CM (engl. *coverage metric*) betrachtet:

$$CM(\mathbf{X}, \mathbf{Y}) = \frac{|\{Y \in \mathbf{Y} : \exists X \in \mathbf{X}, X \succ Y\}|}{|\mathbf{Y}|}$$

Dies ist also der Anteil an Lösungen in der Front Y, die von Lösungen der Front X dominiert werden. Damit lässt sich zwar keine Aussage über den Abstand zu einem Optimum treffen, jedoch können zwei Fronten hinsichtlich Dominanz verglichen werden.

Im Folgenden werden Resultate der Lösungsalgorithmen beschrieben, die durch Aggregation der Ergebnisse von zehn Durchläufen gebildet wurden. Nach den Ergebnissen des Effizienzexperiments ist dies nur bei der Tabu-Suche für eine akzeptable Performance vonnöten, kann jedoch auch für die anderen Verfahren vorteilhaft sein. Dabei wurden die nicht-dominierten Lösungen über alle Läufe extrahiert. Für die Normalisierung wurden die Konstanten $C_{min} = 500.000, C_{max} = 1.500.000, A_{min} = 0,985, A_{max} = 1$ verwendet, da dieser Bereich fast alle Werte der identifizierten Pareto-Fronten enthält. In Tabelle 4.24 sind zunächst die Kennzahlen für die Anzahl der Elemente, das Abstandsmaß und die maximale Streuung dargestellt. Wieder sind die besten Werte hellgrau und die schlechtesten Werte dunkelgrau hinterlegt.

Tabelle 4.24: Kennzahlenvergleich für die ITRAP-Pareto-Fronten der jeweiligen Algorithmen für das Optimierungsproblem (3)

	NPS	S	MS
GA	46	0,100	1,078
TS	46	0,050	0,734
SSO	55	0,045	0,744
PSO	51	0,013	0,666
ABC	44	0,016	0,996

Die ABC produziert dabei die wenigsten Ergebnisse, wird jedoch im Abstandsmaß nur von der PSO und in der Streuung vom GA übertroffen. Die hohe Streuung des Genetischen Algorithmus bedingt auch ein hohes Abstandsmaß, da nur wenige Lösungskandidaten in der Region mit hohen Kosten identifiziert werden. Dies führt umgekehrt für die PSO

dazu, dass das niedrigste Abstandsmaß mit der schlechtesten Streuung erreicht wird. Die SSO identifiziert die meisten nicht-dominierten Elemente, Abstandsmaß und Streuung sind dabei mit denen der Tabu-Suche vergleichbar.

Die Maße für die gegenseitige Abdeckung der Fronten werden in Tabelle 4.25 präsentiert.

Tabelle 4.25: Matrix der Abdeckungsmetriken $CM(\mathbf{X}, \mathbf{Y})$ der Algorithmen für das Optimierungsproblem (3)

X \ **Y**	GA	TS	SSO	PSO	ABC	MW
GA		0,39	0,11	0,02	0,55	0,27
TS	0,15		0,04	0,04	0,57	0,20
SSO	0,33	0,48		0,18	0,64	0,40
PSO	0,41	0,48	0,22		0,77	0,47
ABC	0,15	0,17	0,04	0		0,07
MW	0,26	0,38	0,10	0,06	0,63	

So zeigt die Tabelle z.B., dass die GA-Lösungen 39 % der TS-Lösungen dominieren. Auf der anderen Seite werden nur 15 % der GA-Lösungen durch die TS-Lösungen dominiert. Für die jeweiligen Spalten und Zeilen wurde außerdem ein Mittelwert bestimmt (Spalte bzw. Zeile MW). Dabei zeigt sich, dass die Partikelschwarmoptimierung im Schnitt nicht nur die meisten Individuen dominiert, sondern ihre Lösungen auch am wenigsten dominiert werden. Auf der anderen Seite werden die ABC-Lösungen am häufigsten dominiert und dominieren selber am wenigsten.

Die SSO erreicht vergleichbare Abdeckungswerte wie die PSO, wenn auch nicht auf demselben Niveau. Wie die Einzelwerte zeigen, gibt es in beiden Fronten ca. ein Fünftel Kandidaten, die von Kandidaten der jeweils anderen Front dominiert werden. Die PSO-Lösungen werden dabei von keinem einzigen der ABC-Kandidaten dominiert. Umgekehrt gibt es keinen Algorithmus, dessen Ergebnisse komplett von denen eines anderen dominiert werden. Neben den starken Schwarmoptimierungen fällt der Genetische Algorithmus ab, zeigt jedoch bessere Performance als die Tabu-Suche.

Da die Abdeckungsmaße keine Aussage über die Differenz zweier Fronten treffen können, ist ein Ausschnitt aller Fronten in Abbildung 4.31 illustriert.

Die ABC- und TS-Front werden dabei deutlich von den anderen Algorithmen im Bereich bis ca. $C = 550.000$ dominiert. Mit höheren Kosten steigt dann die Performance der Tabu-Suche, sodass ihre Front ab $C = 600.000$ dominant ist. In dieser Region, die aus Übersichtlichkeitsgründen nicht dargestellt ist, produzieren PSO und SSO nur noch wenige Ergebnisse, was die niedrige Streuung erklärt. Im dargestellten Bereich dominieren diese Verfahren jedoch wechselseitig alle anderen Ergebnisse.

In einem weiteren Verarbeitungsschritt wurden die resultierenden Fronten aggregiert und eine neue Pareto-Front identifiziert. Indem dasselbe Vorgehen für die RAP-Lösungen gewählt wurde, können die von allen Algorithmen zusammen erstellten Ergebnisse für ITRAP und RAP verglichen werden. Ein Kennzahlenvergleich der Fronten ist in Tabelle

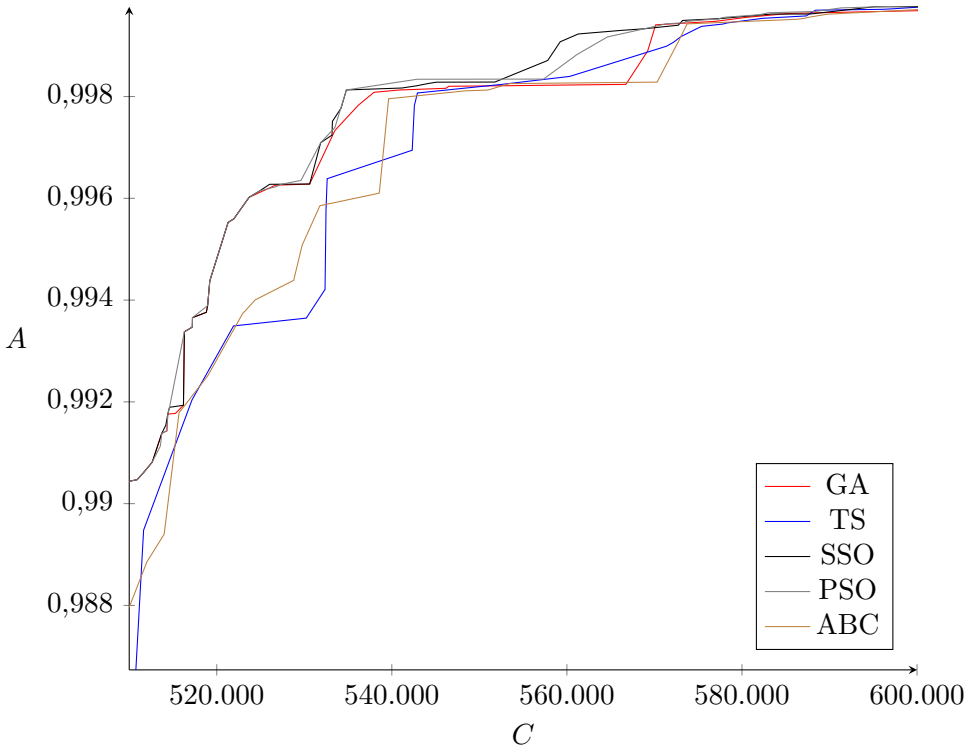

Abbildung 4.31: Ausschnitt der Pareto-Fronten der Lösungsalgorithmen für das ITRAP

4.26 aufgelistet.

Tabelle 4.26: Kennzahlenvergleich für die aggregierten Pareto-Fronten des ITRAP und des klassischen RAP für das Optimierungsproblem (3)

	NPS	S	MS
ITRAP	81	0,0300	0,77
RAP	238	0,0042	0,75

Die RAP-Front ist dabei vor allem in Anzahl und Verteilung der Lösungen performanter. Nur in der maximalen Streuung erzielt die ITRAP-Front einen leicht besseren Wert. Eine grafische Gegenüberstellung beider Fronten ist in Abbildung 4.32 zu finden. Für eine bessere Darstellung des Verhaltens in der Region hoher Kosten wird zusätzlich eine Vergrößerung dieses Bereichs präsentiert. Eine Übersicht aller Lösungen der ITRAP-Front ist in Tabelle B.2 in Anhang B gegeben.

Die schwarze Kurve der ITRAP-Front beginnt und endet dabei bei geringerer Verfügbarkeit als die graue Kurve der RAP-Lösungen. Im Mittelbereich der Fronten (bis ca. $C = 600.000$) existieren jedoch ITRAP-Lösungen, die die RAP-Lösungen dominieren. In der höheren Kostenregion kann sich die RAP-Kurve der 1 jedoch beliebig nahe annähern, während die ITRAP-Front eine deutlich geringere obere Schranke zu besitzen scheint. So erreichen die RAP-Lösungen maximal $1 - 6,2 \cdot 10^{-13}$ Verfügbarkeit, wobei im ITRAP nur 0,99996 erreicht werden. Die geringe Dichte der ITRAP-Front in dieser Region zeigt zusätzlich, dass minimale Steigerungen der Verfügbarkeit hier nur mit hohen Zusatzosten

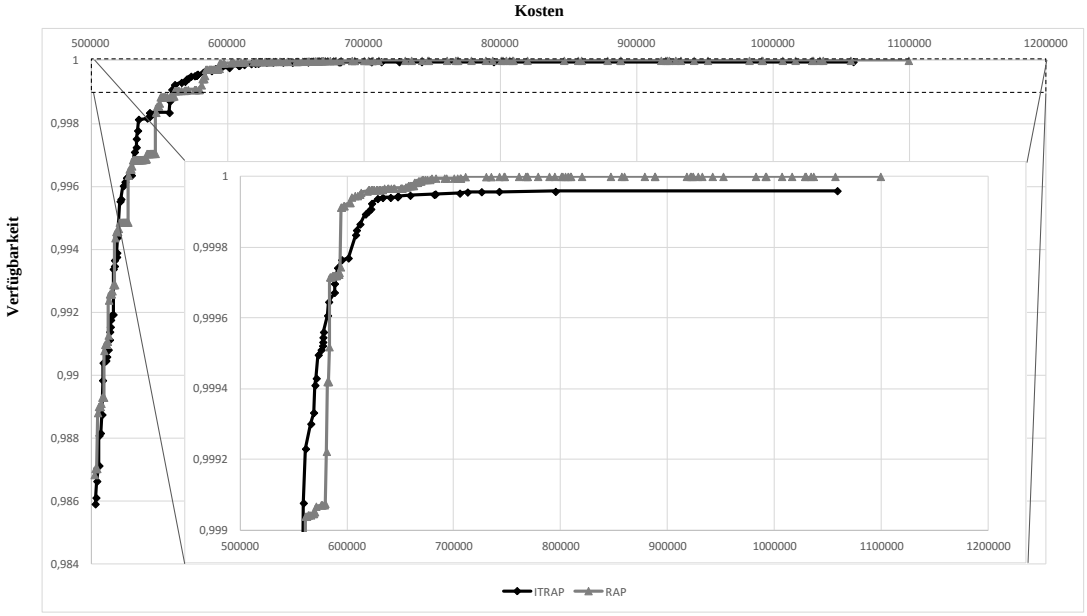

Abbildung 4.32: Darstellung der aggregierten Pareto-Fronten von ITRAP und klassischem RAP

erreicht werden können.

4.2.4 Diskussion der Ergebnisse

In diesem Abschnitt werden die aus den oben beschriebenen Experimenten erhaltenden Resultate interpretiert und diskutiert. Dazu wird zunächst die Performance der Lösungsalgorithmen für das ITRAP betrachtet, bevor ein Vergleich zwischen RAP und ITRAP durchgeführt wird.

Vergleich der Lösungsalgorithmen In allen durchgeführten Experimenten wird deutlich, dass die Performance der Lösungsalgorithmen stark vom untersuchten Bereich im Lösungsraum abhängt. So ist die Tabu-Suche vor allem im Bereich der Hochverfügbarkeit sehr effektiv, der z.B. in den dritten Szenarien der Optimierungsprobleme (1) und (2) adressiert wird, während z.B. die PSO dort im Vergleich zu den ersten Szenarien schlechter abschneidet. Dies liegt vor allem darin begründet, dass die Initialisierungsoperation die Parameter $opNum = 1$ und $subSize = 1$ benutzt. Damit ist die Wahrscheinlichkeit, Designs mit vielen Komponenten – und damit hoher Verfügbarkeit – zu erstellen, sehr gering.

In den ersten beiden Szenarien der Optimierungsprobleme (1) und (2) stehen somit in der Regel initial schon Lösungskandidaten mit guter Fitness bereit, hier bedarf es nur einer Ausbeutung dieser Region. Daher schneiden in diesen Fällen die Algorithmen mit starker Rekombinationskomponente (GA, SSO und PSO) besser ab. In den dritten Szenarien muss der initiale Bereich jedoch verlassen werden, um gute Lösungen zu finden.

