Dienstgüteunterstützung für Service-orientierte Workflows

Vom Fachbereich
Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

Dissertationsschrift

von

Dipl.-Wirtsch.-Inform. Rainer Berbner

geboren am 4. November 1976 in Mannheim

Darmstadt 2007
Hochschulkennziffer D-17

Vorsitzender: Prof. Dr.-Ing. Jürgen Stenzel
Erstreferent: Prof. Dr.-Ing. Ralf Steinmetz
Korreferent: Prof. Dr. rer. nat. Paul Müller

Tag der Einreichung: 3. April 2007
Tag der Disputation: 20. Juni 2007

Danksagung

Die vorliegende Dissertationsschrift entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am von Prof. Dr.-Ing. Ralf Steinmetz geleiteten Fachgebiet Multimedia Kommunikation (KOM) an der Technischen Universität Darmstadt.

An erster Stelle möchte ich mich ganz herzlich bei Prof. Dr.-Ing. Ralf Steinmetz für die Betreuung und Förderung meiner Arbeit bedanken. Besonders bedanken möchte ich mich in diesem Zusammenhang dafür, dass Prof. Dr.-Ing. Ralf Steinmetz mir sehr früh die Möglichkeit gegeben hat, Verantwortung an seinem Fachgebiet zu übernehmen. Ebenfalls bedanken möchte ich mich bei Herrn Prof. Dr. rer. nat. Paul Müller (TU Kaiserslautern) für die Übernahme des Korreferats.

Mein besonderer Dank gilt meinen Kollegen der Forschungsgruppe IT-Architekturen am Fachgebiet KOM Nicolas Repp, Julian Eckert und Stefan Schulte sowie meinen ehemaligen Kollegen Ivan Martinovic und Alejandro Perez für die hervorragende, freundschaftliche Zusammenarbeit. Auch möchte ich mich bei Dr.-Ing. Oliver Heckmann und Dr. Andreas U. Mauthe bedanken, deren Ideen und Anregungen diese Arbeit maßgeblich mitgeprägt haben. Andreas sei darüber hinaus gedankt, dass er mir einen Forschungsaufenthalt am InfoLab 21 an der Lancaster University ermöglicht hat. In dieser Zeit konnten wesentliche Teile der vorliegenden Dissertationsschrift in die endgültige Form gebracht werden.

Mein Dank gilt allen Kollegen und ehemaligen Kollegen am Fachgebiet KOM für die Unterstützung in den vergangenen Jahren. Besonders genannt seien hier Dr.-Ing. Matthias Hollick, Dr.-Ing. Andreas Faatz, Nicolas Liebau und Dr. Wolfgang Johannsen für deren wertvolle Ratschläge und konstruktive Diskussionen. Ebenso gilt mein Dank Karola Schork-Jakobi, die uns bei der Organisation von Veranstaltungen im Rahmen des E-Finance Lab e.V. tatkräftig unterstützt hat sowie Sabine Kräh im Bereich Finanzen und unserer Sekretärin Frau Kolb. Herzlichen Dank auch allen, die sich die Zeit genommen haben, diese Arbeit zu lesen und mir wertvolles Feedback zu geben.

Bedanken möchte ich mich auch bei denen von mir betreuten Studien- und Diplomarbeitern. Besonders genannt sei hier Michael Spahn, der mich auch nach seiner Diplomarbeit als HiWi an unserem Fachgebiet auf hervorragende Weise unterstützt hat.

Bedanken möchte ich mich auch bei allen Professoren und Kollegen im E-Finance Lab e.V., allen voran Herrn Prof. Dr. Wolfgang König sowie seinen Mitarbeitern im Cluster 1, für die Zusammenarbeit im spannenden Umfeld des E-Finance Lab e.V..

Meinen herzlichen Dank möchte ich weiterhin an die Projektpartner und insbesondere die dortigen Ansprechpartner für die konstruktive Zusammenarbeit in den vergangenen vier Jahren richten. Neben den Partnerunternehmen im E-Finance Lab e.V. sind dies die Unternehmen Amadee AG, btexx business technologies GmbH, jCOM1 AG, Siemens AG CT IC 2, Softpro GmbH, T-Systems Business Services GmbH und T-Systems Multimedia Solutions GmbH.

Dem IDG Verlag und den Kollegen des SOA-Expertenrates der Computerwoche gilt mein Dank für die spannenden Diskussionsrunden und Veranstaltungen zum Thema Service-orientierte Architekturen.

Nicht zuletzt möchte ich mich bei meinen Eltern bedanken, die mir durch ihre Unterstützung das Studium an der TU Darmstadt sowie die damit verbundenen Auslandsaufenthalte an der TU Eindhoven und ein sechsmonatiges Praktikum in den USA ermöglicht haben.

Inhaltsverzeichnis

Danksagung **2**

Inhaltsverzeichnis **4**

1 Einführung **8**
1.1 Motivation 8
1.2 Beitrag dieser Arbeit 9
1.3 Aufbau der Arbeit 10

2 Grundlagen **13**
2.1 Service-orientierte Architektur (SOA) 13
2.1.1 Rollen 13
2.1.2 Servicearten 14
2.1.3 Merkmale 15
2.1.4 Anforderungen 16
2.2 Web Service Technologie 18
2.2.1 Definition 18
2.2.2 SOAP 19
2.2.3 Web Service Description Language (WSDL) 21
2.2.4 Universal Description, Discovery and Integration (UDDI) 24
2.2.5 Business Process Execution Language for Web Services (BPEL4WS) 26
2.3 Geschäftsprozesse und Workflows 30
2.3.1 Geschäftsprozesse 30
2.3.2 Workflows 32
2.3.3 Service-orientierte Workflows 33
2.4 Dienstgüte 35
2.4.1 Definition 35
2.4.2 Dienstgüte für Service-orientierte Workflows 36
2.4.3 Service Level Agreements (SLAs) 40

3 WSQoSX – Eine Dienstgüteunterstützende Web Service Architektur **42**
3.1 Anforderungen an eine Web Service Architektur 42
3.2 Beteiligte Akteure und Anwendungsszenario 43
3.2.1 Beteiligte Akteure 43
3.2.2 Anwendungsszenario 45
3.3 Vorgehensmodell 45
3.4 Architektur und Funktionsweise 48
3.4.1 Portalbasierte Anmeldung 49
3.4.2 Kategorisierung 51

3.4.3 Bewertung der Web Services ... 52
3.4.4 Auswahl und Ausführung ... 56
3.4.5 Monitoring und Accounting ... 59
3.4.6 Deaktivierung und Löschung ... 62
3.5 Umsetzung und Implementierung ... 63
3.6 Zusammenfassung ... 68

4 Optimierung von Web Service Workflows ... 69
4.1 Problemstellung ... 69
4.1.1 Workflow, Kategorie und Ausführungsplan ... 69
4.1.2 Nutzen eines Ausführungsplans ... 71
4.1.3 Anwendungsszenario ... 72
4.2 Optimierungsmodelle und Lösungsverfahren ... 73
4.2.1 Einführung ... 74
4.2.2 Lineare Optimierungsmodelle ... 74
4.2.3 Ganzzahlige lineare Optimierungsmodelle ... 75
4.2.4 Heuristiken ... 76
4.2.5 Simulated Annealing ... 77
4.3 Mathematisches Modell ... 78
4.4 Heuristiken zur Optimierung von Web Service Workflows ... 83
4.4.1 Motivation ... 83
4.4.2 Eröffnungsverfahren H1_RELAX_IP ... 83
4.4.3 Verbesserungsverfahren H2_SWAP ... 91
4.4.4 Verbesserungsverfahren H3_SIMUL_ANNEAL ... 93
4.5 Implementierung ... 96
4.5.1 Eröffnungsverfahren H1_RELAX_IP ... 97
4.5.2 Verbesserungsverfahren H2_SWAP ... 99
4.5.3 Verbesserungsverfahren H3_SIMUL_ANNEAL ... 99
4.6 Evaluation ... 100
4.6.1 Einfluss der Workflowlänge ... 101
4.6.2 Einfluss der Prozessschrittkandidaten ... 104
4.6.3 Einfluss der Restriktionsstärke ... 106
4.6.4 Evaluation der Verbesserungsverfahren ... 108
4.7 Zusammenfassung ... 109

5 Ein Replanning-Mechanismus für Web Service Workflows ... 110
5.1 Problemstellung ... 110
5.2 Funktionsweise ... 112
5.3 Integration von Replanning in eine Workflow-Engine ... 114
5.3.1 Architektur ... 114
5.3.2 Modellierung von Replanning-Strategien ... 115
5.3.3 Ein ECA-Mechanismus zur Umsetzung der Replanning-Strategien ... 116

5.3.4 Implementierung ... 118
5.4 Evaluation ... 119
5.4.1 Versuchsaufbau ... 119
5.4.2 Einfluss der Workflowlänge ... 120
5.4.3 Einfluss der Prozessschrittkandidaten ... 123
5.4.4 Einfluss der Restriktionsstärke ... 126
5.5 Zusammenfassung ... 128

6 Verwandte Arbeiten ... 130
6.1 Dienstgüteunterstützung für Web Services ... 130
6.2 SOA und Web Services in anderen Forschungsgebieten ... 137

7 Zusammenfassung und Ausblick ... 139
7.1 Zusammenfassung ... 139
7.2 Ausblick ... 140

Literaturverzeichnis ... 142

Abkürzungsverzeichnis ... 153

Verwendete Bezeichner ... 156

Anhang ... 158

A WSQoSX ... 159
A.1 WSProxy ... 159
A.2 WSPortal ... 162
A.3 WSProxyAdmin ... 163

B WorkflowOptimizer ... 165
B.1 Implementierung der Simulationsumgebung ... 165
B.2 Definition der Testfälle ... 168
B.3 Erstellen des mathematischen Modells ... 172
B.4 Generierung der Testfälle ... 173
B.5 Benutzungsoberfläche ... 176
B.6 Statusinformationen ... 177

C Publikationen des Verfassers ... 179
C.1 Wissenschaftliche Veröffentlichungen ... 179
C.2 Patentanmeldungen ... 179

C.3 Mitautorenschaft bei Veröffentlichungen 179
C.4 Weitere Veröffentlichungen 180
C.5 Technical Reports 180

D Lebenslauf des Verfassers 181

E Eidesstattliche Erklärung laut §9 PromO 182

1 Einführung

1.1 Motivation

Globalisierte und deregulierte Märkte sowie anspruchsvoll gewordene Kunden haben zu einem hohen Kostendruck und harten Wettbewerb geführt. Um in diesem Wettbewerb als Unternehmen bestehen zu können, sind innovative und dynamisch adaptierbare Geschäftsprozesse ein wichtiger Erfolgsfaktor [7, 91]. Grundvoraussetzung für dynamisch adaptierbare Geschäftsprozesse ist eine flexible IT-Architektur, da Geschäftsprozesse eine Vielzahl von heterogenen Legacy-Anwendungen, Plattformen, Betriebssystemen und Kommunikationsmechanismen integrieren müssen [74]. Diese Heterogenität hat zu einer Inflexibilität und Komplexität geführt, die kaum noch beherrschbar scheint. So hat eine Forrester-Studie [77] ergeben, dass bei 3.500 befragten Unternehmen weniger als 35% der durchgeführten Integrationsprojekte zeitlich und finanziell im vorgegebenen Rahmen durchgeführt werden konnten.

Die Heterogenität der IT-Landschaften ist umso gravierender, wenn man bedenkt, dass im Zuge der Globalisierung und des Business Process Outsourcing (BPO) [20, 48, 78, 116] organisationsübergreifende, IT-unterstützte Geschäftsprozesse (sog. *Workflows*) eine immer größere Bedeutung erlangen [84]. Diese Entwicklung stellt insofern eine besondere Herausforderung für die beteiligten Unternehmen dar, weil zusätzlich zur Überwindung der internen Heterogenität im eigenen Unternehmen noch die Kopplung mit den unterschiedlichsten IT-Systemen der Geschäftspartner realisiert werden muss [79, 86].

In zunehmendem Maße gewinnen *Web Services* als Technologie zur Realisierung verteilter Workflows im Umfeld heterogener IT-Landschaften an Bedeutung, z. B. [4, 85, 111]. Web Services sind lose gekoppelte, wiederverwendbare Softwarekomponenten, die über das Internet aufgerufen werden können und durch den Austausch von Nachrichten miteinander kommunizieren, z. B. [3, 38]. Web Services unterschieden sich von anderen Integrationstechnologien dadurch, dass sie auf offenen XML-Standards wie *SOAP* (vormals: *Simple Object Access Protocol)* [150, 151, 152], *WSDL (Web Service Definition Language)* [148] und *UDDI (Universal Description, Discovery and Integration)* [9] beruhen und somit in verteilten, heterogenen Umgebungen eine Unabhängigkeit von den eingesetzten Plattformen, Betriebssystemen und Programmiersprachen ermöglichen.

Der Web Service Technologie liegt das Paradigma der *Service-orientierten Architektur (SOA)* zugrunde, das sich vor allem durch die lose Kopplung der beteiligten Services, die eine wohldefinierte fachliche Funktionalität anbieten, auszeichnet, z. B. [80, 106]. Im Gegensatz zu klassischen Softwarearchitekturen, die eine komplette Systemstruktur beschreiben, liegt der Fokus einer SOA auf der Beschreibung und Bereitstellung fachlicher Dienste und Funktionalitäten in Form von Services, vornehmlich zum Zwecke der Anwendungsintegration. Eine SOA beschreibt die Modularisierung einer meist heterogenen, komplexen Anwendungslandschaft in Form von Services [115]. Hierbei wird eine zusätzliche Schicht über der eigentlichen Softwareinfrastruktur eingeführt, die bestimmte Funktionalitäten der vorhandenen Software in Form von Services zur Verfügung stellt, sodass diese über ein Netzwerk aufgerufen werden

können. Die Komposition von Services zu Workflows führt zu einer prozessorientierten Ausrichtung der Anwendungslandschaft. Durch die Komposition von Services zu Workflows können flexible Produkte und Dienstleistungen nach dem Baukastenprinzip [105] erstellt werden. Aufgrund der losen Kopplung der beteiligten Services ist es möglich, auch Services externer Anbieter zu verwenden. Dies ermöglicht die Realisierung unternehmensübergreifender Workflows basierend auf Services.

Ein wesentlicher Faktor für den weiteren Erfolg des SOA-Paradigmas und der Web Service Technologie ist das Dienstgütemanagement Service-orientierter Workflows, z. B. [19, 92, 140, 141, 149]. So sind Unternehmen nicht bereit, Services ohne Garantien bezüglich deren Dienstgüteeigenschaften, angeboten von oftmals unbekannten Anbietern, in kritischen Workflows ihrer Produktivsysteme einzusetzen [149]. Dienstgüte im Umfeld von Web Services ist jedoch ein noch relativ wenig erforschtes Themengebiet. In diesem Zusammenhang bedarf es einer Dienstgütearchitektur, welche die dienstgütebasierte Auswahl von Web Services sowie die Steuerung und Ausführung Service-orientierter Workflows unter Berücksichtigung entsprechender Service Level Agreements (SLAs) übernimmt. Die zur Serviceauswahl benötigten Verfahren müssen weitgehend automatisiert sein und die Nutzerpräferenzen hinsichtlich der Dienstgüteeigenschaften berücksichtigen. Zudem sollten diese Verfahren hochperformant sein, um auch in Echtzeitanwendungen eingesetzt werden zu können. Eine weitere Anforderung an eine dienstgütebasierte Serviceauswahl besteht darin, bei der Zuordnung von Services zu noch nicht ausgeführten Prozessschritten des Workflows das tatsächliche Laufzeitverhalten der bereits ausgeführten Web Services zu berücksichtigen.

Diese Arbeit stellt daher die erweiterte Web Service Architektur *WSQoSX (Web Services Quality of Service Architectural Extension)* und deren prototypische Implementierung vor, die das Dienstgütemanagement der Services basierend auf einem aus neun Phasen bestehenden Vorgehensmodell übernimmt. Zudem werden als Weiterentwicklung von WSQoSX heuristische Lösungsverfahren zur Serviceauswahl entwickelt, implementiert und evaluiert. Diese stellen sicher, dass die Präferenzen des Nutzers hinsichtlich der nicht-funktionalen Web Service Eigenschaften berücksichtigt werden. Eine dieser Heuristiken wird ferner dazu verwendet, einen sog. *Replanning-Mechanismus* umzusetzen, der während der Ausführung des Workflows dafür sorgt, dass das tatsächliche Laufzeitverhalten bereits ausgeführter Web Services bei der Auswahl noch auszuführender Web Services mit einbezogen wird. So werden Nutzerpräferenzen und -anforderungen an die Dienstgüteeigenschaften auch dann sichergestellt, wenn das tatsächliche Laufzeitverhalten der Web Services vom ursprünglich geplanten abweicht.

1.2 Beitrag dieser Arbeit

Im Rahmen dieser Arbeit wird die Dienstgütearchitektur WSQoSX zur Steuerung Service-orientierter Workflows entwickelt. WSQoSX übernimmt das Management der Dienstgüte auf Grundlage eines im Zusammenhang mit dieser Arbeit entworfenen Vorgehensmodells. WSQoSX stellt sicher, dass nur Web Services zum Einsatz kommen, deren Dienstgüte den Präferenzen des Nutzers entsprechen. Externe Anbieter müssen ihre Web Services an einem

Portal anmelden und deren Dienstgüteeigenschaften in Form eines Service Level Agreements garantieren. Zudem bietet WSQoSX Monitoring-Mechanismen, um Informationen für eine verursachungsgerechte Kostenzuordnung zu erhalten und SLA-Verletzungen zu erkennen. Als Proof-of-Concept wird im Rahmen dieser Arbeit eine prototypische Implementierung auf Basis der Web Service Technologie vorgestellt.

Des Weiteren wird ein Verfahren entwickelt, dass die im Rahmen von WSQoSX beschriebene Komposition von Web Services zu Workflows als ein Optimierungsproblem modelliert. Dieses Optimierungsproblem berücksichtigt bei der Auswahl der Web Services die Präferenzen des Nutzers hinsichtlich der nicht-funktionalen Web Service Eigenschaften und die vom Nutzer definierten Restriktionen, bezogen auf die nicht-funktionalen Eigenschaften des gesamten Workflow. Da die Evaluation im Rahmen verwandter Arbeiten [22, 168, 172] gezeigt hat, dass dieses Optimierungsproblem NP-schwer ist und die Rechenzeit exakter Verfahren bei realistischen Problemgrößen für Echtzeitanwendungen nicht akzeptabel ist, werden daher in dieser Arbeit heuristische Lösungsverfahren zur Serviceauswahl entwickelt, implementiert und evaluiert. Zur Evaluierung dieser Heuristiken wurde eine Simulationsumgebung entwickelt. Die Simulation der in dieser Arbeit vorgestellten Heuristiken zeigt, dass die errechneten Lösungen bei sehr gutem Laufzeitverhalten nahe an das Optimum herankommen.

Das tatsächliche Laufzeitverhalten der ausgewählten Web Services weicht in der Realität von dem ursprünglich geplanten ab. Daher wird im Rahmen dieser Arbeit ein Replanning-Mechanismus basierend auf den vorgestellten Heuristiken entwickelt. Dieser Mechanismus berücksichtigt das tatsächliche Laufzeitverhalten bereits ausgeführter Web Services bei der Auswahl noch auszuführender Web Services, um sicherzustellen, dass die nutzerdefinierten Restriktionen und Präferenzen trotz abweichenden Laufzeitverhaltens gewahrt bleiben. Der Replanning-Mechanismus wird als Proof-of-Concept basierend auf den zuvor entwickelten Heuristiken implementiert und simuliert. Die Simulation zeigt u. a., dass durch die Anwendung von Replanning signifikante Kosteneinsparungen bei einem vertretbaren Rechenaufwand erzielt werden können.

Abbildung 1 zeigt, wie die in der vorliegenden Arbeit untersuchten Themen in eine SOA-Forschungsagenda [106] einzuordnen sind.

1.3 Aufbau der Arbeit

In Kapitel 2 werden die grundlegenden, zum Verständnis der Arbeit notwendigen Begriffe erläutertet. So wird das der Service-orientierten Architektur zugrunde liegende Konzept beschrieben und mit Web Services eine Technologie zur Umsetzung dieses Architekturparadigmas vorgestellt. Ein wesentlicher Vorteil einer SOA besteht darin, dass Services flexibel zu Service-orientierten Workflows zusammengesetzt werden können. Eine SOA ermöglicht somit die prozessorientierte Ausrichtung der Anwendungslandschaft. Ein Faktor für den weiteren Erfolg des SOA-Paradigmas und der Web Service Technologie ist das Management der Service-orientierten Workflows. Daher wird mit Dienstgüte der zentrale Ansatzpunkt für das technische Management Service-orientierter Workflows diskutiert.

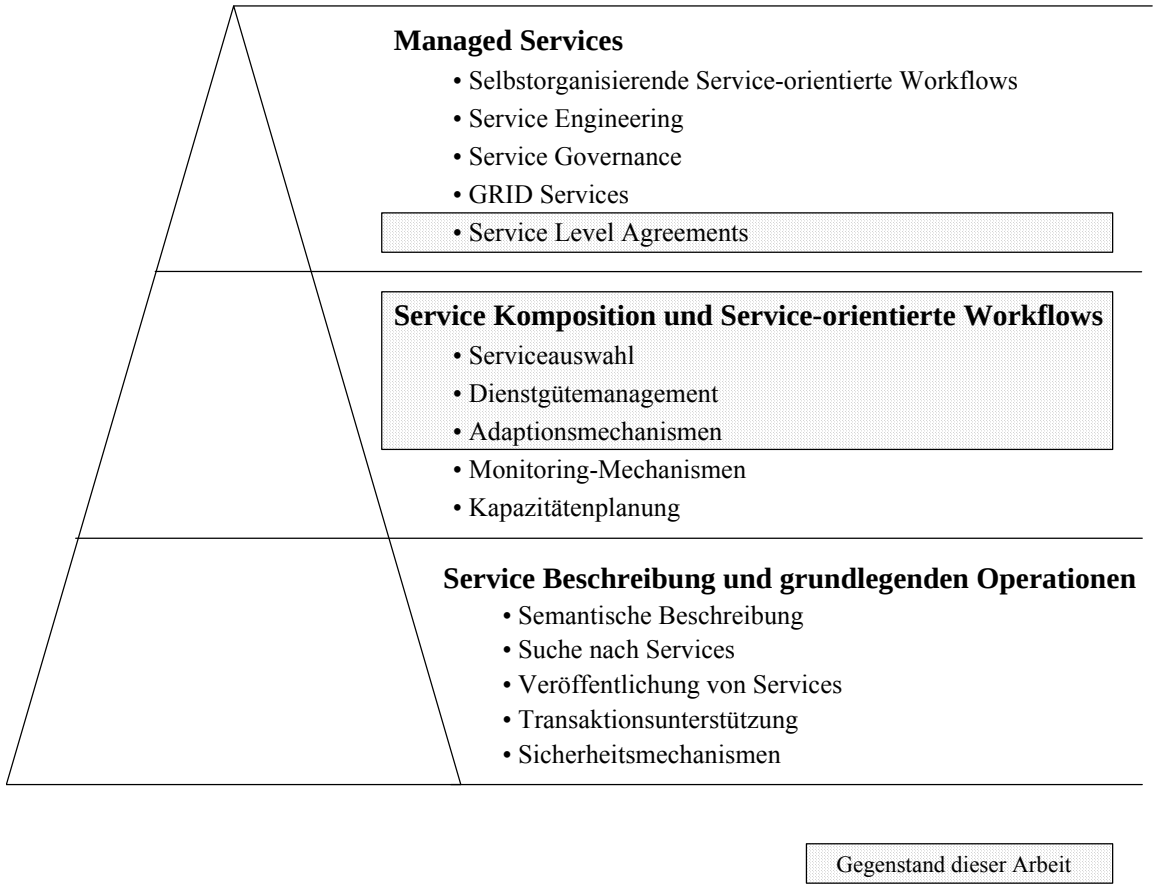


Abbildung 1: SOA-Forschungsagenda (in Anlehnung an [106])

Die im Rahmen dieser Arbeit entwickelte und prototypisch implementierte Web Service Architektur WSQoSX wird in Kapitel 3 erläutert. WSQoSX unterstützt das Dienstgütemanagement von Web Services basierend auf einem hierfür entwickelten, umfassenden Vorgehensmodell. So erfolgt die Auswahl konkreter Web Services dynamisch zur Laufzeit unter Berücksichtigung der zugesicherten Dienstgüteeigenschaften. WSQoSX überwacht zudem die Einhaltung der SLAs während der Laufzeit.