Hier haben Algorithmen mit Nachbarschaftssuche und möglichst wenig Einfluss der Zufallssuche (TS und eingeschränkt ABC) eine bessere Performance. Dieser Effekt kann auch in Optimierungsproblem (3) und im Effizienzexperiment beobachtet werden. Algorithmen wie PSO und SSO dominieren dort im Bereich niedriger, die Tabu-Suche im Bereich hoher Verfügbarkeit.

Der Genetische Algorithmus schneidet – über alle Situationen betrachtet – durchschnittlich ab. Durch die Kombination von Nachbarschaftssuche und Rekombination ist er in der Lage, sowohl die Initialregion auszubeuten als auch gute Lösungen in der Hochverfügbarkeitsregion zu identifizieren. Auffallend sind für den GA die Performanceunterschiede zwischen den Optimierungsproblemen (1) und (2), wobei im letzteren Problem im Vergleich zu den anderen Algorithmen bessere Ergebnisse identifiziert werden. Dies mag in der Definition der Fitnessfunktionen begründet liegen, die im Optimierungsproblem (2) zu größeren Straftermen führt. Zwar produzieren andere Algorithmen im Einzelfall bessere Ergebnisse, dennoch kann der Einsatz eines GA grundsätzlich empfohlen werden, da dieser unter allen Bedingungen eine stabile und hohe Ergebnisqualität bietet.

Die Tabu-Suche basiert auf der Betrachtung eines einzelnen Lösungskandidaten per Iteration. Durch die fehlende Populationskomponente kann bei der Tabu-Suche die größte Schwankung in der Ergebnisqualität von allen Algorithmen beobachtet werden. Dies führt im Falle der mehrkriteriellen Optimierung sogar dazu, dass die Betrachtung nur eines Durchlaufes der Tabu-Suche nicht zu einer mit den anderen Algorithmen vergleichbaren Ergebnisqualität führt (vgl. Abschnitt 4.2.2). Findet eine Aggregation der Ergebnisse mehrerer Durchläufe statt, kann die Tabu-Suche jedoch vor allem im Hochverfügbarkeitsbereich überzeugen. Auf der anderen Seite beschränkt die Art der Tabu-Suche die Qualität vor allem in der Region niedriger Verfügbarkeit, wie es auch in den ersten Szenarien der Optimierungsprobleme (1) und (2) beobachtet werden kann. Ein Grund dafür ist in der Definition der Nachbarschaftssuche zu finden, in der Hinzufüge- und eine Löschoperationen möglich sind. Die Definition einer Austauschoperation könnte hier zu deutlich besseren Ergebnissen führen.

Für die Partikelschwarmoptimierung kann insgesamt eine durchschnittliche Performance festgestellt werden. Dabei schwankt die Güte der Ergebnisse stark in Abhängigkeit des Szenarios. So werden in Regionen niedriger Verfügbarkeit überdurchschnittliche, in der Hochverfügbarkeits-Region jedoch unterdurchschnittliche Lösungen gefunden. Dies wird durch die fehlende Nachbarschaftssuche verursacht, womit die Rekombinationskomponente nur den initialem Bereich durchforstet. Dies führt für das Optimierungsproblem (3) zu einer hohen Dominanz gegenüber den anderen Algorithmen, jedoch mit dem Nachteil, die geringste Streuung aller Algorithmen zu erzielen.

Als vielversprechende Alternative zur Partikelschwarmoptimierung wurde die Vereinfachte Schwarmoptimierung implementiert. Statt wie in der PSO einzelne Komponenten hinzuzufügen oder zu löschen, werden hier Subsysteme ausgetauscht. Als Ausgleich für diese Abstraktion wird ein Zufallsfaktor eingeführt. Dabei zeigt die SSO eine absolut vergleichbare Performance zur PSO und ist dieser sogar in den einkriteriellen Problemen überlegen. In diesen Szenarien zeigt die SSO eine sehr hohe und stabile Ergebnisqualität,

die nur in den dritten Szenarien leicht abfällt. Auch im mehrkriteriellen Optimierungsproblem (3) wird deutlich, dass die SSO den Niedrigverfügbarkeitsbereich sehr gut ausbeutet und auf der anderen Seite eine höhere Streuung erzielt als die PSO. Nichtsdestotrotz kann durch das Fehlen einer Nachbarschaftssuche der Hochverfügbarkeitsbereich kaum betrachtet werden, da auch hier die Zufallssuche in der Regel nur Individuen mit niedriger Verfügbarkeit erzeugt. An dieser Stelle muss jedoch auch erwähnt werden, dass die Einteilung einer Lösung in Subsysteme die Effektivität der SSO systematisch begrenzen kann. Dies ist besonders dann der Fall, wenn viele Komponenten in einem Subsystem verwendet werden müssen und diese aus unterschiedlichen Typen ausgewählt werden können. Unter diesen Umständen gleicht die SSO einer Zufallssuche, da sehr viele möglichen Kombinationen von Komponenten in einem Subsystem zulässig sind. Diesem Problem könnte durch Verwendung der erwähnten Vektorkodierung aus Abschnitt 3.4.2 begegnet werden, wobei nicht ganze Subsysteme, sondern Werte für die Anzahl von Komponenten manipuliert werden.

Die Künstliche Bienenkolonie kann aufgrund ihrer Implementierung mit einer parallelisierten Tabu-Suche verglichen werden. So werden zwar mehr Kandidaten pro Iteration, jedoch weniger mögliche Nachbarschaftslösungen betrachtet. Deren Anzahl wird maßgeblich von einer Selektionsmethode beeinflusst. Im einkriteriellen Fall ist dies die fitnessproportionale Selektion. Hier identifiziert die ABC Lösungen mit überdurchschnittlicher Qualität, unabhängig vom betrachteten Szenario. Diese Qualität kann jedoch nicht für das mehrkriterielle Problem beobachtet werden. In diesen Experimenten ist die Performance im Niedrigverfügbarkeitsbereich vergleichbar mit der Tabu-Suche, ohne deren Qualität im Bereich höherer Verfügbarkeit zu erreichen. Dies wird wahrscheinlich durch die unterschiedliche Selektionsmethode im mehrkriteriellen Fall bedingt, die zufällig Elemente extrahiert und auf Nicht-Dominanz prüft. Daher kann die Anzahl der betrachteten Nachbarschaftslösungen nicht so effektiv verteilt werden wie im einkriteriellen Fall.

Insgesamt kann festgestellt werden, dass keiner der betrachteten Algorithmen in den durchgeführten Experimenten komplett dominiert wird. Dennoch stechen vor allem die SSO insgesamt und die ABC für einkriterielle Probleme durch hohe Fitness und Stabilität heraus. Die Tabu-Suche scheint vor allem dann besonders geeignet zu sein, wenn die optimalen Lösungen nicht im Bereich der initialen Lösungen zu finden ist. Ein breiterer Bereich von Initiallösungen durch Erhöhung von $subSize$ und $opNum$ könnte wahrscheinlich die Performance der PSO und des GA deutlich erhöhen. Da eine hohe Entfernung von Initialbereich und Zielbereich jedoch nicht ausgeschlossen werden sollte, kann z.B. eine Kombination von SSO und TS nach den durchgeführten Experimenten empfohlen werden.

Vergleich RAP und ITRAP Ein direkter Vergleich der Ergebnisse des klassischen RAP und der des ITRAP ist nicht besonders aussagekräftig, da beide Probleme durch ihre unterschiedliche Definition auch andere Lösungsräume betrachten. Dennoch kann eine Gegenüberstellung der Ergebnisse zu Erkenntnissen führen, inwiefern sich die Definitionsunterschiede auf die vorgeschlagenen Designs auswirkt.

Zunächst fällt im Hochverfügbarkeitsbereich auf, dass die ITRAP-Designs bei ver-

gleichbaren Kosten eine niedrigere Verfügbarkeit erreichen. Dies lässt sich in sowohl in den dritten Szenarien der Optimierungsprobleme (1) und (2) als auch in der mehrkriteriellen Optimierung beobachten, wo die ITRAP-Front sich asymptotisch einer Verfügbarkeitsschranke nähert, die deutlich geringer als 100 % ist. Im Gegensatz dazu nähern sich die Elemente der Front des klassischen RAP der 100 % Verfügbarkeit beliebig an. Außerdem kann in diesem Bereich beobachtet werden, dass das RAP viel mehr Pareto-optimale Lösungen identifiziert als das ITRAP (vgl. Abbildung 4.32).

Dieses Verhalten ist eine Folge der modellierten Abhängigkeiten sowie der Administratorinteraktion. Beispielsweise kann der Ausfall eines Datenbank-Systems zu einem Ausfall der ERP-Systeme führen. Daher müssen für das Datenbank- sowie für das ERP-Subsystem mehr Komponenten betrieben werden, um die Wahrscheinlichkeit eines Totalausfalls zu verringern. So werden in den dritten Szenarien der einkriteriellen Optimierungsprobleme in diesen Subsystemen mehr Komponenten als in den RAP-Lösungen benötigt (8 ggü. 5 im ersten und 10 ggü. 4 im zweiten Optimierungsproblem). Da die Wahrscheinlichkeit eines Totalausfalls aufgrund der Abhängigkeiten jedoch nie komplett ausgeschlossen werden kann, können sich die Lösungen des ITRAP nicht beliebig der 100 % Verfügbarkeit annähern.

Die fehlerbehaftete Administratorinteraktion verstärkt diesen Effekt zusätzlich. So können durch Interaktionsfehler die Wiederherstellungszeiten der Komponenten deutlich steigen. Außerdem wird in nahezu allen ITRAP-Lösungen nur ein Administrator genutzt. Die Kapazitätsprobleme, die damit einhergehen, führen vor allem bei hoher Anzahl an Komponenten zu signifikanten Beeinträchtigungen der Verfügbarkeit. Dennoch ist eine zusätzliche Stelle äußerst kostspielig. So wird die ITRAP-Lösung mit der höchsten Verfügbarkeit in der Pareto-Front durch die Nutzung von zwei Stellen charakterisiert, hat jedoch über 33 % höhere Kosten als der nächstbeste Kandidat. Die Verfügbarkeit wird dabei nur um $2 \cdot 10^{-6}$ erhöht.

Die RAP-Lösungen nähern sich hingegen beliebig nahe der 1 an, da jede zusätzliche Komponente die Verfügbarkeit positiv beeinflusst. Im ITRAP kann eine zusätzliche Komponente in einem Subsystem mit überdurchschnittlicher Verfügbarkeit sogar die Verfügbarkeit des Gesamtdesigns senken, da Kapazitätsprobleme und Abhängigkeiten auftreten können. Damit ist die scheinbare Beschränktheit der Verfügbarkeit im ITRAP eine direkte Folge des Verzichts auf die Annahme unabhängiger Fehler. Dies führt zwar – besonders im Optimierungsproblem (3) – zu Ergebnissen mit niedrigerer Performance (z.B. Uniformität), ist aber ein Verhalten, das auch in der Realität beobachtet werden kann.

Ein weiterer Unterschied zwischen ITRAP- und RAP-Lösungen kann im Bereich zwischen 99,5 und 99,96 % Verfügbarkeit (ca. 520.000 bis 575.000 €) identifiziert werden. In diesem Bereich dominieren die ITRAP-Lösungen die des RAP und erreichen eine vergleichbare Verfügbarkeit zu geringeren Kosten, was sich für Optimierungsproblem (3) und auch in den ersten Szenarien der Optimierungsprobleme (1) und (2) beobachten lässt. Dies ist mit der Modellierung der Standby-Redundanz im ITRAP zu erklären. Eine passiv betriebene Komponente erzeugt weniger Betriebskosten, besonders bei Hardware-Komponenten. Da die Übernahmezeiten deutlich kürzer als die Wiederherstellungszeiten sind, wird der

negative Effekt der Standby-Redundanz auf die Verfügbarkeit von den geringeren Kosten deutlich kompensiert. Dazu kommt, dass passive Komponenten eine geringere Fehlerrate besitzen und von bestimmten Ausfällen aufgrund gemeinsamer Ursache verschont bleiben. Mit steigender Komponentenzahl tritt dieser positive Effekt jedoch hinter die negativen Effekte von Abhängigkeiten und Administratorinteraktion zurück.

Im Niedrigverfügbarkeitsbereich ist das RAP dominant. Hier werden in den Lösungskandidaten nur sehr wenige Komponenten verwendet, sodass die Standby-Redundanz kaum anwendbar ist. Die Kreuzabhängigkeit zwischen Datenbank- und ERP-Systemen ist jedoch auch bei geringer Komponentenanzahl vorhanden und verringert die Verfügbarkeit der ITRAP-Lösungen in diesem Bereich gegenüber dem klassischen RAP.

Abschließend kann festgestellt werden, dass sich die identifizierten Lösungen von ITRAP und klassischem RAP deutlich in Design, Verfügbarkeit und Kosten unterscheiden. Während die Modellierung von Standby-Redundanz bereits in der wissenschaftlichen Literatur untersucht wurde, ist der Effekt von Abhängigkeiten auf ein Redundanz-Allokation-Problem noch nicht untersucht worden. Daher zeigen die oben beschriebenen Experimente für den Anwendungsfall beispielhaft, dass die Beschränkung der Verfügbarkeit in der Realität in einem Redundanz-Allokation-Problem abbildbar ist sowie dass die Identifikation eines Designs mit nahezu null Nichtverfügbarkeit sich schwierig darstellt. Damit präsentiert das ITRAP einem möglichen Entscheidungsträger eine realistischere Einschätzung der vorhandenen Alternativen, da die Differenz zwischen vorhergesagter und tatsächlicher Verfügbarkeit der Designs gegenüber den bisherigen RAP-Ansätzen sinken sollten.