In Kapitel 4 wird, als konsequente Weiterentwicklung des WSQoSX zugrunde liegenden Ansatzes, ein Verfahren beschrieben, das bei der Auswahl funktional identischer Web Services sicherstellt, dass die Präferenzen des Nutzers hinsichtlich der nicht-funktionalen Web Service Eigenschaften, die sich auf den gesamten Workflow beziehen, berücksichtigt werden. Zudem gewährleistet dieses Verfahren, dass nutzerspezifische Restriktionen eingehalten werden. Die in diesem Kapitel entwickelten heuristischen Lösungsverfahren werden implementiert sowie deren Laufzeiterhalten und Lösungsgüte evaluiert.

Das tatsächliche Laufzeitverhalten der Web Services weicht zumeist von dem ursprünglich geplanten ab. Gründe hierfür sind u. a. die Unzuverlässigkeit externer Serviceanbieter und die Unzuverlässigkeit des Internets als Kommunikationskanal. Daher wird in Kapitel 5 ein Replanning-Mechanismus vorgestellt, der zur Ausführungszeit das tatsächliche Laufzeitverhalten bereits ausgeführter Web Services bei der Auswahl noch nicht aufgerufener Web Services berücksichtigt, um sicherzustellen, dass sowohl die Präferenzen des Nutzers als auch die

von ihm definierten Restriktionen berücksichtigt werden. Als Proof-of-Concept wird das entwickelte Replanning-Verfahren implementiert sowie dessen Laufzeitverhalten und die durch Replanning realisierten Kosteneinsparungen evaluiert.

Verwandte Arbeiten und Themenbereiche, die sich mit dem Dienstgütemanagement von Web Services und Service-orientierten Workflows auseinandersetzen, werden in Kapitel 6 diskutiert. Zudem werden Arbeiten vorgestellt, die den Zusammenhang des SOA-Paradigmas bzw. der Web Service Technologie mit den Themenbereichen *Grid Computing* und *Peer-to-Peer Computing* untersuchen.

Kapitel 7 fasst die wesentlichen Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf Erweiterungsmöglichkeiten und weitere Einsatzgebiete.

2 Grundlagen

In diesem Kapitel werden die zum Verständnis der vorliegenden Arbeit notwendigen Grundlagen und Begriffe erläutert. Ausgehend vom Paradigma der *Service-orientierten Architektur* (siehe Abschnitt 2.1) wird in Abschnitt 2.2 mit *Web Services* eine Technologie zur Umsetzung dieses Architekturparadigmas vorgestellt. In diesem Zusammenhang werden die im Web Service Umfeld relevanten Standards erläutert. Ein wesentlicher Vorteil des SOA-Paradigmas verglichen mit anderen Architekturkonzepten besteht darin, dass es die prozessorientierte Ausrichtung einer Anwendungslandschaft durch die Komposition von Services zu *Service-orientierten Workflow*s erlaubt. Dies wird in Abschnitt 2.3 erläutert. *Dienstgüte* als zentraler Ansatzpunkt für das technische Management Service-orientierter Workflows auf Basis der Web Service Technologie wird in Abschnitt 2.4 diskutiert. Hierbei werden unterschiedliche, für Web Service relevante Dienstgütekriterien vorgestellt. Zudem werden SLAs als ein Konzept beschrieben, wie solche Dienstgütekriterien vertraglich zwischen Servicenutzer und -anbieter vereinbart werden können.

2.1 Service-orientierte Architektur (SOA)

Anders als klassische Softwarearchitekturen, die eine komplette Systemstruktur beschreiben, beschränkt sich eine Service-orientierte Architektur auf die Bereitstellung fachlicher Dienste und Funktionalitäten in Form von Services, vornehmlich zum Zwecke der Anwendungsintegration [115]. Unter einem *Service*[1] wird hierbei eine abgeschlossene, unabhängige Komponente verstanden, die eine klar definierte Funktionalität über eine Schnittstelle anbietet, z. B. [106, 164]. Ein Service abstrahiert von zugrunde liegenden Entitäten, Objekten und Klassen. Eine SOA beschreibt die Modularisierung einer meist heterogenen, komplexen Anwendungslandschaft in Form von Services [54]. Hierbei wird eine zusätzliche Schicht über der eigentlichen Softwareinfrastruktur eingeführt, die bestimmte Funktionalitäten der vorhandenen Anwendungen in der Form von Services zur Verfügung stellt, sodass diese über ein Netzwerk aufgerufen werden können [106].

In diesem Abschnitt werden die drei für eine SOA charakteristischen Rollen *Servicenutzer*, *Serviceanbieter* und *Service Broker* vorgestellt (siehe Abschnitt 2.1.1). Verschiedene Servicearten, die im Rahmen einer SOA unterschieden werden, sind Gegenstand von Abschnitt 2.1.2. Merkmale einer SOA und Anforderungen an eine SOA werden in den Abschnitten 2.1.3 bzw. 2.1.4 beschrieben.

2.1.1 Rollen

Eine SOA basiert auf der Interaktion zwischen den drei verschiedenen Rollen Serviceanbieter, Servicenutzer und einem optionalen Service Broker (siehe Abbildung 2), der einen Verzeichnisdienst für Services bereitstellt, z. B. [37, 80, 106].

---

[1] In dieser Arbeit wird Service nicht mit dem deutschen Begriff *Dienst* übersetzt, da sich Service auch in der deutschen Literatur zu SOA durchgesetzt hat, z. B. [54, 60, 115].

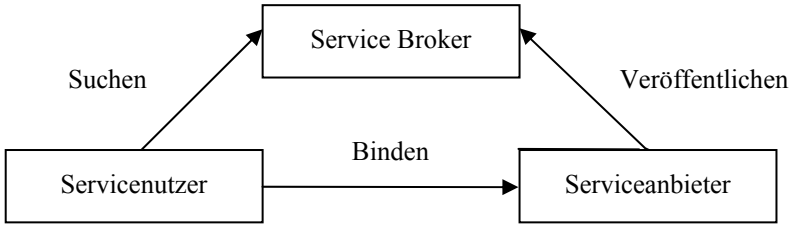


Abbildung 2: Rollen in einer SOA (in Anlehnung an [155])

Der Serviceanbieter implementiert einen Service, installiert ihn auf seiner Plattform und stellt ihn über ein Netzwerk (z. B. das Internet) zur Verfügung. Um einen Service potenziellen Nutzern leichter zugänglich zu machen, kann der Serviceanbieter einen Service bei einem oder mehreren Verzeichnisdiensten, die von einem Service Broker betrieben werden, mit der dazugehörigen Schnittstellenbeschreibung registrieren und veröffentlichen. Der Serviceanbieter ist für den ordnungsgemäßen Betrieb und die Verfügbarkeit des Services zuständig und auch verantwortlich.

Ist einem potenziellen Servicenutzer kein entsprechender Service bekannt, kann er im Verzeichnisdienst eines Service Brokers nach einem passenden Service suchen und diesen dynamisch zur Laufzeit in seine Anwendungen und Prozesse einbinden und aufrufen. Dies ist möglich, da die Schnittstellenbeschreibung in maschinenlesbarer Form vorliegt und vom Servicenutzer zur Laufzeit abgefragt werden kann.

Der Service Broker agiert als Vermittler zwischen Servicenutzer und Serviceanbieter. Er legt die vom Serviceanbieter publizierten Services in einem Verzeichnisdienst ab und bietet den Servicenutzer über eine Schnittstelle Suchmöglichkeiten für veröffentlichte Services an. Hierfür können Services nach verschiedenen Kriterien (z. B. Funktionalität oder Branchenzugehörigkeit) klassifiziert werden. Sind dem Servicenutzer alle für ihn relevanten Services bekannt, kann er die Services direkt aufrufen, ohne auf einen Service Broker als Vermittler zurückgreifen zu müssen.

2.1.2 Servicearten

Bezogen auf ihre Funktionalität können folgende Servicearten unterschieden werden [80]:

- *Infrastruktur Services* stellen technische Funktionalitäten bereit, die zum Betrieb einer SOA notwendig sind. Infrastruktur Services übernehmen die Aufgabe von Adaptern bei der Integration von Legacy-Anwendungen. Des Weiteren können Infrastruktur Services Sicherheitsmechanismen zur Verfügung stellen, die z. B. eine Authentifizierungs- oder Autorisierungsfunktionalität anbieten.
- *Basis Services* bilden das Fundament einer SOA. Basis Services sind für das Management der Datenquellen zuständig, d. h. Basis Services regeln den Zugriff auf Datenquellen und stellen benötigte Daten zur Verfügung. Zudem implementieren Basis Services die grundlegende fachliche Funktionalität für Business Services, wie z. B. die Kalkulation von Versicherungsprämien oder die Anordnung von Zahlungen an Geschädigte.

- *Business Services* stellen den Geschäftspartnern und Kunden eine umfassende fachliche Funktionalität zur Verfügung, wie z. B. die Abarbeitung eines Versicherungsfalls. So gesehen kapseln Business Services die Ablauflogik der entsprechenden Geschäftsprozesse. Business Services können ihrerseits wiederum Basis Services aufrufen, die sie zur Erbringung der Funktionalität benötigen.

Die Kommunikation und der Datenfluss zwischen den jeweiligen Services werden über den *Enterprise Service Bus (ESB)* als zentrale Komponente einer SOA realisiert, z. B. [19, 37]. Der ESB stellt den Backbone einer SOA dar, der das Routing der Nachrichten zwischen den Services übernimmt. Eine Hauptaufgabe des ESB ist die Konvertierung der verschiedenen Datentypen (z. B. XML-Schema- und Java-Datentypen) sowie die Transformation zwischen verschiedenen Kommunikationsparadigmen, wie z. B. synchrone und asynchrone Serviceaufrufe.

2.1.3 Merkmale

Folgende Merkmale sind für eine SOA charakteristisch und dienen zur Unterscheidung von anderen Architekturkonzepten:

- Die Services einer SOA sind *lose gekoppelt* [59, 106]. Kopplung ist hierbei ein Maß für die Abhängigkeiten der beteiligten Services. Das Prinzip der losen Kopplung im Rahmen des Software Engineering meint, dass der Abhängigkeitsgrad der beteiligten Services relativ gering ist [119]. Die lose Kopplung wird dadurch realisiert, dass Services durch den Austausch von Nachrichten miteinander kommunizieren. Durch eine lose Kopplung können Services bei Bedarf dynamisch gesucht, ausgeführt und eingebunden werden [37].
- Implementierungsdetails eines Services sind dem Prinzip des *Information Hiding* [107] folgend für den Servicenutzer unsichtbar gekapselt. Der Service wird über eine definierte Schnittstelle aufgerufen. Die Schnittstellenbeschreibung muss in maschinenlesbarer Form vorliegen, damit der Service auch dynamisch zur Laufzeit gefunden werden kann [37].
- Verglichen mit den aus der Objektorientierten Programmierung bekannten Objekten sind Services meist *grobgranular* [54, 59, 60], da sie eine umfassende, wohldefinierte fachliche Funktionalität anbieten. Services sind als autonome Einheiten *funktional unabhängig* von anderen Services. Die Granularität der Services ist aber auch abhängig von der Serviceart (vgl. Kap. 2.1.2). Business Services bieten eine umfassendere Funktionalität (z. B. Versicherungsschadensfall bearbeiten) an als eher feingranulare Basis Services (z. B. Kalkulation einer Versicherungsprämie).
- Aus dem Prinzip der funktionalen Unabhängigkeit einzelner Services resultiert auch die *zeitliche Unabhängigkeit*. Dies bedeutet, dass Services sowohl synchron als auch asynchron aufgerufen werden können [37].
- Durch *Servicekomposition* können neue (komplexe) Services aus vorhandenen (atomaren) Services konstruiert werden. Die Komposition von Services zu Workflows er-

möglicht eine prozessorientierte Ausrichtung der Anwendungslandschaft [19, 115, 121] und trägt so zu deren Flexibilisierung bei (vgl. Abschnitt 2.3.3).

- Durch den Einsatz von Verzeichnisdiensten seitens eines Service Brokers wird eine *Ortstransparenz* erreicht [106]. Diese gewährleistet, dass Services unabhängig von Ihrer physikalischen Adresse und Lokation gefunden und aufgerufen werden können.
- Services werden meist von anderen Services oder Applikationen aufgerufen. Ein Serviceaufruf durch menschliche Interaktion (z. B. via Browser) ist ebenfalls möglich.

2.1.4 Anforderungen

Anforderungen an den Einsatz einer SOA können nach technischen, betriebswirtschaftlichen und organisatorischen Gesichtspunkten klassifiziert werden:

Technische Gesichtspunkte

- Komplexitätsreduktion [41, 115]
  Dem SOA-Paradigma liegt als Entwurfsprinzip die Kapselung von Implementierungsdetails hinter wohldefinierten Schnittstellen zu Grunde. Dies ist ein bewährtes Mittel zur Komplexitätsreduktion. Daher ermöglicht das SOA-Paradigma auch eine verbesserte Beherrschbarkeit der unternehmensweiten Anwendungslandschaft.
- Erweiterbarkeit [41]
  Aufgrund der modularen Struktur einer SOA können neue, zusätzliche Services hinzugefügt werden. Die Funktionalität einer bestehenden Anwendungslandschaft kann somit leicht erweitert und an neue Anforderungen angepasst werden.
- Standardisierung und Wiederverwendung [41, 79, 115]
  Der Einsatz einer SOA fördert die Standardisierung der im Unternehmen zum Einsatz kommenden Services. Services mit standardisierten Funktionsbeschreibungen und Schnittstellendefinitionen ermöglichen eine einfache und kostengünstige Erstellung von Produkten und Geschäftsprozessen, da ein und derselbe Service ohne aufwendige Anpassungen in verschiedenen Kontexten wiederverwendet werden kann.
- Technologie- und Herstellerunabhängigkeit [59, 115]
  Das SOA-Paradigma ist ein abstraktes Architekturkonzept und somit technologieunabhängig. Daher ist es unerheblich, auf welcher Plattform ein Service läuft und in welcher Programmiersprache der Service implementiert ist. Diese Technologieunabhängigkeit zielt auch auf eine weitgehende Unabhängigkeit von den Herstellern ab. Auf diese Weise kann die Umsetzung einer SOA mit Technologien realisiert werden, die, wie z. B. Web Services, auf offenen (XML-basierten) Standards beruhen. Offene Standards erlauben die Verwendung von Services unterschiedlicher Hersteller. Dadurch wird das Risiko von Fehlinvestitionen und einer Abhängigkeit von einzelnen Softwareherstellern reduziert. Durch den Einsatz standardisierter Netzwerk- und Transportprotokolle sowie Nachrichtenformaten wird versucht, ein hohes Maß an Interoperabilität zu erreichen.

Betriebswirtschaftliche Gesichtspunkte

- Verkürzung der Time-to-Market [41, 80]
  Durch die Kombination von neuen und bereits bestehenden Services können sehr leicht neue Produkte bzw. Geschäftsprozesse erstellt werden. Diese Neukombination nach dem Baukastenprinzip wird durch die lose Kopplung der beteiligten Services ermöglicht. Dieses Vorgehen gewährleistet, dass die Time-to-Market erheblich verkürzt werden kann. Unternehmen sind somit in der Lage, dynamisch auf sich ändernde Rahmenbedingungen (z. B. Kundenwünsche und Gesetzesänderungen) zu reagieren.
- Hoher Investitionsschutz [41, 115]
  Eine bereits bestehende Anwendungslandschaft kann fachlich so aufgeteilt und in erneuerbare Bestandteile zerlegt werden, dass einzelne Module dieser Anwendungslandschaft sukzessive abgelöst werden können. Das SOA-Paradigma erlaubt die schrittweise Migration von monolithischen Softwaresystemen hin zu einer modular aufgebauten Anwendungslandschaft. Bereits bestehende Legacy-Anwendungen können weiter betrieben werden, indem die angebotenen Funktionalitäten in Services gekapselt werden. Daher bietet der Einsatz einer SOA einen hohen Investitionsschutz.
- Kosteneinsparungen [41, 80, 115]
  Bei der Entwicklung und dem Betrieb von IT-Systemen und Anwendungen können durch den Einsatz einer SOA Kosteneinsparungen realisiert werden. Dies wird u. a. durch die Wiederverwendung der Services in verschiedenen Kontexten realisiert. Produkte und Lösungen, die aus bereits bestehenden und getesteten Services modular zusammengebaut werden, können schneller und kostengünstiger als komplette Neuentwicklungen realisiert werden. Die Integration von externen Services kann ebenfalls zu Kosteneinsparungen führen, da spezialisierte Drittanbieter aufgrund von Skaleneffekten ihre Dienstleistungen häufig kostengünstiger anbieten können.
- Reduzierung des operationalen Risikos [80]
  Unter operationalen Risiken werden betriebliche Risiken verstanden, die in einem Unternehmen Schaden verursachen können, z. B. [125]. Durch die Wiederverwendung standardisierter Services sinkt das Risiko von Umsatzverlusten durch fehlerhafte Frontend-/ Backend-Prozesse. Zudem wird der Aufwand für umfangreiche Testroutinen deutlich reduziert.

Organisatorische Gesichtspunkte

- Outsourcing von Services [6, 106, 115]
  Aufgrund der losen Kopplung, der Technologieunabhängigkeit und der Standardisierung können Services auch von Drittanbietern bezogen werden. Dies ermöglicht Unternehmen, sich auf ihre Kernkompetenz zu konzentrieren und Services außerhalb dieser Kernkompetenz von spezialisierten Drittanbietern zu beziehen. Ein Business Process Outsourcing auf Serviceebene hat zudem den Vorteil, dass nicht zwangsläufig komplette Geschäftsprozesse ausgelagert werden müssen, sondern nur die Teilprozesse bzw. Prozessschritte, die ein Unternehmen tatsächlich nach außen geben möchte.

- Organisatorische Agilität und Flexibilität [80]
  Der Einsatz einer SOA kann zu größerer organisatorischer Flexibilität führen. Änderungen in der Organisationsstruktur und den Geschäftsprozessen können einfacher auf die zugrunde liegende IT-Architektur abgebildet werden. Dies wird durch die lose Kopplung der beteiligten Services und den modularen Charakter einer SOA erreicht. Letztendlich führt dies dazu, dass die Entscheidungen der Fachabteilungen schneller umgesetzt werden können. Strategische Entscheidungen werden daher wieder stärker von den Fachabteilungen getrieben. Die lose Kopplung der beteiligten Services bringt insofern eine zusätzliche Flexibilität, als dass Unternehmen schnell auf veränderte Rahmenbedingungen reagieren können.

2.2 Web Service Technologie

Zur konkreten Umsetzung des SOA-Paradigmas hat sich die Web Service Technologie in den vergangenen Jahren als De-facto-Standard etabliert. Dies ist vor allem darauf zurückzuführen, dass Web Services auf offenen XML-basierten Standards beruhen. Außerdem bieten führende Middleware-Hersteller Plattformen und Werkzeuge an, welche sowohl die Implementierung von Web Services als auch die Modellierung und Ausführung von Web Service Workflows unterstützen [115].

Nachdem in Abschnitt 2.2.1 der Begriff Web Service definiert wird, werden in den darauffolgenden Abschnitten die offenen XML-Standards SOAP (siehe Abschnitt 2.2.2), WSDL (siehe Abschnitt 2.2.3) und UDDI (siehe Abschnitt 2.2.4) vorgestellt, die den Kern der Web Service Technologie bilden. BPEL4WS (Business Process Execution Language for Web Services) als Kompositionssprache für Web Service Workflows ist Gegenstand von Abschnitt 2.2.5.

2.2.1 Definition

Das W3C definiert *Web Service* wie folgt [154]:

> *"A Web Service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web Service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols."*

Demnach sind Web Services wiederverwendbare, selbstbeschreibende, gekapselte Softwarekomponenten, deren Methoden plattformunabhängig über das Internet aufgerufen werden können und die lose durch den Austausch von Nachrichten miteinander gekoppelt sind. Zur Erreichung der Interoperabilität werden offene, XML-basierte Standards verwendet: WSDL, SOAP und UDDI. Mit WSDL [148] wird die Schnittstelle eines Web Service spezifiziert, mittels SOAP [150, 151, 152] werden Nachrichten von und an Web Services übermittelt und mithilfe von UDDI [9], einem Standard für Verzeichnisdienste für Web Services, können Web Services publiziert und aufgefunden werden.

Die Web Service Technologie erfüllt damit alle Anforderungen an den Aufbau einer SOA, wie sie im vorangegangenen Abschnitt beschrieben wurden. Da Web Services auf offenen XML-Standards basieren und zur Kommunikation die herkömmlichen Internetprotokolle verwenden, eignen sie sich in besonderer Weise zur Realisierung einer in hohem Maße interoperablen SOA. Aus diesem Grund spielen Web Services im Bereich des *Enterprise Application Integration (EAI)* [35, 42] und zunehmend auch bei der Orchestrierung von unternehmensübergreifenden Geschäftsprozessen eine wichtige Rolle [19, 121]. Eine Übersicht zu den Standards und Protokollen im Web Service Umfeld kann Abbildung 3 entnommen werden.

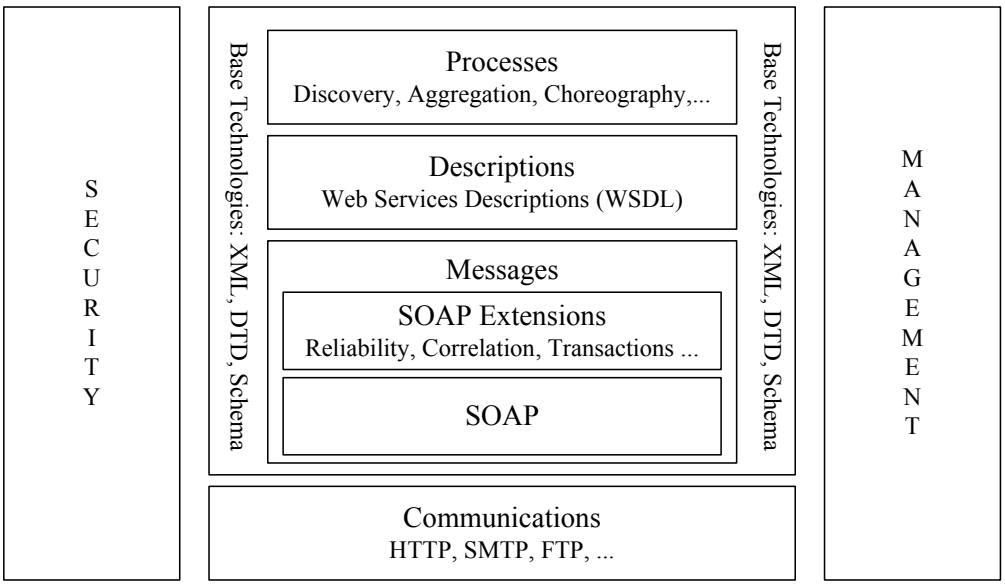


Abbildung 3: Web Service Stack [153]

Zum besseren Verständnis der Web Service Technologie werden die drei bereits erwähnten Basisstandards SOAP, WSDL und UDDI in den folgenden Abschnitten erläutert. Weitere in Abbildung 3 aufgeführte Web Service Protokolle und Standards, die nicht in unmittelbarem Zusammenhang mit dieser Arbeit stehen, werden z. B. in [3, 37] behandelt.

2.2.2 SOAP

SOAP [150, 151, 152] ist das Standard-Kommunikationsprotokoll der Web Service Technologie und spezifiziert den Aufbau von Nachrichten für die Kommunikation mit Web Services[2]. Bei SOAP-Nachrichten handelt es sich um XML-Dokumente, die zwischen Web Service Nutzer und Web Service Anbieter zum Aufruf von Methoden eines Web Service ausgetauscht werden, z. B. [3, 37, 38]. Die Verwendung von XML-Nachrichten als Mittel zur Kommunikation bietet den großen Vorteil der Interoperabilität bei der Nachrichtenverarbeitung, da XML sich als plattformübergreifender Dokument-Standard etabliert hat und die Verarbeitung von XML-Dokumenten unabhängig von bestimmten Anwendungen, Programmiersprachen, Technologien und Betriebssystemen erfolgen kann.

[2] Seit Version 1.2 wird SOAP nicht mehr als Akronym für *Simple Object Access Protocol* gebraucht [151].

Eine SOAP-Nachricht besteht aus den drei Hauptelementen SOAP-Envelope, SOAP-Header und SOAP-Body (siehe Abbildung 4).

Der SOAP-Envelope enthält alle weiteren Elemente einer SOAP-Nachricht. Der optionale SOAP-Header wird zur Übermittlung von zusätzlichen Informationen über die eigentliche Nachricht verwendet. Beispiele hierfür sind Daten, die im Rahmen von Authentifizierungen, des Transaktionsmanagements oder der Kontextverwaltung benötigt werden. Mittels eines speziellen Attributs `mustUnderstand` kann der Sender in einem Element des SOAP-Header angeben, ob es zwingend für die Verarbeitung der SOAP-Nachricht nötig ist, dass der Empfänger dieses Element auch verstehen kann. Ist dies der Fall und der Empfänger kann dieses Element nicht verstehen bzw. verarbeiten, so muss er in seiner Antwort mittels eines `Fault`-Elements darauf hinweisen. Da es möglich ist, dass eine SOAP-Nachricht nicht direkt zum Empfänger gesendet wird, sondern auf seinem Weg dorthin andere zwischengeschaltete Systeme passiert, kann mittels des `actor`-Attributs zu jedem Header-Element spezifiziert werden, durch welches System es verarbeitet werden soll. Nachdem ein solches Zwischensystem die entsprechenden Header-Elemente verarbeitet hat, entfernt es diese, bevor es die SOAP-Nachricht an die nachgelagerten Systeme weitergibt.