5 Schlusskapitel

Nachdem in den vorangegangenen Kapiteln die Entwicklung des Artefakts sowie dessen Evaluierung beschrieben wurden, werden in diesem Kapitel die wichtigsten Erkenntnisse der Arbeit wiedergegeben und diskutiert.

Hierfür folgt zunächst eine umfassende Zusammenfassung der Arbeit, bevor offene Fragen und Weiterentwicklungspotenzial dieser Arbeit in einem Ausblick diskutiert werden. Die Arbeit schließt mit einem kritischen Fazit der Arbeit aus wissenschaftlicher und praktischer Sicht.

5.1 Zusammenfassung

Die digitale Revolution hat das Leben vieler Menschen auf der Welt durchdrungen. Dies gilt auch für zahllose Unternehmen, in denen die IT-Unterstützung ihrer Geschäftsprozesse nicht mehr wegzudenken ist. Der industrielle Trend zur Spezialisierung und der Siegeszug von Unternehmen, deren komplette Geschäftsmodelle auf dem Einsatz von Informationstechnologie beruhen, führen dazu, dass immer mehr Unternehmen auf IT-Services aufbauen, die von (oft externen) Dienstleistern erbracht werden.

Da das Outsourcing der IT jedoch auch einen Kontrollverlust mit sich bringt, werden Vereinbarungen zwischen IT-Service-Kunden und -Anbietern getroffen, die die zu erwartende Servicequalität beschreiben: die so genannten Service-Level-Agreements (SLA). Als eines der wichtigsten Qualitätsmerkmale einer IT-Dienstleistung wird dort die Verfügbarkeit beschrieben. Diese ist die Wahrscheinlichkeit einer Dienstleistung, zu einem bestimmten Zeitpunkt die vom Kunden gewünschte Funktion auszuführen. In der Regel wird sie als Verhältnis der verfügbaren Zeit zur Gesamtlaufzeit eines Services angegeben. Aufgrund der in SLA definierten Strafzahlungen und der Angst vor einem Reputationsverlust haben IT-Service-Provider ein Interesse daran, Verletzungen der Verfügbarkeit und damit der SLA zu vermeiden.

Auf der anderen Seite müssen beim Design fehlertoleranter Systeme zur Erhöhung der Verfügbarkeit auch erhöhte Kosten, u.a. für Anschaffung oder Betrieb dieser Systeme, beachtet werden. Damit stehen Entscheidungsträger im IT-Service-Design vor der Aufgabe, die Kosten und die Verfügbarkeit eines Designs als widersprüchliche Optimierungsziele abzuwägen. Für ein effektives und effizientes IT-Service-Management sollte diese Abwägung durch adäquate Werkzeuge unterstützt werden.

Im klassischen Zuverlässigkeitsmanagement können Redundanz-Allokation-Probleme (RAP) definiert werden, um Verfügbarkeit und Kosten eines Systemdesigns computer-

gestützt zu optimieren. Dabei werden die relevanten Funktionsbausteine eines Systems, die so genannten Subsysteme, modelliert. In jedem Subsystem können eine beliebige Anzahl funktional äquivalenter Komponenten genutzt werden. Dabei erhöhen sich Kosten und Verfügbarkeit eines Subsystems mit einer höheren Anzahl an Komponenten. Mithilfe der Annahme unabhängiger Komponentenausfälle können Kosten und Verfügbarkeit eines Systemdesigns kombinatorisch berechnet werden. Auf dieser Basis können Optimierungsverfahren angewandt werden.

Die Annahme unabhängiger Komponentenausfälle ist jedoch in IT-Systemen unzutreffend. Besonders zwischen Software-Systemen können Abhängigkeiten bestehen, die die tatsächliche Verfügbarkeit eines IT-Service-Designs beschränken. Ein Beispiel dafür sind Ausfälle aufgrund gemeinsamer Ursachen, die auf systematischen Fehlern beruhen und mehrere redundante Systeme gleichzeitig betreffen können. Weiterhin kann fehlerbehaftete Administratorinteraktion, die für einen Großteil der Ausfälle von IT-Systemen verantwortlich sind, nicht abgebildet werden.

Ziel dieser Arbeit ist es daher, durch bessere Unterstützung von Entscheidungsträgern in der IT-Service-Design-Phase die Wirtschaftlichkeit von IT-Service-Providern zu verbessern. Dieses Ziel wird durch Definition eines neuartigen RAP ohne die Annahme unabhängiger Komponentenausfälle adressiert Dafür wurden Anforderungen an ein RAP für IT-Service-Design identifiziert. Auf dieser Basis wurde das ITRAP definiert. Da eine kombinatorische Auswertung eines möglichen Designs im ITRAP nicht möglich ist, wurde der Zustandsraum eines ITRAP in Abhängigkeit vom zu untersuchenden Design modelliert. Um das stabile Verhalten dieser Zustandsräume zu analysieren, wurde eine Monte-Carlo-Simulation auf Basis von automatisch generierten Petri-Netzen entwickelt. So können Kosten und Verfügbarkeit eines Designs auch ohne die Annahme unabhängiger Komponentenausfälle geschätzt werden.

Für die effiziente Optimierung eines ITRAP konnte auf für RAP genutzte Meta-Heuristiken zurückgegriffen werden. Durch Anpassung der Algorithmen auf das ITRAP und auf die Auswertung der Lösungskandidaten durch Simulation kann für die Lösung des ITRAP ein Genetischer Algorithmus, eine Tabu-Suche, eine Partikelschwarmoptimierung, eine Vereinfachte Schwarmoptimierung sowie eine Künstliche Bienenkolonie genutzt werden.

Auf Basis dieses Konzepts wurde ein Prototyp in dem Java-basierenden Simulationsframework AnyLogic entwickelt. Für diesen Prototypen konnte die Erfüllung der gestellten funktionalen und nicht-funktionalen Anforderungen nachgewiesen werden. So konnte die Anwendbarkeit des ITRAP in einem Anwendungsszenario eines Application Service Providers untersucht werden, für welches das Service-Design in verschiedenen Szenarien optimiert wurde. Dabei wurde die Maximierung der Verfügbarkeit unter einer Kostenbedingung, die Minimierung der Kosten unter Einhaltung einer Verfügbarkeitsschranke sowie die mehrkriterielle, Pareto-basierte Optimierung von Kosten und Verfügbarkeit adressiert.

Die durchgeführten Experimente zeigen, dass die Definition des ITRAP auf ein reales IT-Service-Design anwendbar ist. Weiterhin sind alle genutzten Algorithmen in der Lage, auf effiziente Weise (sub)optimale Lösungen zu identifizieren. Im Vergleich der Algorith-

men konnte außerdem die Robustheit der Verfahren auf die Eingangsparameter und Operationen getestet werden. Dabei überzeugen vor allem die Vereinfachte Schwarmoptimierung bei der Ausbeutung der Regionen im Suchraum mit niedrigen Komponentenzahlen sowie die Tabu-Suche mit der Fähigkeit, einen großen Bereich des Suchraumes abzudecken. Ist die Künstliche Bienenkolonie für die einkriterielle Optimierung noch eine starke Alternative, ist sie für die mehrkriterielle Optimierung weniger geeignet. Der definierte Genetische Algorithmus zeigt eine gute Ergebnisqualität unter verschiedenen Bedingungen, was ein Grund für die Popularität dieses Ansatzes in der RAP-Literatur darstellt.

Die untersuchten Szenarien wurden außerdem auch mit einem klassischen RAP-Ansatz untersucht, um Rückschlüsse auf die Effekte der neuartigen Definition des ITRAP zu ziehen. Es konnte festgestellt werden, dass sich die vorgeschlagenen Designs in allen Experimenten signifikant unterscheiden. Dabei limitieren vor allem die Abhängigkeiten zwischen den Komponenten und die fehlerbehaftete Administratorinteraktion die zu erreichende Verfügbarkeit. Daher müssen im Vergleich zu den Lösungen des klassischen RAP viel höhere Kosten getragen werden, um Hochverfügbarkeit zu erreichen.

Die Ergebnisse der Experimente zeigen damit, dass das ITRAP zu unterschiedlichen Designvorschlägen als ein klassisches RAP führt. Dabei adressiert es einige der Hauptgründe für die Differenz zwischen geplanter und tatsächlicher Verfügbarkeit. Daher stellt das ITRAP ein Konzept dar, dass die Entscheidungsunterstützung für IT-Service-Designer verbessern und somit die Wirtschaftlichkeit von IT-Service-Providern erhöhen kann.

5.2 Ausblick

Anknüpfungspunkte in dieser Arbeit für die zukünftige Forschung können in den Bereichen der Verbesserung des Konzepts, der Verringerung des Aufwandes, der Verbesserung der Lösungsmethoden sowie der Fortsetzung der Evaluierung gefunden werden.

Ein fortgeschritteneres Konzept kann die Vorhersage der Kosten und Verfügbarkeit weiter verbessern und damit die Anwendbarkeit eines RAP für IT-Service-Design erhöhen. Dies betrifft einerseits die Verfügbarkeitsvorhersage. Eine durchgehende Administratorstelle, wie sie im ITRAP modelliert werden kann, entspricht u.U. nicht der Realität von IT-Service-Providern. Hier könnte die Modellierung von Zeitplänen für die Administratoren oder einer bedarfsgerechten Personalpolitik eine realistischere Abbildung sein. Auch kostspielige Maßnahmen gegen das Auftreten von Interaktionsfehlern wie z.B. Validationsumgebungen [NOB+04] oder verbesserte Diagnose-Tools [OGP03] könnten in ein RAP für IT-Service-Design integriert werden. Weiterhin wurde die Performanceerwartung an die IT-Dienstleistung als feste Funktion der Zeit angenommen. Eine Integration komplexerer Methoden zur Performanceanalyse wäre an dieser Stelle vorteilhaft. Dabei muss jedoch das Problem beachtet werden, dass Ausfälle gegenüber den normalen Performanceschwankungen aufgrund unterschiedlicher Last sehr seltene Ereignisse sind. Dies könnte im Besonderen den Aufwand des Simulationsansatzes deutlich erhöhen. Auch können weitere Kostenarten wie z.B. weitere Lohnkosten oder Konstruktionskosten in das Modell integriert werden, um die Gesamtbetriebskosten (TCO von engl. *total cost of ownership*)

eines IT-Service besser zu erfassen.

Auf der anderen Seite kann die Entscheidungsunterstützung verbessert werden. Die Verwendung von Mittelwerten nach einer festen Anzahl an Simulationsläufen kann zu Fehlschlüssen führen. Hier besteht die Möglichkeit, die Grenzen von Konfidenzintervallen als Entscheidungsgrundlage zu verwenden. Statt einer festen Anzahl an Replikationen könnte eine Zielgenauigkeit für dieses Intervall als Abbruchbedingung der Simulation definiert werden. Dieses Vorgehen hätte den Vorteil, dass eine besser fundierte Entscheidung über signifikante Dominanz zwischen unterschiedlichen Designs getroffen werden kann. Dennoch belastet es den Entscheidungsträger mit mehr Daten, die die Übersichtlichkeit der Ergebnisse einschränken.

Einer möglichen Überforderung des Entscheidungsträgers in der mehrkriteriellen Optimierung kann mit der intelligenten Reduktion der Ergebnisse (engl. *pruning*) entgegen gewirkt werden. Dabei werden ähnliche Lösungen zu repräsentativen Kandidaten zusammengefasst, indem z.B. ein Abstandsmaß oder Präferenzen des Entscheidungsträgers berücksichtigt werden (z.B. in [KKCB08]).

Eine weitere Möglichkeit, die Anwendbarkeit eines RAP für IT-Service-Design zu erhöhen, stellt die Reduktion des Aufwandes dar. Durch die komplexe Definition sowie die simulationsbasierte Auswertung der Lösungen sind Modellierung und Lösung eines ITRAP sehr aufwändig. Die Modellierung erfordert dabei oft kostspieliges Expertenwissen. Daher sollten alle bereits dokumentierten Informationen zur Modellierung und Parametrisierung eines RAP genutzt werden, z.B. aus Konfigurations-Management-Datenbanken (CMDB) oder Infrastruktur-Monitoring-Tools (vgl. z.B. [MM11]).

Der Zeitaufwand für die Evaluation eines Designs per Simulation ist gegenüber einer kombinatorischen Auswertung massiv erhöht. Je komplexer dabei die auszuwertenden Lösungskandidaten sind, desto größer ist diese Steigerung. Dies liegt darin begründet, dass nicht nur mehr Ereignisse verarbeitet werden müssen, sondern auch u.U. eine höhere Anzahl an Simulationsläufen notwendig ist, um zu statistisch signifikanten Aussagen zu kommen. Letzteres Problem ließe sich jedoch durch massive Parallelisierung adressieren, da die einzelnen Replikationen voneinander unabhängig sind [SKK08].