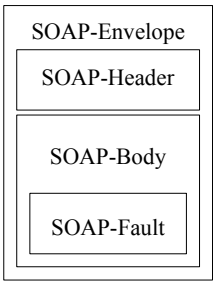


Abbildung 4: Aufbau einer SOAP-Nachricht [155]

Im SOAP-Body sind die eigentlichen Nutzdaten für den Empfänger untergebracht. Im Falle des Aufrufs von Methoden eines Web Service sind dies abstrakte Beschreibungen der aufzurufenden Methode und ggf. serialisierte Parameter- bzw. Ergebniswerte. Gelingt es dem Empfänger nicht, die im Body gelieferten Informationen zu verarbeiten oder tritt während der Verarbeitung der Informationen ein Fehler auf, so informiert der Empfänger den Sender in seiner Antwort mittels eines `Fault`-Elements über das Auftreten des Fehlers. Hierbei wird zudem noch eine Beschreibung des Fehlers mitgeliefert.

SOAP unterstützt zwei Interaktionsmuster, z. B. [3, 38]:

- *Document-Style:* Bei diesem Interaktionsmuster einigen sich die beiden Kommunikationspartner über die Struktur und Reihenfolge der zwischen ihnen auszutauschenden Nachrichten. Die eigentlichen Anwendungsdaten (z. B. Kaufbestellung) werden dann im SOAP-Body als XML-Dokument ausgetauscht.
- *RPC-Style:* Hier enthält der SOAP-Body lediglich den Namen und die Eingabeparameter der aufzurufenden Funktion. Dieses Interaktionsmuster ist im Web Service Umfeld besonders häufig anzutreffen, da es einen plattformunabhängigen entfernten Me-

thodenaufruf über ein Netzwerk gemäß dem Client-Server-Paradigma ermöglicht. Hierzu sendet der Client eine Nachricht mit Informationen über die aufzurufende Methode und die nötigen Parameterwerte an den Server, welcher die Methode ausführt und die Ergebniswerte wiederum in eine Nachricht verpackt und an den Client zurücksendet. Bei einem SOAP-RPC wird dieser Mechanismus dazu verwendet, eine Methode eines Web Service aufzurufen. Der Web Service Nutzer übernimmt hierbei die Rolle des Client und der Web Service Anbieter die des Servers (siehe Abbildung 5). Die aufzurufende Methode und die zugehörigen Parameterwerte werden in einem SOAP-Request und die Ergebniswerte in einem SOAP-Response als XML-Dokumente gemäß dem SOAP-Standard codiert. Das Transportprotokoll, mit welchem die Nachrichten zwischen den Kommunikationspartnern transportiert werden, ist hierbei durch SOAP nicht fest vorgegeben. Meist wird HTTP als Transportprotokoll verwendet, jedoch können hierzu auch Protokolle wie SMTP oder FTP verwendet werden.

Sollen mit einer SOAP-Nachricht auch Daten übermittelt werden, so müssen diese, bevor sie in die XML-Nachricht integriert werden können, vom Sender in ein XML-Format serialisiert werden. Damit der Nachrichtenempfänger erkennen kann, wie er die Daten aus dem XML-Format wieder deserialisieren und in ein für ihn verarbeitbares Format konvertiert kann, gibt der Sender den von ihm zur Codierung der Daten verwendeten Standard mittels des sog. Encodings an. Das Encoding wird üblicherweise im SOAP-Envelope deklariert und gilt für die gesamte SOAP-Nachricht.

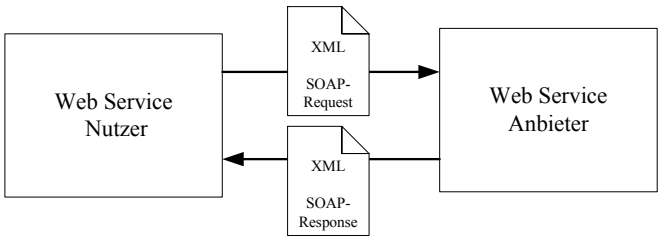


Abbildung 5: Nachrichtenaustausch bei einem SOAP-RPC

2.2.3 Web Service Description Language (WSDL)

WSDL [148] stellt ein XML-Format zur syntaktischen Beschreibung von Web Services dar und übernimmt somit die Rolle einer Schnittstellenbeschreibungssprache (engl. Interface Definition Language, Abk. IDL). Mit WSDL können Web Services unabhängig von der tatsächlich verwendeten Programmiersprache beschrieben werden. WSDL spezifiziert, wie ein Web Service aufgerufen werden kann und welche Operationen er anbietet. Zudem werden die Protokolle beschrieben, an die der Web Service gebunden wird. Die WSDL-Beschreibung eines Web Service ist in zwei Teile gegliedert (siehe Abbildung 6), z. B. [3, 37, 38]:

- Das *Service Interface Definition* beschreibt abstrakt die Schnittstellen des Web Service. So werden die durch den Web Service angebotenen Operationen spezifiziert. Jede dieser Operationen kann jeweils mit einer Input- und Output-Nachricht versehen werden. Weiterhin werden auch komplexe Datentypen sowie die Formate der Nachrichten definiert.

- Das *Service Implementation Definition* beschreibt, wo sich die konkrete Implementierung des Web Service befindet. Hierzu wird spezifiziert, über welche URI und welche Protokolle der Web Service erreichbar ist und wie die Daten für einen Aufruf serialisiert und codiert werden müssen.

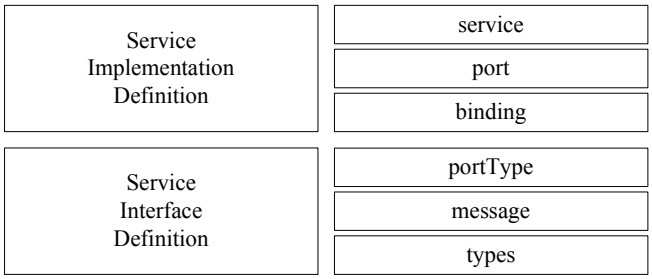


Abbildung 6: Aufbau eines WSDL-Dokumentes (in Anlehnung an [81])

Die Unterscheidung zwischen Service Interface Definition und Service Implementation ermöglicht es, dass mehrere Services dieselbe Schnittstelle implementieren können.

Abbildung 7 veranschaulicht die im Folgenden beschriebene Struktur einer WSDL-Datei.

```
<definitions>
	<types>
		<schema></schema>
	</types>
	<message>
		<part></part>
	</message>
	<portType>
		<operation>
			<input></input>
			<output></output>
		</operation>
	</portType>
	<binding>
		<operation>
			<input></input>
			<output></output>
		</operation>
	</binding>
	<service>
		<port></port>
	</service>
</definitions>
```

Abbildung 7: Struktur eines WSDL-Dokumentes [155]

Die Elemente `portType`, `message` und `types` werden im abstrakten Teil und die Elemente `binding`, `port` und `service` im konkreten Teil einer WSDL-Schnittstellenbeschreibung definiert. Die Bedeutung der einzelnen Elemente wird im Folgenden erläutert, z. B. [3, 38, 155]:

- Das Element `definitions` dient als Container für die Servicebeschreibung und zur Definition der beteiligten XML-Namespaces.
- Das optionale `types`-Element definiert Datentypen, die in Nachrichten verwendet werden. Da Datentypen in Nachrichten mittels XML-Schema beschrieben werden, müssen hier nur solche Datentypen festgelegt werden, die später in Nachrichten verwendet werden sollen, aber nicht bereits als Basistypen in XML-Schema zur Verfügung stehen.
- Mit `message`-Elementen werden Nachrichten beschrieben, die zwischen Web Service Nutzer und Web Service Anbieter beim Aufruf einer Operation eines Web Service ausgetauscht werden. Die Beschreibung der Methodenparameter bzw. Rückgabewerte erfolgt inklusive ihrer Datentypen in `part`-Elementen. Nachrichten, die einen Funktionsaufruf darstellen, definieren mit ihren `part`-Elementen die benötigten Methodenparameter. Nachrichten, die eine Antwort auf einen Funktionsaufruf enthalten, definieren mit ihren `part`-Elementen die von ihnen gelieferten Rückgabewerte.
- Im `portType`-Element werden mittels `operation`-Elemente alle vom Web Service nach außen angebotenen Operationen beschrieben. Für jede Operation können maximal zwei Nachrichten, die zuvor mittels eines `message`-Elements definiert wurden, angegeben werden. Je nachdem, ob diese Nachrichten in einem `input`- oder `output`-Element deklariert werden, können verschiedene Aufrufmechanismen für die Operation beschrieben werden. In diesem Zusammenhang werden vier verschiedene Aufrufmechanismen unterschieden:
  - *One-way:* Der Client ruft den Web Service durch das Senden einer Nachricht auf.
  - *Request-response:* Der Web Service wird durch eine Nachricht des Clients aufgerufen und antwortet durch das Senden einer Antwortnachricht.
  - *Solicit-response:* Der Web Service sendet eine Nachricht und wartet bis eine Antwort eintrifft.
  - *Notification:* Der Web Service als solcher sendet eine Nachricht an den Client.

  Synchrone Nachrichten können auf Grundlage von request-response und solicit-response-Interaktionen modelliert werden, während one-way- und notification-Interaktionsmuster zur Modellierung asynchroner Aufrufe herangezogen werden.
- Zu jeder Operation, die im `portType`-Element deklariert wurde, wird mittels eines `binding`-Element definiert, in welchem Nachrichtenformat die Nachrichten übermittelt und mit welcher Codierung die Daten ausgetauscht werden. Zudem wird festgelegt, mit welchem Transportprotokoll die Nachrichten gesendet werden. Diese Angaben werden als Bindung einer Operation bezeichnet. Zu einer Operation lassen sich hierbei mehrere Bindungen angeben, z. B. an SOAP, HTTP und MIME, die bereits explizit in der WSDL-Spezifikation definiert sind. WSDL ist jedoch erweiterbar, sodass hier auch andere Bindungen denkbar sind.

- Für jede so definierte Bindung wird in einem `service`-Element festgelegt, an welche konkrete URL ein Web Service Nutzer seine gemäß der Bindung erstellten Nachrichten zum Aufruf des Web Service senden kann. Hierbei wird die URL-Adresse durch ein `port`-Element definiert. Durch die Angabe von mehreren `port`-Elementen können verschiedene URL-Adressen definiert werden, unter denen der Web Service zu erreichen ist.

2.2.4 Universal Description, Discovery and Integration (UDDI)

UDDI [9] bezeichnet den von Microsoft, IBM und Ariba ins Leben gerufenen De-facto-Standard für Web Service Verzeichnisse. Bei UDDI handelt es sich um einen XML-basierten Verzeichnisdienst, der es Unternehmen erlaubt, nach Web Services externer Anbieter zu suchen und eigene Web Services zu veröffentlichen. Die Suche der im UDDI-Verzeichnis registrierten Web Services erfolgt über Metadaten, welche die Funktionalität und Kategorie des Web Service näher beschreiben. Grundsätzlich wird zwischen öffentlichen und privaten UDDI-Verzeichnisdiensten unterschieden. So ist der Zugriff auf private UDDI-Verzeichnisse auf bestimmte Kunden und Geschäftspartner begrenzt, während öffentliche UDDI-Verzeichnisdienste frei zugänglich sind.

Bestandteile der UDDI-Spezifikation sind die Beschreibung der verwendeten Datenstrukturen und die APIs (Application Programming Interface) zur Suche und zum Publizieren von Web Services. Das UDDI-Verzeichnis ist hierbei selbst eine Web Service Anwendung und stellt seine Funktionalität als Web Service über das Kommunikationsprotokoll SOAP im Netzwerk zur Verfügung. Alle APIs und Datenstrukturen sind hierbei in XML bzw. im XML-Schema-Format spezifiziert.

Ein UDDI-Verzeichnis bietet die Möglichkeit, vielfältige Informationen aus verschiedenen Bereichen zu den Web Services abzulegen und abzufragen. Diese Bereiche lassen sich wie folgt charakterisieren, z. B. [3, 38]:

- *White Pages*: Name, Beschreibung und sämtliche Kontaktinformationen der Unternehmen, die Web Services im Verzeichnis publiziert haben.
- *Yellow Pages*: Kategorisierungsinformationen der Unternehmen und der durch sie angebotenen Web Services. Dadurch kann das UDDI-Verzeichnis wie ein Branchenbuch oder die Gelben Seiten benutzt werden.
- *Green Pages*: Technische Dokumentation zur Beschreibung der veröffentlichten Web Services. Hier können z. B. WSDL-Dokumente referenziert werden, welche die Schnittstelle des Web Service beschreiben.

Das hierbei zur Ablage der Informationen verwendete Datenmodell besteht im Wesentlichen aus fünf Datenstrukturen [3, 37, 155], die in Abbildung 8 dargestellt sind und im Folgenden beschrieben werden.

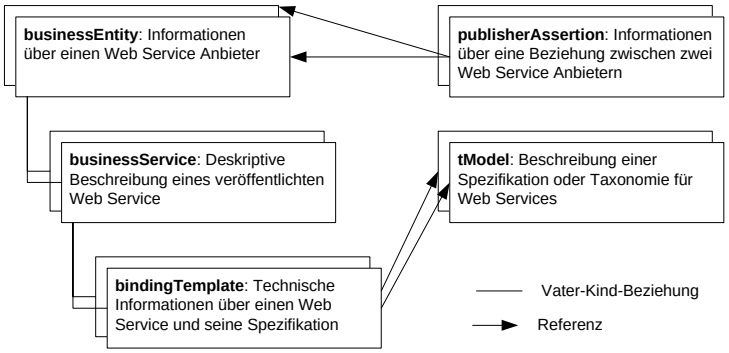


Abbildung 8: UDDI-Datenstruktur (in Anlehnung an [112, 155])

Eine `businessEntity` repräsentiert ein Unternehmen bzw. eine Organisation, die Web Services im UDDI-Verzeichnis veröffentlicht hat. Stehen zwei Organisationen hierbei in Beziehung zueinander, weil sie z. B. gemeinsam an einem elektronischen Marktplatz teilnehmen, so kann diese Beziehung mittels einer `publisherAssertion` modelliert werden. Diese beschreibt die Art der Beziehung und referenziert die beteiligten Organisationen.

Die von einer Organisation veröffentlichten Web Services werden durch `businessServices` repräsentiert und der jeweiligen `businessEntity` in einer Vater-Kind-Beziehung zugeordnet. Da ein Web Service auf verschiedene Weisen, z. B. über unterschiedliche Protokolle, aufgerufen werden kann, können einem `businessService` wiederum mehrere `bindingTemplates` zugeordnet werden, welche jeweils die technischen Details einer Zugangs- bzw. Aufrufmöglichkeit des Web Service beschreiben. Unter dem `accessPoint`-Attribut wird die URL-Adresse für den Zugriff auf den Web Service angegeben und unter der Datenstruktur `tModelInstanceDetails` eine technische Beschreibung des Web Service abgelegt. In der technischen Beschreibung wird u. a. auf Servicetypbeschreibungen (sog. `tModels`) referenziert.

Das `tModel` ist das zentrale Datenelement von UDDI. Es kann als Datentyp gleichartiger Web Services gedacht werden, z. B. [3, 38]. Die `tModel`-Datenstruktur wird von anderen Elementen, wie z. B. `bindingTemplate` oder `businessService`, referenziert. Es besteht jedoch keine Eltern-Kind Beziehung, wie z. B. zwischen `businessEntity` und `businessService`. Ein `tModel` beschreibt die technische Spezifikation eines Web Service. Hierzu wird in der UDDI-Registry ein WSDL-Dokument als Servicespezifikation abgelegt. Dieses WSDL-Dokument beschreibt die abstrakte Serviceschnittstelle mithilfe der Elemente `types`, `message` und `portType`. Wenn ein konkreter Web Service diese abstrakte Servicespezifikation implementiert, wird dies dadurch zum Ausdruck gebracht, dass er das entsprechende `tModel` referenziert. Hierzu muss das `tModelInstanceDetail`-Element des `bindingTemplate` auf das entsprechende `tModel` verweisen. Ein `tModel` verknüpft gewissermaßen die technische Beschreibung eines Web Service (`bindingTemplate`) mit dessen abstrakter Schnittstellenbeschreibung. Diese Verknüpfung wird dadurch erreicht, indem das `tModel` auf den abstrakten Teil der Schnittstellenbeschreibung verweist und z. B. ein `bindingTemplate` eine Referenz auf das `tModel` enthält. Dadurch ermöglicht das `tModel`-Element, zu einer

abstrakten Web Service Schnittstellenbeschreibung alle `bindingTemplates` zu finden und umgekehrt, zu einem gegebenen `bindingTemplate` den abstrakten Teil der Schnittstellenbeschreibung zu bestimmen.

Der Zugriff auf UDDI-Verzeichnisse erfolgt auf zweierlei Weise. Zum einen kann via Web Browser auf das UDDI-Verzeichnis zugegriffen werden. Hierbei steht die Mensch-Maschine-Kommunikation im Mittelpunkt. Zum anderen kann das UDDI-Verzeichnis aber auch über eine Web Service Schnittstelle genutzt werden. Dies ist dann von Bedeutung, wenn Web Services von anderen Web Services dynamisch zur Laufzeit gefunden werden sollen.

Damit Servicenutzer das UDDI-Verzeichnis auf standardisierte Weise durchsuchen und Serviceanbieter ihre Web Services auf standardisierte Weise in das UDDI-Verzeichnis eintragen können, beinhaltet der UDDI-Standard auch die Spezifikation eines entsprechenden API. Das API ist hierbei in Form von XML-Segmenten spezifiziert, die zum Aufruf einer Operation in eine SOAP-Nachricht verpackt an das UDDI-Verzeichnis gesendet werden müssen. Das gesamte API setzt sich hierbei aus zwei einzelnen APIs zusammen, der *Inquiry-API* und der *Publisher-API*.

Die wichtigsten Operationen, die durch das Inquiry-API spezifiziert werden, sind Anfragen zum Auffinden von Informationen im UDDI-Verzeichnis. Dabei dienen `find`-Anfragen zur Suche nach bestimmten UDDI-Datenstrukturen wie `businessEntity`, `businessService`, `bindingTemplate` und `tModel`. Nach dem Auffinden von Datenstrukturen können `get`-Anfragen zum Auslesen von Details aus diesen Datenstrukturen verwendet werden.

Das Publisher-API spezifiziert Anfragen zum Anlegen, Aktualisieren und Löschen von Datenstrukturen, z. B. [3, 37, 155]. Für das Anlegen und Aktualisieren von Datenstrukturen werden `save`-Anfragen und für das Löschen von Datenstrukturen `delete`-Anfragen verwendet. Um das UDDI-Verzeichnis vor unautorisierten Veränderungen zu schützen, ist bei der Benutzung der Publisher-API eine Authentifizierung nötig. Bevor Operationen der Publisher-API verwendet werden können, muss unter Angabe der Benutzeridentität ein Sicherheitstoken (`authToken`) beim UDDI-Verzeichnis angefordert werden. Das Sicherheitstoken muss beim Aufruf von Operationen der Publisher-API in die erzeugten SOAP-Nachrichten eingefügt werden und wird durch das UDDI-Verzeichnis zur Prüfung der Zugriffsrechte bei der Ausführung von Operationen verwendet.

Für den Zugriff auf UDDI-Verzeichnisse existieren verschiedene plattformspezifische APIs, welche die Kommunikation mit einem UDDI-Verzeichnis kapseln, sodass sich Entwickler nicht direkt mit XML- bzw. SOAP-Dokumenten und deren Übermittlung und Empfang auseinandersetzen müssen. JAXR (Java API for XML Registries) stellt ein solches API für die Java-Plattform dar.

2.2.5 Business Process Execution Language for Web Services (BPEL4WS)

Zur Beschreibung von aus Web Services orchestrierten Workflows wurden im Jahre 2001 die zwei konkurrierenden Spezifikationen *XLANG* [138] von Microsoft und die *Web Services*

Flow Language (WSFL) [83] von IBM entwickelt. Um einen einheitlichen Standard zu schaffen, arbeiteten Microsoft und IBM mit BEA, SAP und Siebel Systems zusammen und entwickelten aus den Ideen beider vormals konkurrierenden Spezifikationen einen neuen gemeinsamen Standard namens Business Process Execution Language for Web Services. Die Spezifikation von BPEL4WS erschien im Juli 2002 in seiner ersten Version 1.0 und im Mai 2003 in der Version 1.1 [4]. Die Version 1.1 der Spezifikation wurde zur Standardisierung und Weiterentwicklung an *OASIS (Organization for the Advancement of Structured Information Standards)* übergeben. Seitdem entwickelte sich BPEL4WS zu einem De-facto-Standard zur Modellierung von Workflows auf Web Service Basis und wird durch Produkte führender Hersteller unterstützt. Der Vorteil der Verwendung eines Standards für die Workflowmodellierung liegt in der Interoperabilität der erstellten Workflows. So kann BPEL4WS nicht nur als Ausführungssprache, sondern auch als Austauschformat für die Definition von Workflows auf Basis der Web Service Technologie betrachtet werden. OASIS benannte im September 2004 den Nachfolger von BPEL4WS 1.1 als WS-BPEL 2.0, d. h. BPEL4WS wurde in *Web Services Business Process Execution Language (WS-BPEL)* umbenannt. Im Rahmen dieser Arbeit wird die Bezeichnung BPEL4WS unabhängig von der jeweiligen Version verwendet.

Bei BPEL4WS handelt es sich um die Spezifikation einer auf XML basierenden Sprache zur formalen Notation von Workflows, z. B. [69]. Ein so definierter Workflow kann mittels einer BPEL4WS-fähigen Workflow-Engine ausgeführt werden. Der Workflow wird durch die Workflow-Engine wiederum als Web Service zur Verfügung gestellt, wodurch Workflows im Rahmen von Aktivitäten anderer Workflows als Web Service eingebunden werden können. Die Schnittstellen des Workflows und der an ihm beteiligten Web Services werden in Form von WSDL-Dokumenten beschrieben.

Konzeptionell betrachtet, entstehen mit BPEL4WS modellierte Workflows durch die Komposition verschiedener Web Services. Der so modellierte Workflow kann seinerseits als Web Service von anderen Web Services aufgerufen werden. Hierbei kann es sich sowohl um unternehmensinterne als auch um Unternehmensgrenzen überschreitende, verteilte Workflows handeln.

Workflows können entweder als ausführbare oder als abstrakte Prozesse definiert werden. Ausführbare Prozesse modellieren das Verhalten der an der Geschäftstransaktion teilnehmenden Partner. Abstrakte Workflows spezifizieren den wechselseitigen Nachrichtenaustausch zwischen den Teilnehmern der Geschäftsbeziehung sowie das Format und die Attribute der auszutauschenden Nachrichten. Die interne Implementierung wird hierbei jedoch nicht veröffentlicht. So ist der abstrakten Beschreibung eines Kreditprozesses zwar zu entnehmen, dass ein Kreditantrag angenommen bzw. abgelehnt werden kann, die hierfür ausschlaggebenden Entscheidungskriterien bleiben jedoch verborgen. Abstrakte Prozesse sind nicht ausführbar. Die Unterscheidung zwischen abstrakten und ausführbaren Prozessen berücksichtigt, dass Unternehmen im Allgemeinen kein Interesse daran haben, Geschäftspartnern Einblick in ihre internen Workflows zu geben.