Zur besseren Vergleichbarkeit der Experimentergebnisse wurden alle Verfahren in derselben Umgebung durchgeführt und ein Repository der bereits simulierten Lösungen vorgehalten. Diese Informationen könnten durch Methoden der intelligenten Datenanalyse verwendet werden, um die Verfügbarkeit und Kosten eines Designs auch ohne Simulation vorherzusagen. In [CS96b] wird die Nutzung eines Neuronalen Netzes vorgeschlagen, welches mit den Simulationsergebnissen trainiert wird. Sobald der Unterschied zwischen simulierten und vorhergesagten Ergebnissen unter eine definierte Schranke fällt, kann auf die Simulation verzichtet werden. Damit könnte der Zeitaufwand für die Auswertung von Lösungen deutlich verringert werden, da die Ergebnisse des Trainings für weitere Optimierungen gespeichert werden können. Dabei muss jedoch beachtet werden, dass die gespeicherten Vorhersagen bei veränderter RAP-Definition unbrauchbar werden können.

Als weitere Möglichkeit zur Reduktion des Aufwandes kann die Effizienz der Lösungsalgorithmen verbessert werden. Zum Beispiel wurde für die Berechnung der Fitness ein

Grenzwert genutzt, der Lösungen mit Verletzungen über dem Grenzwert stärker bestraft als Lösungen mit Verletzungen innerhalb dieses Wertes. In [KKSN06] wird vorgeschlagen, diesen Grenzwert nicht fest im Szenario zu definieren, sondern ihn in jeder Iteration eines Lösungsalgorithmus auf Basis des Verhältnisses von zulässigen zu unzulässigen Lösungen anzupassen. Damit könnte einerseits die Effizienz der Algorithmen erhöht werden und andererseits die Komplexität einer ITRAP-Definition leicht vermindern.

Weiterhin legen die Experimente einige Verbesserungsmöglichkeiten in den definierten Operationen nahe. Zunächst kann die Steuerung der Zufallserstellung von Lösungen sich stark auf die Qualität der Algorithmen auswirken. Hier könnten Möglichkeiten eruiert werden, auf Basis der Performance eines Algorithmus die Steuerparameter anzupassen, wie z.B. der NFT-Wert in [KKSC03]. Die Tabu-Suche zeigt vor allem im Bereich der niedrigen Verfügbarkeit ein unterdurchschnittliches Verhalten. Wie bereits oben erwähnt, sollte neben der Hinzufüge- und Löschoperation auch eine Tauschoperation zur Durchforstung der Nachbarschaft einer Lösung genutzt werden. Auch können andere Meta-Heuristiken das ITRAP effektiv und effizient optimieren. Neben den genutzten Algorithmen wurden in der wissenschaftlichen Literatur auch z.B. die Harmonie-Suche, die Ameisenkolonieoptimierung, das Simulierte Ausglühen, ein Honigbienen-Paarungs-Algorithmus, Immunbasierte Algorithmen sowie die Kuckuck-Suche erfolgreich für RAP eingesetzt.

Für die Abbildung von Ungenauigkeit wurde u.a. die Fitness einer Lösung als Intervall definiert. Im Rahmen der Arbeit werden für die jeweiligen Intervallgrenzen unabhängige Optimierungsläufe durchgeführt. Die eingesetzten Verfahren könnten jedoch auf die Behandlung von Fitnessintervallen angepasst werden, um die derzeit manuelle Analyse der pessimistischen und optimistischen Ergebnismengen zu automatisieren.

Zusätzlich zur Verfeinerung des Konzepts für die verbesserte Anwendbarkeit sollte das Konzept weiter evaluiert werden. Dazu sollten ITRAP für weitere Anwendungsfälle aufgestellt und gelöst werden. Damit könnte die Eignung der Lösungsmethoden in einem breiteren Kontext untersucht werden, zum Beispiel für Probleme in denen mehr Komponententypen in Subsystemen zur Verfügung stehen. Außerdem können in zusätzlichen Anwendungsfällen die gewonnenen Erkenntnisse und die genutzten Parametrisierungen weiterverwendet bzw. -entwickelt werden sowie die nicht in den Experimenten adressierten ITRAP-Elemente wie Ungenauigkeit oder komplexe Designs untersucht werden.

Auch wenn Probleme bei der langfristigen Datensammlung für die Analyse der Verfügbarkeit bekannt sind, könnte ein Vergleich zwischen vorhergesagter und realer Verfügbarkeit eines Designs wichtige Aufschlüsse darüber geben, inwieweit die Differenz zwischen Theorie und Praxis im Verfügbarkeitsmanagement besteht. Auf Basis eines solchen Vergleiches könnte auch eine Abwägung von Modellieraufwand und Ergebnisgenauigkeit erfolgen.

5.3 Fazit

Das Ziel dieser Arbeit ist die Verbesserung der Wirtschaftlichkeit von IT-Service-Providern durch optimiertes Service-Design im Kontext der Verfügbarkeit und Kosten. Dafür wurde eine Entscheidungsunterstützung konzipiert, um Kosten und Verfügbarkeit eines Designs

von fehlertoleranten IT-Service-Systemen abzuwägen. Basis dafür sind Vorarbeiten im Bereich des Redundanz-Allokation-Problems (RAP), welche durch die Annahme unabhängiger Komponentenausfälle nicht auf IT-Service-Design anwendbar sind. Die adressierte Forschungsfrage war dabei, ob und wie der RAP-Ansatz auf IT-Service-Design übertragbar wäre.

Um diese Frage zu beantworten, musste die aktuelle Forschung im RAP-Bereich, die sich z.B. mit der Entwicklung von effizienten und effektiven Lösungsalgorithmen oder der Einführung von Ungewissheit beschäftigt, mit den wissenschaftlichen Bemühungen verbunden werden, die Verfügbarkeit von IT-Services vorherzusagen. Beide Forschungsbereiche konnten mit dem entwickelten Artefakt, dem ITRAP, vereinigt werden. Die Experimente im Anwendungsfall zeigen, dass der neue Ansatz zu signifikant unterschiedlichen Designvorschlägen kommt als ein klassischer RAP-Ansatz. Durch die realistischere Modellierung des Ausfallverhaltens versprechen die Ergebnisse des ITRAP dabei eine höhere Validität.

Damit konnte gezeigt werden, dass die Übertragung des RAP-Ansatzes auf das IT-Service-Design möglich ist. Mit der Entwicklung einer zustandsraumbasierten Evaluationsmethode in Form einer Petri-Netz-Monte-Carlo-Simulation konnte außerdem gezeigt werden, dass die Modellierung generischer Komponentenabhängigkeiten im RAP-Kontext möglich ist. Durch die verbesserten Designvorschläge bietet das ITRAP eine fundiertere Entscheidungsunterstützung für IT-Service-Designer. Somit können vermehrte Ausfälle eines IT-Service oder zu hohe Kosten durch fehlende oder unzureichende Planung des Fehlertoleranzdesigns noch vor der Inbetriebnahme verhindert werden. Damit konnte das Ziel der Arbeit erreicht werden.

Betrachtet man die Arbeit in einem breiteren Forschungskontext, konnte nachgewiesen werden, dass ein RAP auch ohne kombinatorischen Berechnung effizient, d.h. in nichtexponentiellem Zeitaufwand, gelöst werden kann. Dennoch liegt der Zeitaufwand einige Größenordnungen über dem der kombinatorischen Evaluation. Die dafür benötigte Annahme unabhängiger Komponentenausfälle limitiert die Anwendbarkeit des RAP-Ansatzes dabei jedoch nicht nur im IT-Service-Design. Schließlich finden sich IT-Komponenten heute in sehr vielen technischen Systemen, deren Analyse durch eine zustandsraumbasierte Evaluation in RAP-Ansätzen effektiver gestaltet werden kann. Weiterhin wurden in dieser Arbeit einige Konzepte genutzt, die bisher in der RAP-Forschung nicht berücksichtigt worden sind. So wird erstmals eine mehrkriterielle Bienenkolonieoptimierung auf ein RAP angewandt, wenn auch mit unterdurchschnittlichem Erfolg. Die Nutzung eines Graphen aus Subsystemen erweitert die Möglichkeit, beliebige RAP mit komplexem Design zu untersuchen. In verwandten Arbeiten wurden bisher nur Einzelfälle untersucht, aber keine generalisierbare Möglichkeit geboten, komplexe Designs zu definieren. Eine weitere Neuerung ist die Nutzung einer Zufälligen Intervallgröße, die vor allem beim verwendeten Simulationsansatz Ungenauigkeit in den Parametern effizient abbilden kann.

Für die Forschung im Bereich des Verfügbarkeitsmanagements für IT-Services werden existierende Ansätze oft als zu komplex beschrieben und deren Anwendbarkeit aufgrund fehlender Validität in Frage gestellt. Die Einbettung einer Vorhersagemethode für IT-

Service-Design in einen Optimierungsrahmen sollte die Relevanz dieser Methoden in der Praxis und Forschung deutlich erhöhen. Die Modellierung der verfügbarkeitsrelevanten Charakteristika eines Systems ist ein komplexer Vorgang, der oft durch Experten unterstützt werden muss. Die erhaltenden Daten durch solche Modelle für die Verfügbarkeit eines Designs sind dabei schwierig einzuordnen. Das ITRAP zeigt hier einen Weg auf, wie diese Modelle mit nur geringem Aufwand so erweitert werden können, dass eine Designoptimierung möglich ist. Damit wird der Aufwand zur Modellierung der IT-Service-Verfügbarkeit besser gerechtfertigt.

Neben den oben diskutierten Vorteilen des Ansatzes gegenüber einem klassischen RAP sollen im Folgenden die Nachteile des ITRAP kritisch betrachtet werden. Wie bereits in der Evaluierung deutlich geworden und im Ausblick erwähnt, ist der Modellier- und Zeitaufwand des ITRAP deutlich erhöht gegenüber einem klassischen Ansatz. Die im Ausblick erwähnten Maßnahmen können diesen Nachteil nur verringern, aber nicht eliminieren. Dies ist vor allem im Hinblick auf die Parametrisierung der Komponenten kritisch. Um deren Zustandsraum zu modellieren, müssen Informationen über das Fehlerverhalten dieser Komponenten vorhanden sein. Diese sind jedoch, speziell bei neu entwickelten oder geänderten Softwarekomponenten, oft nur begrenzt vorhanden.

Dazu gehören auch die Abhängigkeiten zwischen den Komponenten. Sind diese unbekannt, müssen sie zuerst in Test-Umgebungen identifiziert werden, was wiederum den Aufwand beträchtlich erhöht. Sollten jedoch existierende Abhängigkeiten nicht modelliert werden, wird die Qualität der ITRAP-Vorschläge reduziert und damit die Entscheidungsfindung erschwert. Dass Herstellerangaben bezüglich des Ausfallverhaltens oft überoptimistisch sind und Verfügbarkeitsstatistiken erst über einen Zeitraum gesammelt werden können, der länger dauern kann als die Entwicklungszyklen der Hersteller, erschwert die Parametrisierung zusätzlich.

An dieser Stelle muss also im konkreten Einzelfall entschieden werden, ob die Anwendung des ITRAP sinnvoll ist. Einerseits ist zu klären, ob die Charakteristika des zu optimierenden Systems (z.B. folgenschwere Abhängigkeiten oder häufige Interaktionsfehler) einen so komplexe RAP-Definition wie die des ITRAP verlangen. Auf der anderen Seite stellt sich die Frage, ob und inwieweit ein komplexeres Modell auch parametrisiert werden kann. Nur wenn beide Fragen zu bejahen sind, kann die Anwendung des ITRAP zu Designvorschlägen kommen, die als fundierte Entscheidungsunterstützung dienen können.

Weiterhin müssen durch die Nutzung von Meta-Heuristiken zur Lösung des ITRAP geeignete Werte für die Steuerparameter dieser Verfahren identifiziert werden. Die durchgeführten Experimente zeigen dabei, dass vor allem die Steuerung der Zufallssuche für die Performance eines Lösungsalgorithmus entscheidend sein kann. Daher müssen vor der eigentlichen Optimierung zunächst Tuningexperimente durchgeführt werden, was den Aufwand zum Einsatz des ITRAP weiter erhöht.

Trotz der erwähnten Nachteile im Bereich des Aufwands zur Definition und Lösung eines ITRAP verspricht die Anwendung des ITRAP ein besser kalkulierbares Verhalten des Fehlertoleranzdesigns von IT-Services. Zusätzlich ermöglicht das ITRAP eine bessere Betrachtung der TCO eines IT-Services, da z.B. auch Betriebskosten erfasst sind. Damit

können einerseits Kundenunzufriedenheit und Strafzahlungen, andererseits unerwartet hohe Kosten besonders in der ersten Betriebszeit des Services verringert bzw. vermieden werden. Diese Tatsache und die beschriebene Bedeutung der Arbeit für die Forschung im RAP-Bereich sowie die Vorhersage der Verfügbarkeit von IT-Services zeichnen daher ein positives Gesamtbild der Arbeit.