Durch BPEL4WS wird eine Reihe von Problembereichen, die sich in Bezug auf die Definition und Ausführung von Workflows ergeben, adressiert. Dies sind u. a.:

- Beschreibung des Kontrollflusses, z. B. mittels Konstrukten zur seriellen und parallelen Ausführung, Verzweigungen, Schleifen sowie der Synchronisation paralleler Aktivitäten
- Beschreibung des Datenflusses, z. B. welche Daten aus welchen Nachrichten an welchen Web Service gesendet werden
- Zuordnung von Nachrichten zu einem gemeinsamen Kontext, sowohl bei synchroner als auch langläufiger asynchroner Kommunikation
- Verwaltung von Zustandsinformationen zu jedem Kontext
- Fehlerbehandlungsmechanismen
- Kompensationsmechanismen für den Fall des Abbruchs oder des Scheiterns einer bestimmten Menge von Operationen

Die Struktur eines mit BPEL4WS modellierten Workflows wird in Abbildung 9 schematisch dargestellt. Einige hierbei verwendete Sprachkonstrukte zur Modellierung des Daten- und Kontrollflusses werden im Folgenden diskutiert. Für eine weitergehende Betrachtung der BPEL4WS-Sprachkonstrukte wird z. B. auf [69, 156] verwiesen.

```
<process>
// Definition der Beziehungen zu externen Partnern (Web Services)
<partnerLinks>...</partnerLinks>
// Definition der verwendeten Daten- und Nachrichtencontainer
<variables>...</variables>
// Definition von CorrelationSets zur Identifikation einer Prozessinstanz
<correlationSets>...</correlationSets>
// Fehlerbehandlung und Kompensation
<faultHandlers>...</faultHandlers>
<compensationHandlers>...</compensationHandlers>
// Definition des eigentlichen Workflows (Aktivitäten)
activity
</process>
```

Abbildung 9: Struktur eines BPEL4WS-Prozesses

- Integration externer Web Services
  Das Element `partnerLinkType` beschreibt die Abhängigkeiten zwischen dem Workflow und den beteiligten Partnern (siehe Abbildung 10). Jeder `partnerLinkType` besteht aus den Definitionen von ein oder zwei `roles`. Dies ist abhängig davon, ob nur eine oder beide involvierte Parteien Operationen zur Verfügung stellen. Zu jeder Rolle wird ein WSDL-`portType` angegeben, durch den die von der Rolle zu unterstützenden Operationen eindeutig definiert werden.

```
<partnerLinkType name="LoanService">
<role name="LoanServiceRequester">
      <portType name="LoanServiceCallback"/>
</role>
<role name="LoanServiceProvider">
      <portType name="LoanService"/>
</role>
</partnerLinkType>
```

Abbildung 10: Verwendung des Konstruktes partnerLinkType

`partnerLinks` beschreiben die Verbindungen zwischen den verschiedenen Partnern, die innerhalb des Workflows interagieren (siehe Abbildung 11). Jeder `PartnerLink` wird durch einen zugehörigen `partnerLinkType` sowie durch die im verwendeten `partnerLinkType` definierten Rollen charakterisiert. Der `partnerLinkType` beschreibt die benötigte Funktionalität der involvierten Partner. Durch die Angabe von `myRole` wird die Rolle des Prozessinhabers definiert. `PartnerRole` gibt dabei die Rolle des externen Web Service an.

```
<partnerLinks>
      <partnerLink name="client"
            partnerLinkType="LoanFlow"
            partnerRole="LoanFlowRequester"
            myRole="LoanFlowProvider"/>
</partnerLinks>
```

Abbildung 11: Verwendung des Konstruktes partnerLink

Im Abschnitt `Variables` werden die für den Prozess benötigten Variablen definiert (siehe Abbildung 12). In diesen werden empfangene und zu sendende Nachrichten gespeichert bzw. Informationen, die den aktuellen Status des Prozesses beschreiben.

```
<variable name="loanApplication"
      messageType="initiateLoanServiceMessage"/>
```

Abbildung 12: Definition von Variablen

- Steuerung des Kontrollflusses
  Nach der Beschreibung der `partnerLinks` und Variablen folgt die Modellierung des eigentlichen Prozesses, der aus einer Aneinanderreihung und Schachtelung von Aktivitäten besteht (siehe Abbildung 13). Die einzelnen Befehle bzw. Blöcke werden sequenziell abgearbeitet, was durch den Einschluss in das `sequence`-Element zum Ausdruck kommt. Der Befehl `receive` bewirkt, dass der Prozess auf einen Aufruf durch den `partnerLink` namens Client wartet. Der auf `yes` gesetzte Parameter *`createInstance`* gibt an, dass mit dem Empfangen einer Anfrage eine neue Instanz des Prozesses erzeugt wird.

```
<sequence>
   <receive
      name="receiveInput"
      partnerLink="client"
      portType="LoanFlow"
      operation="initiate"
      variable="input"
      createInstance="yes"/>
   <invoke name="replyOutput"
      partnerLink="client"
      portType="LoanFlowCallback"
      operation="onResult"
      inputVariable="selectLoanOffer"/>
</sequence>
```

Abbildung 13: Aufruf eines Web Service

Mit `invoke` wird das in der Variable `selectedLoanOffer` abgelegte Kreditangebot an den anfragenden Web Service zurückgesendet. Mit `reply` kann auf eine eingegangene Nachricht geantwortet werden.

2.3 Geschäftsprozesse und Workflows

In diesem Kapitel werden die grundlegenden Begrifflichkeiten Geschäftsprozess (siehe Abschnitt 2.3.1) und Workflow (siehe Abschnitt 2.3.2) unterschieden. Service-orientierte Workflows als Komposition von Services werden in Abschnitt 2.3.3 behandelt.

2.3.1 Geschäftsprozesse

In der Organisationslehre wird unter einem Geschäftsprozess (engl. Business Process) eine inhaltlich abgeschlossene, zeitlich-sachlogische Abfolge von Aktivitäten verstanden, die zusammengenommen einen Wert für den Kunden schaffen, z. B. [40, 163]. Die *Workflowmanagement Coalition (WFMC)*, ein internationales Standardisierungsgremium für Workflowmanagement, definiert Geschäftsprozess wie folgt [161]:

> *"A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships."*

Eine Aktivität stellt einen Prozessschritt dar und beschreibt hierbei eine Ausprägung der drei Dimensionen *Ressourcen, Aufgaben* und *Anwendungsfälle*. So ist das Senden einer Empfangsbestätigung (Aufgabe) durch den Mitarbeiter einer Versicherung (Ressource) beim Eingang einer Schadensmeldung (Anwendungsfall) eine Aktivität.

Der Geschäftsprozess ist ein Vorgang, der funktions-, hierarchie- und standortübergreifend ablaufen kann. Weiterhin zeichnet sich ein Geschäftsprozess durch einen definierten Anfang, ein definiertes Ende, erforderliche Eingaben und durch produzierte Ergebnisse aus. Geschäftsprozesse können sich über das Unternehmen hinaus erstrecken und Aktivitäten von Kunden, Lieferanten und Partnern einschließen, wodurch an einem Geschäftsprozess eine unterschiedliche Anzahl unabhängiger, autonomer Teilnehmer beteiligt sein können.

Ein Geschäftsprozess kann des Weiteren aus unterschiedlichen Perspektiven betrachtet werden. In [1, 62] werden die vier verschiedene Perspektiven *Kontrollfluss, Daten, Ressourcen* und *Operationalisierung* unterschieden. Die Kontrollflussperspektive beschreibt Aktivitäten und ihre Ausführungsreihenfolge durch verschiedene Konstrukte, wobei diese Konstrukte die Ausführungsreihenfolge des Prozesses bestimmen. Die Datenperspektive befasst sich mit der Bereitstellung von Geschäfts- und Prozessdaten für die Kontrollflussperspektive. Dabei stellen betriebliche Dokumente und andere Objekte, die zwischen den einzelnen Aktivitäten ausgetauscht werden, Vor- und Nachbedingungen von Aktivitäten dar. Die Ressourcenperspektive verbindet die Organisationsstruktur mit dem Geschäftsprozess in Form von Allokationen von Personal und Betriebsmitteln zu einzelnen Aktivitäten. Die Operationalisierungsperspektive beschreibt die Zuordnung von Aktivitäten zu den entsprechenden Systemen und Anwendungen.

Eine an den Geschäftsprozessen orientierte Sichtweise hat, verglichen mit anderen konkurrierenden Ansätzen, wie z. B. einer daten- oder funktionsorientierten Sichtweise, seit Mitte der 90er Jahre in der Organisationslehre zunehmend an Bedeutung gewonnen, da die prozessorientierte Gestaltung des Unternehmens betriebliche Funktionen (wie z. B. Beschaffung, Vertrieb und Produktion) miteinander verknüpft und verschiedene Datenbasen integriert [120]. Somit spiegelt der prozessorientierte Ansatz die betriebliche Realität umfassender wider als z. B. der funktions- oder datenorientierte Ansatz [62].

Im Zuge der Globalisierung und des Business Process Outsourcing nimmt insbesondere die Bedeutung von organisationsübergreifenden Geschäftsprozessen zu [8, 48, 78, 116]. Organisationsübergreifende Prozesse verbinden mehrere Unternehmen zu Wertschöpfungsnetzwerken. Für heute oftmals hoch spezialisierte Unternehmen stellen organisationsübergreifende Geschäftsprozesse einen kritischen Erfolgsfaktor dar. Der Erfolg des einzelnen Unternehmens hängt maßgeblich vom Erfolg des gesamten Wertschöpfungsnetzwerkes ab [86].

In Abbildung 14 wird der stark vereinfachte organisationsübergreifende Geschäftsprozess „Reise buchen“ in Form eines UML-Aktivitätsdiagramms dargestellt. Nachdem die Reiseda-

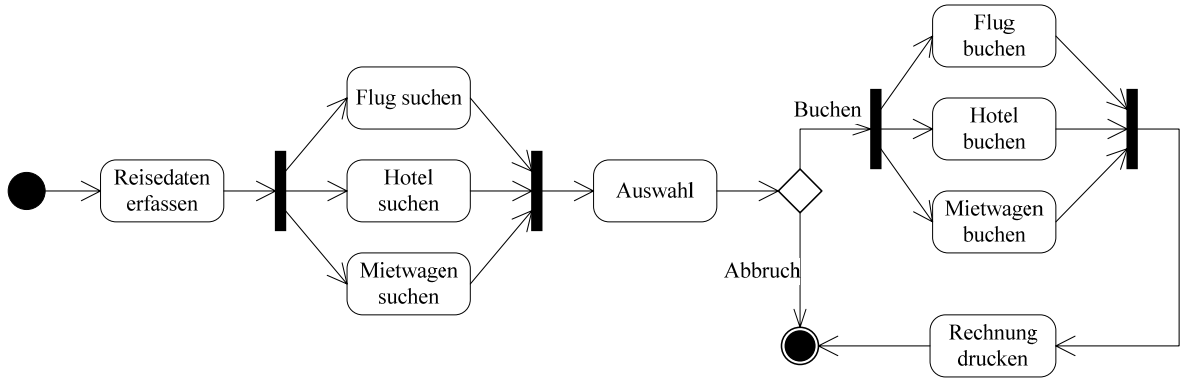


Abbildung 14: Beispiel eines vereinfachten Geschäftsprozesses

ten des Kunden erfasst wurden, wird parallel nach passenden Flügen, Hotels und Mietwagen gesucht. Aus den verfügbaren Flügen, Hotels und Mietwagen kann ein konkretes Reisepaket zusammengestellt werden. Ansonsten kann der Vorgang aber auch abgebrochen werden. Wird

der Vorgang fortgesetzt, so werden parallel der gewählte Flug, das Hotel und der Mietwagen gebucht. Abschließend wird die Rechnung ausgedruckt.

2.3.2 Workflows

Nicht alle Geschäftsprozesse werden zwangsläufig durch IT unterstützt. Geschäftsprozesse, die IT-unterstützt ablaufen, bezeichnet man als Workflows, z. B. [84]. Ein Workflow ist folglich die informationstechnische Realisierung und (Teil-)Automatisierung eines Geschäftsprozesses. Die Workflow Management Coalition definiert den Begriff Workflow als [160]:

> *"A workflow is the computerised facilitation or automation of a business process, in whole or part."*

An anderer Stelle, ebenfalls bei der Workflow Management Coalition, heißt es [161]:

> *"A workflow is the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another, according to a set of predefined rules."*

Ein System zur Ausführung und Steuerung von Workflows wird als *Workflow Management System* bezeichnet. Die WFMC definiert den Begriff Workflow Management System wie folgt [161]:

> *"A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications."*

Bei Workflow Management Systemen handelt es sich um anwendungsneutrale, generische Werkzeuge, die sich überall dort einsetzen lassen, wo geplante, strukturierte und arbeitsteilige Arbeitsabläufe vorzufinden sind.

Grundsätzlich wird zwischen Prozess- und Workflow-Modell unterschieden (siehe Abbildung 15). Hierbei beschreibt das Prozessmodel die Struktur des Geschäftsprozesses in der realen Welt [84]. Es stellt alle möglichen Pfade des Geschäftsprozesses dar und definiert, welche Aktivitäten wann ausgeführt werden müssen. Das Prozessmodell dient als Schema, von dem die konkreten Instanzen des Geschäftsprozesses abgeleitet werden [62, 84]. Das Workflow-Modell ist das Ergebnis der Workflow-Modellierung und beschreibt die rechnerunterstützten Teile des Prozessmodells. Synonym zu Workflow-Modell wird der Begriff Workflow-Schema verwendet. Das Workflow-Modell wird in einer Workflow-Sprache, die auch als Workflow-Metaschema bezeichnet wird, beschrieben. Eine konkrete Workflow-Instanz beschreibt eine Ausprägung des Workflow-Modells, die durch das Workflow Management System ausgeführt wird.