A Parametrisierung des Anwendungsfalls

Tabelle A.1: Parametrisierung der Komponente im ERP-Subsystem

Parameter	**Wert**	
Name	SAP ERP 6.0	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (0,5), Normal (1,0)	
Energiekosten	Normal & Teilausfall & Ausfall	0€ per h
Anschaffungskosten	2.500€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(8.760h)
	TTR	~exp(1,73h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	0€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~exp(4.380h)
	TTR	~exp(0,865h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	0€
Standby-Redundanz	Hot-Standby	
	Aktivierungszeit	~tria(0,083;0,292h; 0,5h)
Abhängigkeiten	Common-Cause-Fehler: Ausfall $\rightarrow_{0,3}$ Ausfall SAP ERP 6.0	

Tabelle A.2: Parametrisierung der Komponente 1 im ERP-Server-Subsystem

Parameter	**Wert**	
Name	HP ProLiant BL620c G7	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (0,9), Normal (1,0)	
Energiekosten	Normal & Teilausfall	0,0573€ per h
	Ausfall	0,0041€ per h
Anschaffungskosten	11.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(1.011,5636h)
	TTR	~exp(2,0222h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	1.000€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~tria(2554h; 28508h; 54463h)
	TTR	~tria(0,77h;2.31h;3,85h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	400€
Standby-Redundanz	Warm-Standby	
	Aktivierungszeit	~tria(0,0167h;0,05h; 0,0833h)

Tabelle A.3: Parametrisierung der Komponente 2 im ERP-Server-Subsystem

Parameter	**Wert**	
Name	HP ProLiant DL980 G7	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (1,2), Normal (2,0)	
Energiekosten	Normal & Teilausfall	0,082€ per h
	Ausfall	0,0205€ per h
Anschaffungskosten	50.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(2.2023,1h)
	TTR	~exp(2,0222h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	2.000€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~tria(2554h; 28508h; 54463h)
	TTR	~tria(0,77h;2.31h;3,85h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	600€
Standby-Redundanz	Warm-Standby	
	Aktivierungszeit	~tria(0,0167h; 0,05h; 0,0833h)

Tabelle A.4: Parametrisierung der Komponente im Datenbank-Subsystem

Parameter	**Wert**	
Name	IBM DB2	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (0,5), Normal (1,0)	
Energiekosten	Alle Zustände	0€ per h
Anschaffungskosten	652€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(8.760h)
	TTR	~exp(1,73h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	0€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~exp(4.380h)
	TTR	~exp(0,865h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	0€
Standby-Redundanz	Hot-Standby	
	Aktivierungszeit	~tria(0,0167h;0,05h; 0,0833h)
Abhängigkeiten	Ausfall $\rightarrow_{0,5}$ Ausfall SAP ERP 6.0	

Tabelle A.5: Parametrisierung der Komponente 1 im DB-Server-Subsystem

Parameter	**Wert**	
Name	HP ProLiant BL460c G7	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (0,9), Normal (1,0)	
Energiekosten	Normal & Teilausfall	0,036€ per h
	Ausfall	0,0029€ per h
Anschaffungskosten	5.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(1.011,56h)
	TTR	~exp(2,0222h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	500€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~tria(2554h; 28508h; 54463h)
	TTR	~tria(0,77h;2.31h;3,85h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	350€
Standby-Redundanz	Warm-Standby	
	Aktivierungszeit	~tria(0,0167h; 0,05h; 0,0833h)

Tabelle A.6: Parametrisierung der Komponente 2 im DB-Server-Subsystem

Parameter	**Wert**	
Name	HP ProLiant DL360 G6	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (1,2), Normal (1,5)	
Energiekosten	Normal & Teilausfall	0,082€ per h
	Ausfall	0,0205€ per h
Anschaffungskosten	10.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(2.2023,1h)
	TTR	~exp(2,0222h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	500€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~tria(2554h; 28508h; 54463h)
	TTR	~tria(0,77h;2.31h;3,85h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	200€
Standby-Redundanz	Warm-Standby	
	Aktivierungszeit	~tria(0,0167h; 0,05h; 0,0833h)

Tabelle A.7: Parametrisierung der Komponente im Proxy-Subsystem

Parameter	**Wert**	
Name	SAP Router	
Zustände (Perf.-Niveau)	Ausfall (0,0), Normal (1,0)	
Energiekosten	Normal	0,041€ per h
	Ausfall	0,0131€ per h
Anschaffungskosten	9.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(8.760h)
	TTR	~exp(1,73h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	900€
Standby-Redundanz	Warm-Standby	
	Aktivierungszeit	~tria(0,0167h; 0,05h; 0,0833h)
Abhängigkeiten	Common-Cause-Fehler: Ausfall $\rightarrow_{0.1}$ Ausfall SAP Router	

Tabelle A.8: Parametrisierung der Komponente im Storage-Management-Subsystem

Parameter	**Wert**	
Name	HP Storage Essentials	
Zustände (Perf.-Niveau)	Ausfall (0,0), Normal (1,0)	
Energiekosten	Normal & Ausfall	0€ per h
Anschaffungskosten	2.500€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(1.011,56h)
	TTR	~exp(2,02h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	250€
Standby-Redundanz	Hot-Standby	
	Aktivierungszeit	~tria(0,0167h; 0,05h; 0,0833h)
Abhängigkeiten	Common-Cause-Fehler: Ausfall $\rightarrow_{0.2}$ Ausfall HP SE	

Tabelle A.9: Parametrisierung der Komponente im Load-Balancer-Subsystem

Parameter	**Wert**	
Name	SAP ACC	
Zustände (Perf.-Niveau)	Ausfall (0,0), Normal (1,0)	
Energiekosten	Normal & Ausfall	0€ per h
Anschaffungskosten	2.500€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(1.011,56h)
	TTR	~exp(2,02h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	250€
Standby-Redundanz	Hot-Standby	
	Aktivierungszeit	~tria(0,0167h; 0,05h; 0,0833h)
Abhängigkeiten	Common-Cause-Fehler: Ausfall $\rightarrow_{0.2}$ Ausfall SAP ACC	

Tabelle A.10: Parametrisierung der Komponente im Storage-Subsystem

Parameter	**Wert**	
Name	HP Storage Works EVA 6400	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (0,5), Beeinträchtigt (0,9), Normal (1,0)	
Energiekosten	Ausfall	0€ per h
	Sonstige	0,688€ per h
Anschaffungskosten	25.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(6.438,76h)
	TTR	~exp(5,64h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	1000€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~exp(2.553,59h)
	TTR	~exp(3,85h)
	Menschl. Interaktion	wahr
	Wiederherstellungskosten	250€
	„Beeinträchtigung“	
	Start- & Endzustand	Normal → Beeinträchtigt
	TTF	~exp(3.219,38h)
	TTR	~exp(2,82h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	0€
Standby-Redundanz	Warm-Standby	
	Aktivierungszeit	~tria(0,083h; 0,292h; 0,5h)
Abhängigkeiten	Beeinträchtigung $\rightarrow_{0.1}$ Beeinträchtigung HP SW EVA 6400	

Tabelle A.11: Parametrisierung der Komponente im LAN-Subsystem

Parameter	**Wert**	
Name	Ethernet-Kabelbaum	
Zustände (Perf.-Niveau)	Ausfall (0,0), Teilausfall (0,5), Normal (1,0)	
Energiekosten	Normal & Teilausfall & Ausfall	0€ per h
Anschaffungskosten	6.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(3.399,96h)
	TTR	~exp(12,33h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	0€
	„Teilausfall“	
	Start- & Endzustand	Normal → Teilausfall
	TTF	~exp(8322h)
	TTR	~exp(4,8h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	2000€

Tabelle A.12: Parametrisierung der Komponente im WAN-Subsystem

Parameter	**Wert**	
Name	Internetanschluss	
Zustände (Perf.-Niveau)	Ausfall (0,0), Normal (1,0)	
Energiekosten	Normal & Ausfall	0€ per h
Anschaffungskosten	10.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(8.759,75h)
	TTR	~exp(0,25h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	0€
Abhängigkeiten	Common-Cause-Fehler: Ausfall $\rightarrow_{0.8}$ Ausfall WAN	

Tabelle A.13: Parametrisierung der Komponente 1 im Strom-Subsystem

Parameter	**Wert**	
Name	Stromanschluss	
Zustände (Perf.-Niveau)	Ausfall (0,0), Normal (1,0)	
Energiekosten	Normal & Ausfall	0€ per h
Anschaffungskosten	10.000€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(8.759,75h)
	TTR	~exp(0,25h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	0€
Abhängigkeiten	Common-Cause-Fehler: Ausfall $\rightarrow_{0.8}$ Ausfall Strom	

Tabelle A.14: Parametrisierung der Komponente 2 im Strom-Subsystem

Parameter	**Wert**	
Name	UPS	
Zustände (Perf.-Niveau)	Ausfall (0,0), Normal (1,0)	
Energiekosten	Normal	3€ per h
	Ausfall	0€ per h
Anschaffungskosten	2.500€	
Transitionen	*„Ausfall“*	
	Start- & Endzustand	Normal → Ausfall
	TTF	~exp(8.759,75h)
	TTR	~exp(0,25h)
	Menschl. Interaktion	falsch
	Wiederherstellungskosten	250€
Standby-Redundanz	Warm-Standby	
	Aktivierungszeit	~tria(0s; 0,05s; 1s)

B Ergebnisse der Experimente

Tabelle B.1: Ergebnisse des Effizienzexperiments

Alg.	Parameter	F_1	MID	MS	Zeit in s
DF		1,00	0,000	1,00	82,52
ZS	$n = 1000$	$[0,32; 0,35]$	$[0,025; 0,029]$	$[0,74; 0,79]$	$[1,08; 1,20]$
	$n = 3000$	$[0,40; 0,43]$	$[0,021; 0,024]$	$[0,77; 0,82]$	$[4,25; 4,33]$
	$n = 5000$	$[0,43; 0,45]$	$[0,019; 0,022]$	$[0,77; 0,82]$	$[9,03; 9,15]$
	$n = 7000$	$[0,44; 0,48]$	$[0,018; 0,021]$	$[0,77; 0,82]$	$[16,72; 17,03]$
	$n = 9000$	$[0,47; 0,50]$	$[0,017; 0,020]$	$[0,77; 0,82]$	$[28,13; 28,89]$
	$n = 11000$	$[0,48; 0,52]$	$[0,017; 0,019]$	$[0,76; 0,81]$	$[42,28; 43,31]$
	$n = 13000$	$[0,49; 0,53]$	$[0,017; 0,019]$	$[0,76; 0,80]$	$[58,86; 59,41]$
	$n = 15000$	$[0,51; 0,54]$	$[0,016; 0,019]$	$[0,77; 0,82]$	$[77,00; 77,68]$
GA	$\mu = \lambda = 10$	$[0,25; 0,29]$	$[0,025; 0,035]$	$[0,72; 0,83]$	$[0,52; 0,54]$
	$\mu = \lambda = 20$	$[0,32; 0,36]$	$[0,019; 0,026]$	$[0,75; 0,85]$	$[1,05; 1,11]$
	$\mu = \lambda = 30$	$[0,38; 0,42]$	$[0,018; 0,025]$	$[0,83; 0,94]$	$[1,59; 1,62]$
	$\mu = \lambda = 50$	$[0,45; 0,50]$	$[0,013; 0,019]$	$[0,80; 0,89]$	$[2,79; 2,94]$
	$\mu = \lambda = 75$	$[0,53; 0,57]$	$[0,012; 0,017]$	$[0,81; 0,90]$	$[4,13; 4,25]$
	$\mu = \lambda = 100$	$[0,58; 0,62]$	$[0,012; 0,018]$	$[0,84; 0,95]$	$[5,55; 5,65]$
	$\mu = \lambda = 150$	$[0,65; 0,69]$	$[0,009; 0,013]$	$[0,86; 0,95]$	$[8,41; 8,54]$
	$\mu = \lambda = 250$	$[0,72; 0,76]$	$[0,007; 0,011]$	$[0,83; 0,91]$	$[14,89; 15,11]$
	$\mu = \lambda = 500$	$[0,81; 0,85]$	$[0,004; 0,008]$	$[0,88; 0,96]$	$[34,64; 35,09]$
	$\mu = \lambda = 1000$	$[0,86; 0,89]$	$[0,005; 0,009]$	$[0,94; 1,01]$	$[84,53; 85,48]$
TS	$maxIt = 10$	$[0,12; 0,18]$	$[0,045; 0,061]$	$[0,51; 0,59]$	$[0,18; 0,23]$
	$maxIt = 25$	$[0,21; 0,28]$	$[0,029; 0,035]$	$[0,55; 0,63]$	$[0,58; 0,76]$
	$maxIt = 50$	$[0,25; 0,32]$	$[0,023; 0,029]$	$[0,61; 0,70]$	$[1,35; 1,74]$
	$maxIt = 75$	$[0,29; 0,37]$	$[0,027; 0,032]$	$[0,60; 0,72]$	$[2,07; 2,63]$
	$maxIt = 100$	$[0,35; 0,43]$	$[0,021; 0,026]$	$[0,64; 0,73]$	$[2,98; 3,72]$
TS_2	$maxIt = 100$	$[0,51; 0,58]$	$[0,013; 0,017]$	$[0,73; 0,82]$	$[6,59; 7,73]$
TS_5	$maxIt = 100$	$[0,73; 0,78]$	$[0,006; 0,008]$	$[0,84; 0,90]$	$[15,66; 17,44]$
TS_{10}	$maxIt = 100$	$[0,88; 0,91]$	$[0,002; 0,003]$	$[0,91; 0,96]$	$[33,99; 36,32]$
TS_{20}	$maxIt = 100$	$[0,96; 0,98]$	$[0,000; 0,001]$	$[0,98; 1,00]$	$[66,35; 69,86]$
PSO	$size = 20$	$[0,23; 0,31]$	$[0,024; 0,033]$	$[0,70; 0,72]$	$[0,76; 0,80]$
	$size = 40$	$[0,41; 0,49]$	$[0,011; 0,015]$	$[0,71; 0,73]$	$[1,34; 1,40]$

Tabelle B.1: Ergebnisse des Effizienzexperiments

Alg.	Parameter	F_1	MID	MS	Zeit in s
	$size = 60$	[0,50; 0,56]	[0,008; 0,011]	[0,71; 0,73]	[1,96; 2,02]
	$size = 100$	[0,58; 0,65]	[0,006; 0,008]	[0,71; 0,73]	[3,11; 3,15]
	$size = 150$	[0,63; 0,68]	[0,005; 0,007]	[0,71; 0,73]	[4,57; 4,67]
	$size = 250$	[0,70; 0,73]	[0,004; 0,006]	[0,71; 0,73]	[7,62; 7,77]
	$size = 500$	[0,75; 0,79]	[0,003; 0,004]	[0,71; 0,74]	[15,73; 15,98]
	$size = 1000$	[0,82; 0,84]	[0,003; 0,005]	[0,72; 0,74]	[40,84; 41,94]
	$size = 2000$	[0,87; 0,89]	[0,003; 0,007]	[0,72; 0,75]	[92,30; 94,26]
SSO	$size = 20$	[0,34; 0,40]	[0,021; 0,029]	[0,71; 0,75]	[0,80; 0,82]
	$size = 40$	[0,49; 0,55]	[0,016; 0,022]	[0,72; 0,77]	[1,50; 1,56]
	$size = 60$	[0,58; 0,63]	[0,012; 0,016]	[0,71; 0,75]	[2,12; 2,17]
	$size = 100$	[0,69; 0,73]	[0,011; 0,015]	[0,74; 0,79]	[3,48; 3,56]
	$size = 150$	[0,74; 0,78]	[0,011; 0,017]	[0,74; 0,79]	[5,10; 5,20]
	$size = 250$	[0,80; 0,83]	[0,013; 0,018]	[0,75; 0,81]	[9,10; 9,39]
	$size = 500$	[0,85; 0,87]	[0,013; 0,018]	[0,75; 0,81]	[16,62; 17,14]
	$size = 1000$	[0,91; 0,93]	[0,007; 0,011]	[0,78; 0,85]	[35,06; 35,28]
	$size = 2000$	[0,93; 0,96]	[0,004; 0,008]	[0,81; 0,88]	[78,70; 79,23]
ABC	$size = 10$	[0,12; 0,17]	[0,035; 0,041]	[0,71; 0,77]	[1,13; 1,18]
	$size = 20$	[0,21; 0,26]	[0,027; 0,031]	[0,76; 0,81]	[2,02; 2,07]
	$size = 30$	[0,27; 0,33]	[0,021; 0,025]	[0,79; 0,83]	[2,94; 3,00]
	$size = 50$	[0,34; 0,39]	[0,019; 0,021]	[0,80; 0,84]	[4,44; 4,53]
	$size = 75$	[0,41; 0,46]	[0,015; 0,017]	[0,80; 0,85]	[6,41; 6,48]
	$size = 100$	[0,45; 0,50]	[0,014; 0,016]	[0,81; 0,86]	[8,48; 8,58]
	$size = 150$	[0,53; 0,58]	[0,012; 0,014]	[0,81; 0,86]	[12,60; 12,76]
	$size = 200$	[0,59; 0,64]	[0,011; 0,013]	[0,83; 0,88]	[17,34; 17,80]
	$size = 400$	[0,72; 0,76]	[0,008; 0,011]	[0,83; 0,90]	[34,38; 35,00]
	$size = 800$	[0,82; 0,85]	[0,005; 0,007]	[0,86; 0,92]	[72,89; 74,34]

Tabelle B.2: Aggregierte Pareto-Front für das ITRAP

Admin	Router	LB	ERP	ERP-Blade	Storage Mgmt.	DB	DB-Blade	Storage	LAN	WAN	Power	A	C
1	1	1	1	1	1	1	1	1	1	1	1	.9859179	503039
1	1	1	1	1	1	1_{+1}	1	1	1	1	1	.9861179	503623
1	1	1	1	1	1	2_{+1}	1	1	1	1	1	.9866418	504259
1	1	1	1	1	1_{+1}	1	1	1	1	1	1	.9871348	505841
1	1	1_{+1}	1	1	1	1	1	1	1	1	1	.9880934	505911
1	1	1_{+1}	1	1	1	1_{+3}	1	1	1	1	1	.9881674	506902
1	1	1_{+1}	1_{+1}	1	1	1_{+1}	1	1	1	1	1	.9887562	508050
1	1	1_{+1}	1	1	1_{+1}	3	1	1	1	1	1	.9898436	508469
1	1	1_{+1}	1	1	1_{+1}	2_{+1}	1	1	1	1	1	.9903994	508840
1	1	1	1	1	1_{+1}	1_{+1}	1_{+1}	1	1	1	1	.9904545	510432
1	1	1_{+1}	2	1	1_{+1}	2	1	1	1	1	1	.9904709	510901
1	1	1_{+1}	1_{+1}	1	1_{+1}	2_{+1}	1	1	1	1	1	.9905929	511552
1	1	1_{+1}	1	1	1_{+1}	1	1_{+1}	1	1	1	1	.9908158	512629
1	1	1_{+1}	1_{+1}	1	1_{+1}	1_{+1}	1	1	1	1	1_{+1}	.9911330	513558
1	1	1_{+1}	1	1	1	2	1	1	2	1	1	.9913849	513727
1	1	1_{+1}	1	1	1_{+1}	3	1_{+1}	1	1	1	1	.9915406	514178
1	1	1_{+1}	1	1	1	1_{+1}	1	1	2	1	1	.9917612	514407
1	1	1_{+1}	1	1	1_{+1}	2_{+1}	1_{+1}	1	1	1	1	.9918931	514603
1	1	1	1	1	1_{+2}	1_{+1}	1	1	2	1	1	.9919329	516196
1	1	1_{+1}	1	1	1_{+1}	1	1	1	2	1	1	.9933770	516300
1	1	1_{+1}	1	1	1_{+1}	2_{+1}	1	1	2	1	1	.9934700	517210
1	1	1_{+1}	1	1	1_{+1}	1_{+1}	1	1	2	1	1	.9936550	517228
1	1	1_{+1}	1	1	1_{+2}	1_{+1}	1	1	2	1	1	.9937599	518810
1	1	1_{+1}	2	1	1_{+1}	1_{+1}	1	1	2	1	1	.9938925	518958
1	1	1_{+1}	1_{+1}	1	1_{+1}	1_{+1}	1	1	2	1	1	.9943772	519205
1	1	1_{+1}	1	1	1_{+1}	2	1_{+1}	1	2	1	1	.9955270	521298
1	1	1_{+1}	1	1	1_{+1}	1_{+1}	1_{+1}	1	2	1	1	.9955965	521951
1	1	1_{+1}	1_{+1}	1	1_{+1}	1_{+1}	1_{+1}	1	2	1	1	.9960199	523694
1	1	1_{+1}	2	1	1_{+1}	1_{+1}	1_{+1}	1	2	1	1	.9961461	524903
1	1	1_{+1}	3	1	1_{+1}	1_{+1}	1_{+1}	1	2	1	1	.9962754	526044
1	1	1_{+1}	1_{+1}	1	2_{+1}	1_{+1}	1_{+1}	1	2	1	1	.9963523	529625
1	1	1_{+1}	1	1_{+1}	1_{+1}	1	1_{+1}	1	2	1	1	.9970934	531873
1	1	1_{+1}	1	1_{+1}	1_{+1}	3_{+1}	1_{+1}	1	2	1	1	.9972424	533169
1	1	1_{+1}	1	1_{+1}	1_{+1}	2_{+2}	1_{+1}	1	2	1	1	.9975121	533211
1	1	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	1_{+1}	1	2	1	1	.9977724	534202

Tabelle B.2: Aggregierte Pareto-Front für das ITRAP

Admin	Router	LB	ERP	ERP-Blade	Storage Mgmt.	DB	DB-Blade	Storage	LAN	WAN	Power	A	C
1	1	1_{+1}	2	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1	2	1	1	.9981293	534809
1	1	1_{+1}	2_{+1}	1_{+1}	2	1_{+3}	1_{+1}	1	2	1	1	.9981677	541151
1	1	2_{+1}	3	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1	2	1	1	.9982118	542854
1	1	1_{+1}	2_{+1}	1_{+1}	1_{+1}	2_{+1}	2	1	2	1	1	.9983401	542899
1	1_{+1}	1_{+1}	2_{+1}	1_{+1}	2	1_{+1}	2	1	2	1	1	.9983431	557332
1	1	1_{+1}	1	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	1	1	.9987091	557809
1	1	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	1_{+1}	1_{+1}	2	1	1	.9990765	559238
1	1	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	1	1	.9992300	561307
1	1	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+3}	2	1_{+1}	2	1	1	.9993006	566248
1	1	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	2	1	1	.9993321	568861
1	1	1_{+1}	1_{+1}	1_{+1}	2	1_{+1}	1_{+1}	2	2	1	1	.9994096	570139
1	1	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	2	1	1	.9994291	571332
1	1	2_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	2	1	1	.9994955	573223
1	1	1_{+1}	2_{+1}	1_{+1}	1_{+1}	1_{+1}	1_{+1}	2	2	1	1_{+1}	.9995098	575924
1	1	1_{+2}	2_{+1}	1_{+1}	2	1_{+1}	1_{+1}	2	2	1	1	.9995213	577298
1	1	1_{+1}	2_{+1}	1_{+1}	2	1_{+1}	2	2	2	1	1	.9995316	577488
1	1	1_{+1}	2_{+1}	1_{+1}	1_{+1}	1_{+1}	2	2	2	1	1	.9995449	577634
1	1	1_{+1}	2_{+1}	1_{+1}	2_{+1}	1_{+1}	1_{+1}	2	2	1	1	.9995600	578123
1	1_{+1}	1_{+1}	2	1_{+1}	1_{+1}	1_{+1}	2	2	2	1	1	.9996068	581955
1	1_{+1}	1_{+1}	3	1_{+1}	1_{+1}	1_{+1}	2	2	2	1	1	.9996454	583013
1	1	2_{+1}	2_{+1}	1_{+1}	2_{+1}	1_{+1}	2	2	2	1	1_{+1}	.9996719	588316
1	1_{+1}	1_{+1}	2_{+1}	1_{+1}	1_{+2}	5	1_{+1}	2	2	1	1_{+1}	.9996969	588406
1	1_{+1}	1_{+1}	2_{+1}	1_{+1}	2_{+1}	1_{+1}	2	2	2	1	1	.9997418	591316
1	1_{+2}	1_{+1}	2_{+1}	1_{+1}	2	1_{+1}	2	2	2	1	1	.9997643	595104
1	2	1_{+1}	4_{+1}	1_{+1}	2	7	1_{+2}	2	2	1	1_{+1}	.9997692	601298
1	1_{+1}	1_{+1}	3_{+1}	2	2	6_{+1}	2	2	3	1	1	.9998348	608222
1	1_{+1}	1_{+1}	3_{+1}	2	2	7_{+1}	2	2	3	1	1	.9998480	609010
1	1_{+1}	1_{+1}	4_{+1}	2	2	6_{+2}	2	2	3	1	1	.9998653	612285
1	1_{+1}	1_{+2}	5_{+1}	2	2	6_{+2}	2	2	3	1	1	.9998935	617401
1	1_{+1}	2_{+1}	5_{+2}	2	2	4_{+1}	2	2	3	1	1	.9999020	620374
1	1_{+1}	2_{+1}	4_{+2}	2	3	4_{+1}	2	2	3	1	1	.9999079	622422
1	1_{+1}	2_{+1}	4_{+2}	2	2	5_{+2}	2	2	3	1	1_{+1}	.9999237	623200
1	1_{+1}	2_{+1}	5_{+1}	2	2	7_{+1}	2_{+1}	2	3	1	1_{+1}	.9999372	628639
1	1_{+1}	2_{+1}	5_{+1}	2	3	7_{+1}	2_{+1}	2	3	1	1_{+1}	.9999402	633390
1	1_{+1}	2_{+2}	6_{+1}	2	2_{+1}	6_{+2}	2	2	3	1	1_{+1}	.9999404	640725

Tabelle B.2: Aggregierte Pareto-Front für das ITRAP

Admin	Router	LB	ERP	ERP-Blade	Storage Mgmt.	DB	DB-Blade	Storage	LAN	WAN	Power	A	C
1	1_{+1}	2_{+2}	5_{+1}	2	2_{+1}	6_{+2}	3	2	3	1	1_{+1}	.9999422	647455
1	2_{+1}	2_{+2}	5_{+1}	2	2	6_{+2}	2	2	3	1	1_{+1}	.9999448	647773
1	2_{+1}	2_{+2}	5_{+1}	2	2	6_{+2}	2	2	3	1	1_{+2}	.9999465	659116
1	2_{+1}	3_{+1}	5_{+2}	2	2_{+2}	3_{+3}	2_{+1}	2_{+1}	3	1	2_{+1}	.9999489	681685
1	2_{+1}	2_{+2}	5_{+2}	2	2_{+2}	3_{+4}	2_{+1}	2_{+1}	3	1	2_{+1}	.9999502	682819
1	2_{+2}	2_{+2}	5_{+2}	2	2_{+2}	3_{+4}	2_{+1}	2_{+1}	3	1	2_{+2}	.9999525	705574
1	2_{+2}	2_{+3}	7_{+1}	2	2_{+2}	2_{+4}	2_{+3}	2_{+1}	4	1	2_{+1}	.9999557	712815
1	2_{+1}	2_{+2}	5_{+1}	2	2_{+1}	4_{+3}	2_{+2}	2	3	1	1_{+1}	.9999566	725909
1	2_{+1}	2_{+1}	5_{+2}	2	2_{+1}	4_{+4}	3_{+1}	2	3	1	1_{+1}	.9999575	742396
1	2_{+1}	2_{+2}	5_{+2}	2_{+1}	2_{+1}	4_{+3}	4	2_{+1}	4	1	1_{+3}	.9999576	795193
2	2_{+1}	2_{+3}	7_{+1}	2_{+1}	2_{+2}	2_{+4}	2_{+3}	2_{+1}	3	1	2_{+1}	.9999595	1058974

Literaturverzeichnis

[AAMY13] R. Abdelkader, Z. Abdelkader, R. Mustapha, and M. Yamani. *Search Algorithms for Engineering Optimization*, chapter Optimal Allocation of Reliability in Series Parallel Production System, pages 241–258. InTech, 2013.

[AH14] M. A. Ardakan and A. Z. Hamadani. Reliability–redundancy allocation problem with cold-standby redundancy strategy. *Simulation Modelling Practice and Theory*, 42:107–118, 2014.

[AHA15] M. A. Ardakan, A. Z. Hamadani, and M. Alinaghian. Optimizing bi-objective redundancy allocation problem with a mixed redundancy strategy. *ISA Transactions*, 55:116–128, 2015.

[Bäc96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

[Ban98] J. Banks. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons, 1998.

[BCDGG00] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. *IEEE Transactions on Computers*, 49(3):230–245, March 2000.

[BCH13] L.A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2 edition, 2013.

[BHS+15] S. Bosse, J. Hintsch, C. Schulz, M. Splieth, H. Müller, and K. Turowski. Evaluating IT Service Design Alternatives With Respect to Availability, Response Times and Costs. In S. Bosse, M. E. Elsaid, F. Feinbube, and H. Müller, editors, *Proceedings of the 2nd HPI Cloud Symposium "Operating the Cloud" 2014*, volume 94 of *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik*, pages 1–14. Universitätsverlag Potsdam, 2015.

[Bos13] S. Bosse. Predicting an IT Service's Availability with Respect to Operator Errors. In *Proceedings of the 19th Americas Conference on Information Systems (AMCIS)*, Chicago, IL, USA, 2013.

[Bos14] S. Bosse. Vergleich Analytischer Vorhersageansätze für die Verfügbarkeit von IT-Diensten. In Dennis Kundisch, Leena Suhl, and Lars Beckmann, editors, *Tagungsband Multikonferenz Wirtschaftsinformatik 2014*, pages 2243–2255. Universität Paderborn, February 2014.

[BST13] S. Bosse, M. Splieth, and K. Turowski. Vorhersagemodell für die Verfügbarkeit von IT-Services aus Anwendungssystemlandschaften. In *11th International Conference on Wirtschaftsinformatik (WI)*, Leipzig, Germany, February 27-March 1 2013.

[BST14a] S. Bosse, C. Schulz, and K. Turowski. Predicting Availability and Response Times of IT Services. In *Proceedings of the 22nd European Conference on Information Systems (ECIS)*, Tel Aviv, Israel, June 9-11 2014.

[BST14b] S. Bosse, M. Splieth, and K. Turowski. Model-Based Prediction of IT Service Availability - A Literature Review. In *Proceedings of the 13th International Conference on Modeling and Applied Simulation*, pages 141–148, Bordeaux, France, September 10-12 2014. DIME Università di Genova.

[BST15] S. Bosse, M. Splieth, and K. Turowski. Optimizing IT Service Costs With Respect to the Availability Service Level Objective. In *10th International Conference on Availability, Reliability and Security (ARES)*, pages 20–29, 2015.

[BST16] S. Bosse, M. Splieth, and K. Turowski. Multi-Objective Optimization of IT Service Availability and Costs. *Reliability Engineering & System Safety*, 147:142–155, March 2016.

[BT13] S. Bosse and K. Turowski. Automatic Generation of Petri Net-Based Availability Prediction Models for IT Services. In Roman Beck and Ada Scupola, editors, *Proceedings of the Pre-ICIS SIGSVC Workshop 'Delivering and Managing Services in Systems of Service Systems'*, Milano, Italy, December 15 2013.

[Can11] D. Cannon. *ITIL Service Strategy 2011 Edition*. The Stationery Office, 2011.

[Car03] Nicholas G. Carr. IT doesn't matter. *Harvard Business Review*, 81(5):41–49, May 2003.

[Che92] M.-S. Chern. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*, 11:309–315, 1992.

[Che06] T.-C. Chen. IAs based approach for reliability redundancy allocation problems. *Applied Mathematics and Computation*, 182:1556–1567, 2006.

[CK90] D.-H. Chi and W. Kuo. Optimal design for software reliability and development cost. *IEEE Journal on Selected Areas in Communications*, 8(2):276–282, February 1990.

[CK06] D.W. Coit and A. Konak. Multiple weighted objectives heuristic for the redundancy allocation problem. *IEEE Transactions on Reliability*, 55:551–558, 2006.

[CMC13] D. Cao, A. Murat, and R. B. Chinnam. Efficient exact optimization of multi-objective redundancy allocation problems in series-parallel systems. *Reliability Engineering & System Safety*, 111:154–163, 2013.

[CMT89] G. Ciardo, J.K. Muppala, and K.S. Trivedi. SPNP: Stochastic Petri Net Package. In *Proceedings of the 3rd International Workshop PNPM*, pages 142–151. IEEE Computer Society, 1989.

[CMT+12] G. Callou, P. Maciel, D. Tutsch, J. Araújo, J. Ferreira, and R. Souza. A Petri Net-Based Approach to the Quantification of Data Center Dependability. In Pawel Pawlewski, editor, *Petri Nets - Manufacturing and Computer Science*, chapter 14, pages 313–336. InTech, 2012.

[CNRK13] A. Chambari, A.A. Najafi, S.H. Rahmati, and A. Karimi. An efficient simulated annealing algorithm for the redundancy allocation problem with a choice of redundancy strategies. *Reliability Engineering & System Safety*, 119:158–164, 2013.

[CRNK12] A. Chambari, S.H.A. Rahmati, A.A. Najafi, and A. Karimi. A bi-objective model to optimize reliability and cost of system with a choice of redundancy strategies. *Computers & Industrial Engineering*, 63:109–119, 2012.

[CS96a] D.W. Coit and A.E. Smith. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability*, 45:254–266, 1996.

[CS96b] D.W. Coit and A.E. Smith. Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach. *Computers & Operations Research*, 23:515–526, 1996.

[CV09] C. Chellappan and G. Vijayalakshmi. Dependability modeling and analysis of hybrid redundancy systems. *International Journal of Quality & Reliability Management*, 26:76–96, 2009.

[CV15] M. Caserta and S. Voß. An exact algorithm for the reliability redundancy allocation problem. *European Journal of Operational Research*, 244:110–116, 2015.

[CY05] T.-C. Chen and P.-S. You. Immune algorithms-based approach for redundant reliability problems with multiple component choices. *Computers in Industry*, 56:195–205, 2005.

[DAPM00] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, volume 1917 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, September 2000. Springer.

[DoD81] Military Standard: Reliability Modeling and Prediction (MIL-STD-756B), 1981.

[DZKD15] A. Dolatshahi-Zand and K. Khalili-Damghani. Design of SCADA water resource management control center by a bi-objective redundancy allocation problem and particle swarm optimization. *Reliability Engineering & System Safety*, 133:11–21, 2015.

[ENC+12] V. C. Emeakaroha, M. A. S. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. F. De Rose. Towards autonomic detection of SLA violations in Cloud infrastructures. *Future Generation Computer Systems*, 28(7):1017–1029, 2012.

[ERS+07] J. Eckert, N. Repp, S. Schulte, R. Berbner, and R. Steinmetz. An approach for capacity planning of web service workflows. In *Proceedings of the 13th Americas Conference on Information Systems (AMCIS)*, 2007.

[FAM14] M. J. Feizollahi, S. Ahmed, and M. Modarres. The robust redundancy allocation problem in series-parallel systems with budgeted uncertainty. *IEEE Transactions on Reliability*, 63(1):239–250, March 2014.

[FF95] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[FHL68] D.E. Fyffe, W.W. Hines, and N.K. Lee. System reliability allocation and a computational algorithm. *IEEE Transactions on Reliability*, 17:64–69, 1968.

[Fin14] G.A. Fink. *Markov Models for Pattern Recognition.* Springer, 2014.

[FJK14] U. Franke, P. Johnson, and J. König. An architecture framework for enterprise IT service availability analysis. *Software and Systems Modeling*, 13:1417–1445, October 2014.

[Fra12] U. Franke. Optimal IT Service Availability: Shorter Outages, or Fewer? *IEEE Transactions on Network and Service Management*, 9:22–33, 2012.

[FWB07] X. Fan, W.-D. Weber, and L.A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th International Symposium on Computer Architecture*, pages 13–23, San Diego, CA, USA, June 9-13 2007.

[GRSV14] H. Garg, M. Rani, S. P. Sharma, and Y. Vishwakarma. Bi-objective optimization of the reliability-redundancy allocation problem for series-parallel system. *Journal of Manufacturing Systems*, 33:335–347, 2014.

[GS13] H. Garg and S.P. Sharma. Multi-objective reliability-redundancy allocation problem using particle swarm optimization. *Computers & Industrial Engineering*, 64:247–255, 2013.

[GSFT08] M. Grottke, H. Sun, R. M. Fricks, and K. S. Trivedi. Ten Fallacies of Availability and Reliability Analysis. In T. Nanya, F. Maruyama, A. Pataricza, and M. Malek, editors, *Service Availability*, volume 5017 of *Lecture Notes in Computer Science*, pages 187–206. Springer Berlin Heidelberg, 2008.

[GT05] M. Grottke and K.S. Trivedi. A classficiation of software faults. *Journal of Reliability Engineering Association of Japan*, 27(7):425–438, 2005.

[GY06] M. Gen and Y.S. Yun. Soft computing approach for reliability optimization: State-of-the-art survey. *Reliability Engineering & System Safety*, 91:1008–1026, 2006.

[HMPR04] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28:75–105, 2004.

[HSM04] G. A. Hoffmann, F. Salfner, and M. Malek. Advanced failure prediction in complex software systems. Technical report, Informatik-Bericht 172 der Humboldt-Universität zu Berlin, 2004.

[HTB05] A. Hochstein, G. Tamm, and W. Brenner. Service Oriented IT Management: Benefit, Cost and Success Factors. In *Proceedings of the European Conference on Information Systems (ECIS) 2005*, number 98, 2005.

[Hun11] L. Hunnebeck. *ITIL Service Design 2011 Edition.* The Stationery Office, Norwich, UK, 2011.

[HY11] Y.-C. Hsieh and P.-S. You. An effective immune based two-phase approach for the optimal reliability–redundancy allocation problem. *Applied Mathematics and Computation*, 218:1297–1307, 2011.

[HY12] T.-J. Hsieh and W.-C. Yeh. Penalty guided bees search for redundancy allocation problems with a mix of components in series–parallel systems. *Computers & Operations Research*, 39:2688–2704, 2012.

[IN08] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, 7:49–65, 2008.

[Inf12] Information Systems Audit and Control Association. *COBIT 5.* ISACA, 2012.

[ISO11] ISO/IEC 20000-1, International Organization for Standardization, 2011.

[ISO14] ISO/IEC 25000:2014, International Organization for Standardization, 2014.

[ITSMF05] Netherlands Information-Technology-Service-Management-Forum. *Foundations of IT Service Management basierend auf ITIL V3.* Van Haren Publishing, 2005.

[ITU08] ITU-T E.800 : Definitions of terms related to quality of service, International Telecommunications Union, 2008.

[Jew08] D. Jewell. *Performance Modeling and Engineering*, chapter Performance Engineering and Management Method - A Holistic Approach to Performance Engineering, pages 29–55. Springer US, 2008.

[JZMY14] G. Jiansheng, W. Zutong, Z. Mingfa, and W. Ying. Uncertain multiobjective redundancy allocation problem of repairable systems based on artificial bee colony algorithm. *Chinese Journal of Aeronautics*, 27(6):1477–1487, 2014.

[KDA12] K. Khalili-Damghani and M. Amiri. Solving binary-state multi-objective reliability redundancy allocation series-parallel problem using efficient epsilon-constraint, multi-start partial bound enumeration algorithm, and DEA. *Reliability Engineering & System Safety*, 103:35–44, 2012.

[KJ62] J. D. Kettelle Jr. Least-cost allocations of reliability investment. *Operations Research*, 10(2):249–265, March - April 1962.

[KKCB08] S. Kulturel-Konak, D. W. Coit, and F. Baheranwala. Pruned pareto-optimal sets for the system redundancy allocation problem based on multiple prioritized objectives. *Journal of Heuristics*, 14:335–357, 2008.

[KKSC03] S. Kulturel-Konak, A.E. Smith, and D.W. Coit. Efficiently solving the redundancy allocation problem using tabu search. *IIE Transactions*, 35:515–526, 2003.

[KKSN06] S. Kulturel-Konak, A. E. Smith, and B. A. Normal. Multi-objective tabu search using a multinomial probability mass function. *European Journal of Operational Research*, 169:918–931, 2006.

[KP00] W. Kuo and V. R. Prasad. An annotated overview of system-reliability optimization. *IEEE Transactions on Reliability*, 49(2):176–187, June 2000.

[KPJ13] G. Kanagaraj, S.G. Ponnambalam, and N. Jawahar. A hybrid cuckoo search and genetic algorithm for reliability–redundancy allocation problems. *Computers & Industrial Engineering*, 66:1115–1124, 2013.

[Krc15] H. Krcmar. *Informationsmanagement*. Springer, 6th edition, 2015.

[Kwa78] H. Kwakernaak. Fuzzy random variables-I. Definitions and theorems. *Information Sciences*, 15(1):1–29, 1978.

[Lü14] Lünendonk. Führende IT-Service-Unternehmen in Deutschland. Pressemitteilung der Lünendonk GmbH IT-22-05-14, 2014.

[LA90] P.A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*. Springer-Verlag Wien, 2nd edition, 1990.

[Lap95] J.-C. Laprie. Dependable computing: Concepts, limits, challenges. In *25th International Symposium On Fault-Tolerant Computing*, pages 42–54, 1995.

[LD09] I. D. Lins and E. L. Droguett. Multiobjective optimization of availability and cost in repairable systems design via genetic algorithms and discrete event simulation. *Pesquisa Operacional*, 29:43–66, 2009.

[LD11] I. D. Lins and E. L. Droguett. Redundancy allocation problems considering systems with imperfect repairs using multi-objective genetic algorithms and discrete event simulation. *Simulation Modelling Practice and Theory*, 19:362–381, 2011.

[Lew99] L. Lewis. Service Level Management Definition, Architecture and Research Challenges. In *IEEE Global Telecommunications Conference*, pages 1974–1978, 1999.

[Lit06] B. Littlewood. Comments on 'Reliability and performance analysis for fault-tolerant programs consisting of versions with different characteristics' by Gregory Levitin. *Reliability Engineering and System Safety*, 91:119–120, 2006.

[LPG11] J.-Q. Li, Q.-K. Pan, and K.-Z. Gao. Pareto-baed discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling systems. *International Journal of Advanced Manufacturing Technology*, 55:1159–1169, 2011.

[LS04] Y.-C. Liang and A.E. Smith. An Ant Colony Optimization Algorithm for the Redundancy Allocation Problem (RAP). *IEEE Transactions on Reliability*, 53:417–423, 2004.

[Mal08] Miroslaw Malek. Predictive algorithms and technologies for availability enhancement. In T. Nanya, F. Maruyama, A. Pataricza, and M. Malek, editors, *5th International Service Availability Symposium (ISAS)*, volume 5017 of *Lecture Notes in Computer Science*, pages 17–19, Tokyo, Japan, 2008. Springer Verlag Berlin Heidelberg.

[MBB+89] M. A. Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 15(7):832–846, July 1989.

[MG11] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology - Special Publication*, 800-145:1–3, 2011.

[Mil10] N. Milanovic. *Models, Methods and Tools for Availability Assessment of IT Services and Business Processes*. Habilitation - Technische Universität Berlin, 2010.

[Mis71] K.B. Misra. Dynamic programming formulation of the redundancy allocation problem. *International Journal of Mathematical Education in Science and Technology*, 2:207–215, 1971.

[MM11] N. Milanovic and B. Milic. Automatic generation of service availability models. *IEEE Transactions on Service Computing*, 4(1):56–69, 2011.

[MSB12] S. K. Mahato, L. Sahoo, and A.. Bhunia. Reliability-redundancy optimization problem with interval valued reliabilities of components via genetic algorithm. *Journal of Information and Computer Science*, 7(4):284–295, 2012.

[MT06] K. Mishra and K. Trivedi. Model based approach for autonomic availability management. In D. Penkler, M. Reitenspieß, and F.. Tam, editors, *3rd International Service Availability Symposium*, volume 4328 of *Lecture Notes in Computer Science*, pages 1–16, Helsinki, Finland, May 15-16 2006. Springer Berlin Heidelberg.

[NM81] Y. Nakagawa and S. Miyazaki. Surrogate constraints algorithm for reliability optimization problems with two constraints. *IEEE Transactions on Reliability*, R-30(2):175–180, June 1981.

[NOB+04] K. Nagaraja, F. Oliveira, R. Bianchini, R.P. Martin, and T.D. Nguyen. Understanding and dealing with operator mistakes in internet services. In *6th Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[ODAL14] A.-C. Orgerie, M. D. De Assuncao, and L. Lefevre. A survey on techniques for improving the energy efficiency of large scale distributed systems. *ACM Computing Surveys*, 46(4):1–35, December 2014.

[OGP03] D. Oppenheimer, A. Ganapathi, and D.A. Patterson. Why do internet services fail, and what can be done about it? In *4th Usenix Symposium on Internet Technologies and Systems (USITS)*, 2003.

[OKJN07] J. Onishi, S. Kimura, R.J.W. James, and Y. Nakagawa. Solving the redundancy allocation problem with a mix of components using the improved surrogate constraint method. *IEEE Transactions on Reliability*, 56(1):94–101, Marchl 2007.

[ONG08] M. Ouzineb, M. Nourelfath, and M. Gendreau. Tabu search for the redundancy allocation problem of homogenous series–parallel multi-state systems. *Reliability Engineering & System Safety*, 93:1257–1272, 2008.

[ONG10] M. Ouzineb, M. Nourelfath, and M. Gendreau. An efficient heuristic for reliability design optimization problems. *Computers & Operations Research*, 37:223–235, 2010.

[PC95] L. Painton and J. Campbell. Genetic algorithms in optimization of system reliability. *IEEE Transactions on Reliability*, 44:172–178, 1995.

[PS05] C.D. Patel and A.J.f Shah. Cost model for planning, development: and operation of a data center. Technical report, Hewlett-Packard Laboratories Palo Alto, 2005.

[PTRC08] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24:45–78, 2008.

[PWB07] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[RD07] K. Rinsaka and T. Dohi. A faster estimation algorithm for periodic preventive rejuvenation schedule maximizing system availability. In M. Malek, M. Reitenspieß, and A.P.A. van Moorsel, editors, *4th International Service Availability Symposium (ISAS)*, volume 4526 of *Lecture Notes in Computer Science*, pages 94–109. Springer-Verlag Berlin Heidelberg, 2007.

[RMC04] J. E. Ramirez-Marquez and D. W. Coit. A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems. *Reliability Engineering and System Safety*, 83:341–349, 2004.

[RMR97] V. Ravi, B.S.N. Murty, and P.J. Reddy. Nonequilibrium simulated annealing-algorithm applied to reliability optimization of complex system. *IEEE Transactions on Reliability*, 46:233–239, 1997.

[RvdM15] J. Rivera and R. van der Meulen. Gartner's 2015 Hype Cycle for Emerging Technologies Identifies the Computing Innovations That Organizations Should Monitor, August 18 2015. [accessed on 11.01.2016].

[Sar10] R.G. Sargent. Verification and validation of simulation models. In B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, editors, *Proceedings of the 2010 Winter Simulation Conference*, pages 130–143, 2010.

[SBR10] L. Sahoo, A.K. Bhunia, and D. Roy. A genetic algorithm based reliability redundancy optimization for interval valued reliabilities of components. *Journal of Applied Quantitative Methods*, 5:270–287, 2010.

[SBST15] M. Splieth, S. Bosse, C. Schulz, and K. Turowski. Analyzing the effects of load distribution algorithms on energy consumption of servers in cloud data centers. In *Proceedings of the 12th International Conference on Wirtschaftsinformatik (WI)*, 2015.

[SCC02] Ruhul Sarker and Carlos A. Coello Coello. Assessment methodologies for multiobjective evolutionary algorithms. In *Evolutionary Optimization*, volume 48 of *International Series in Operations Research & Management Science*, pages 177–195. Springer US, 2002.

[SDKS13] M. Silic, G. Delac, I. Krka, and S. Srbljic. Scalable and accurate prediction of availability of atomic web services. *IEEE Transactions on Service Computing*, 2013.

[SH02] P. Stahlknecht and U. Hasenkamp. *Einführung in die Wirtschaftsinformatik*. Springer Berlin Heidelberg, 2002.

[Sho02] M.L. Shooman. *Reliability of Computer Systems and Networks – Fault Tolerance, Analysis, and Design*. John Wiley & Sons New York, New York, NY, USA, 2002.

[SKK08] A. Sachdeva, D. Kumar, and P. Kumar. Reliability analysis of pulping system using Petri nets. *International Journal of Quality & Reliability Management*, 25:860–877, 2008.

[Sol14] R. Soltani. Reliability optimization of binary state non-repairable systems: A state of the art survey. *International Journal of Industrial Engineering Computations*, 5:339–364, 2014.

[SPW11] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM Errors in the Wild: A Large-Scale Field Study. *Communications of the ACM*, 54:100–107, 2011.

[SS12] S. J. Sadjadi and R. Soltani. Alternative design redundancy allocation using an efficient heuristic and a honey bee mating algorithm. *Expert Systems with Applications*, 39:990–999, 2012.

[SS15] S. J. Sadjadi and R. Soltani. Minimum–Maximum regret redundancy allocation with the choice of redundancy strategy and multiple choice of component type under uncertainty. *Computers & Industrial Engineering*, 2015.

[STP12] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems*. Springer Science & Business Media, 2012.

[SWCC12] T. Sooktip, N. Wattanapongsakorn, D.W. Coit, and N. Chatwattanasiri. Multi-objective optimization for k-out-of-n redundancy allocation problem. In *International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE)*, pages 1050–1054, Chengdu, China, June 15-18 2012. IEEE.

[TCD+08] K. Trivedi, G. Ciardo, B. Dasarathy, M. Grottke, R. Matias, A. Rindos, and B. Vashaw. Achieving and assuring high availability. In T. Nanya, F. Maruyama, A. Pataricza, and M. Malek, editors, *5th International Service Availability Symposium (ISAS)*, volume 5017 of *Lecture Notes in Computer Science*, pages 20–25, Tokyo, Japan, 2008. Springer Verlag Berlin Heidelberg.

[TK11] D. Terlit and H. Krcmar. Generic Performance Prediction for ERP and SOA Applications. In *Proceedings of the 18th European Conference on Information Systems (ECIS)*, 2011.

[TLZ09] Z. Tian, G. Levitin, and M.J. Zuo. A joint reliability–redundancy optimization approach for multi-state series–parallel systems. *Reliability Engineering & System Safety*, 94:1568–1576, 2009.

[TY99] T. Taguchi and T. Yokota. Optimal design problem of system reliability with interval coefficient using improved genetic algorithms. *Computers & Industrial Engineering*, 37:145–149, 1999.

[TY00] K. Tokuno and S. Yamada. Markovian availability modeling for software-intensive systems. *International Journal of Quality & Reliability Management*, 17:200–212, 2000.

[VV13] E. Valian and E. Valian. A cuckoo search algorithm by lévy flights for solving reliability redundancy allocation problems. *Engineering Optimization*, 45(11):1273–1286, 2013.

[WCTY09] Z. Wang, T: Chen, K. Tang, and X. Yao. A multi-objective approach to redundancy allocation problem in parallel-series systems. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 582–589, Trondheim, Norway, May 18-21 2009. IEEE.

[WL12] L. Wang and L.-P. Li. A coevolutionary differential evolution with harmony search for reliability–redundancy optimization. *Expert Systems with Applications*, 39:5271–5278, 2012.

[WW09] S. Wang and J. Watada. Modelling redundancy allocation for a fuzzy random parallel-series system. *Journal of Computational and Applied Mathematics*, 232:539–557, 2009.

[Yeh14] W.-C. Yeh. Orthogonal simplified swarm optimization for the series–parallel redundancy allocation problem with a mix of components. *Knowledge-Based Systems*, 64:1–12, 2014.

[YH11] W.-C. Yeh and T.-J. Hsieh. Solving reliability redundancy allocation problems using an artificial bee colony algorithm. *Computers & Operations Research*, 38:1465–1473, 2011.

[Zai13] D. A. Zaitsev. Toward the Minimal Universal Petri Net. *IEEE Transactions on Systems, Man, and Cybernetics*, 2013.

[ZB03] R. Zarnekow and W. Brenner. A product-based information management approach. In *Proceedings of the European Conference on Information Systems (ECIS) 2003*, 2003.

[ZBGD10] V. Zille, C. Bérenguer, A. Grall, and A. Despujols. Simulation of maintained multicomponent systems for dependability assessment. In P. Faulin, A. Juan, S. Martorell, and J.E. Ramírez-Márquez, editors, *Simulation Methods for*

Reliability and Availability of Complex Systems, chapter 12, pages 253–272. Springer, Berlin, Heidelberg, 2010.

[ZDT00] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

[ZGLW11] D. Zou, L. Gao, S. Li, and J. Wu. An effective global harmony search algorithm for reliability problems. *Expert Systems with Applications*, 38:4642–4648, 2011.

[ZHA14] H. Zoulfaghari, A. Z. Hamadani, and M. A. Ardakan. Bi-objective redundancy allocation problem for a system with mixed repairable and non-repairable components. *ISA Transactions*, 53:17–24, 2014.

[Zia13] M. Ziaee. Optimal Redundancy Allocation in Hierarchical Series–Parallel Systems Using Mixed Integer Programming. *Applied Mathematics*, 4:79–83, 2013.

[ZL04] R. Zhao and B. Liu. Redundancy optimization problems with uncertainty of combining randomness and fuzziness. *European Journal of Operational Research*, 157:716–735, 2004.

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 29. April 2016

. . . . . .
Sascha Bosse