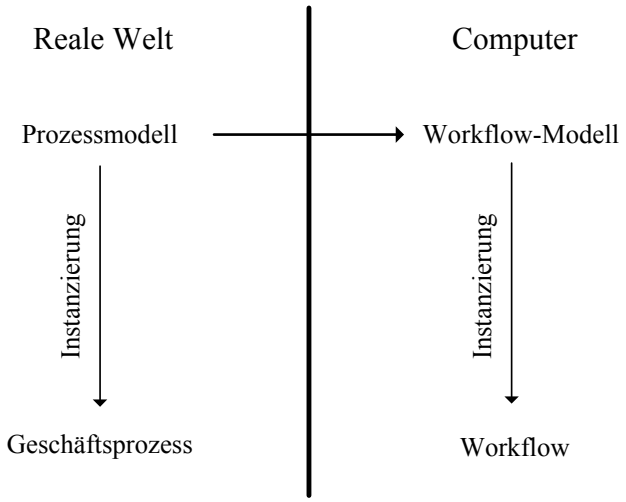


Abbildung 15: Unterscheidung Geschäftsprozess und Workflow [84]

Workflows lassen sich nach ihrer Ausführungshäufigkeit sowie dem Geschäftswert klassifizieren, z. B. [61, 84]. Die Ausführungshäufigkeit steht für die wiederholte und unveränderte Ausführung eines Workflow. Als Beispiel eines Workflow mit hoher Ausführungshäufigkeit kann der Produktionsprozess einer Serienfertigung genannt werden. Gleichzeitig weist dieses Beispiel eine hohe Bedeutung für den Geschäftswert auf und ist somit in die Kategorie *Production Workflow* einzuordnen. Die Dimension des Geschäftswertes steht für den Wert und damit das wirtschaftliche Risiko, welches der Durchführung des jeweiligen Prozesses innewohnt. Ein Fehler im genannten Serienfertigungsprozess ist wegen seiner Auswirkungen auf alle durch ihn gefertigten Produkte mit enormen Kosten verbunden. *Administrative Workflows* dienen nur mittelbar der Erreichung des Unternehmenszieles. Diese sind ebenfalls fest strukturiert und kommen häufig zur Ausführung, allerdings ist ihr Wertbeitrag für das Unternehmen eher gering. Als Beispiele hierfür können Prozesse im Rechnungswesen angeführt werden. *Ad-hoc Workflows* sind meist im Voraus nicht vollständig planbar und kommen nur sehr selten zur wiederholten Ausführung. Während Production Workflows und Administrative Workflows strukturierte Geschäftsprozesse zugrunde liegen, sind Ad-hoc Workflows nur schwach strukturiert. Bei *Kollaborativen Workflows* steht die Zusammenarbeit innerhalb eines Teams im Mittelpunkt. Kollaborative Workflows werden häufig durch Groupware unterstützt [61].

2.3.3 Service-orientierte Workflows

Service-orientierte Workflows beschreiben die Komposition von Services im Rahmen einer SOA. Somit kann ein Service-orientierter Workflow konzeptionell als eine Folge von Serviceaufrufen verstanden werden. Services können die Funktionalität bestehender Legacy-Anwendungen kapseln. Dies hat den Vorteil, dass bereits existierende Legacy-Anwendungen weiterbetrieben bzw. kontinuierlich ersetzt werden können. Die Flexibilität der Workflows ist trotz monolithischer Legacy-Anwendungen gewahrt, da die Funktionalität der Legacy-Anwendungen in Form von Services angeboten wird [46]. Änderungen an einem Workflow betreffen daher vorwiegend die Entscheidung, welche Services aufgerufen werden bzw. welche Aufrufreihenfolge gewählt wird (siehe Abbildung 16). So ist sichergestellt, dass eine Mo-

difizierung des Workflows keine direkten Auswirkungen auf die Legacy-Anwendungen hat. Dies entspricht dem Paradigma des *Two-Level-Programming* [84], d. h. es wird explizit zwischen der Modellierung des Workflows und der eigentlichen Anwendungsentwicklung

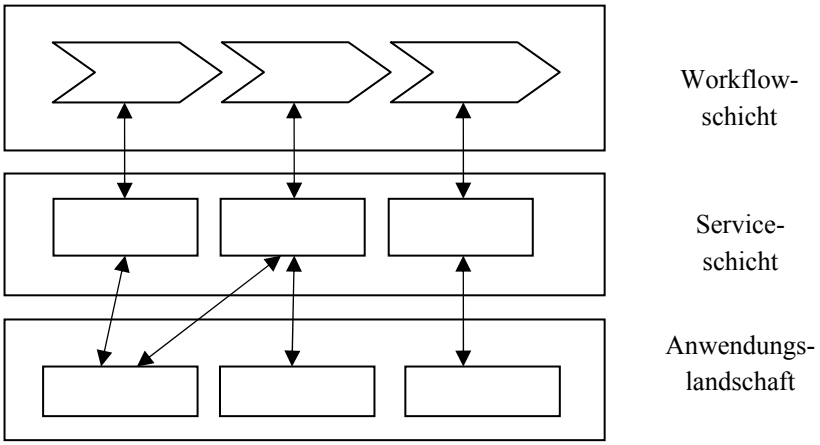


Abbildung 16: Abbildung von Prozessschritten auf Services

unterschieden. Hierbei wird die Modellierung des Workflows, also die Definition des Daten- und Kontrollflusses, als *Programming-in-the-large*, die eigentliche Implementierung der Services als *Programming-in-the-small* bezeichnet.

Um bestehende Workflows auf eine SOA abzubilden, wird der entsprechende Workflow zuerst konzeptionell in Prozessschritte (Aktivitäten) zerlegt und somit modularisiert, z. B. [41]. Die durch die Prozessschritte beschriebene Funktionalität wird dann durch Services bereitgestellt. Die Dekomposition des Workflows wird solange angewendet, bis Aktivitäten von geeigneter Granularität ermittelt sind. Ausschlaggebend für eine geeignete Granularität der Aktivitäten sind deren fachliche und funktionale Handhabung, die Umsetzbarkeit einer IT-Unterstützung sowie das Vorhandensein entsprechender Services. Abbildung 17 verdeutlicht diese Vorgehensweise am Beispiel eines vereinfachten Kreditprozesses [14]. Die Funktionalität dieser konzeptionell modellierten Prozessschritte wird nun durch Services bereitgestellt. Konzeptionelle Prozessschritte werden somit auf Services abgebildet.

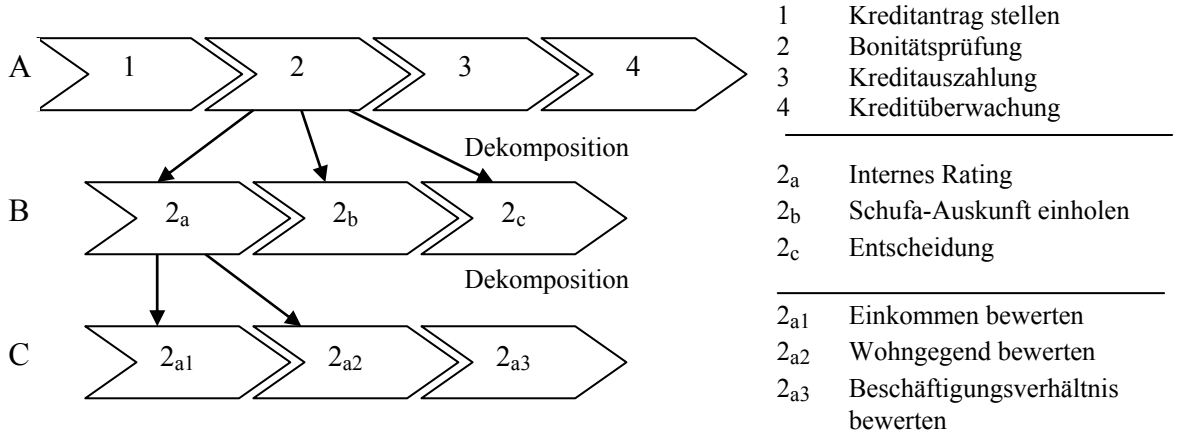


Abbildung 17: Dekomposition am Beispiel eines generischen Kreditprozesses

Dieser Ansatz ermöglicht zudem, dass Services auch von externen Anbietern bezogen werden können (siehe Abbildung 18). Im Gegensatz zum klassischen Business Process Outsourcing wird nicht ein gesamter Workflow an einen externen Dienstleister ausgelagert, sondern nur Prozessschritte oder Teilprozesse in Form von Services [6, 19]. Die Steuerung des Prozesses verbleibt weiterhin im Unternehmen. Die in diesem Abschnitt beschriebene Vorgehensweise unterstützt die Wiederverwendung von Services in verschiedenen Workflows. Um Services anderen Organisationseinheiten bzw. Unternehmen zugänglich zu machen, werden diese in Verzeichnisdiensten abgelegt [104]. In diesem Zusammenhang werden Services mit zusätzlichen Metainformationen beschrieben, sodass eine gezielte Suche basierend auf verschiedenen Kriterien möglich ist. Im Rahmen der Web Service Technologie beschreibt die UDDI-Spezifikation (vgl. Abschnitt 2.2.4) einen XML-basierten Standard für einen Verzeichnisdienst.

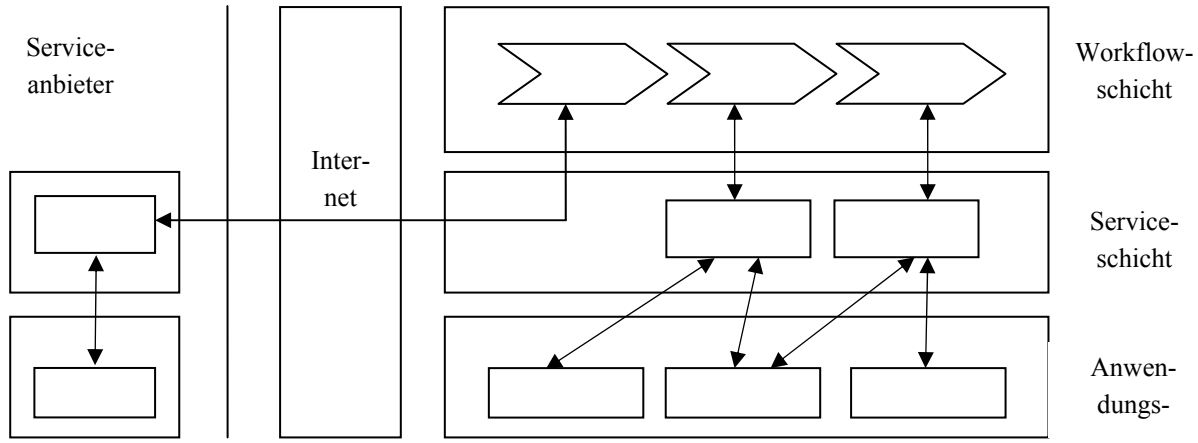


Abbildung 18: Service-basiertes Business Process Outsourcing

2.4 Dienstgüte

In diesem Abschnitt wird Dienstgüte als Ansatzpunkt für das technische Management Service-orientierter Workflows vorgestellt. Hierzu wird Dienstgüte zuerst allgemein betrachtet (siehe Abschnitt 2.4.1), bevor auf die Besonderheiten von Dienstgüte im Umfeld Service-orientierter Workflows eingegangen wird (siehe Abschnitt 2.4.2). Die Bedeutung von Service Level Agreements wird in Abschnitt 2.4.3 erläutert.

2.4.1 Definition

Für den Begriff *Dienstgüte* (engl. *Quality of Service*, Abk. *QoS*) existiert keine einheitliche Definition. Je nachdem, welche Art von Dienst unter welchen Aspekten betrachtet wird, setzt sich die Beurteilung der Güte dieses Dienstes aus anderen Teilaspekten zusammen. Auch wenn im Folgenden nur Dienste in Bezug auf verteilte Systeme und sie verbindende Netzwerke betrachtet werden, ergibt sich keine wesentliche Vereinfachung. Betrachtet man z. B. das Schichtenmodell eines Netzwerkes, wie das ISO-OSI-Referenzmodell, so wird in jeder Schicht eine Reihe von Diensten definiert, die wiederum durch die jeweils höhere Schicht verwendet werden. Durch die Dienste werden hierbei jeweils schichtspezifische Aufgaben adressiert und die Güte eines Dienstes daher in jeder Schicht anhand unterschiedlicher Kriterien beurteilt. So werden beispielsweise an einen Dienst der Bitübertragungsschicht (Übertra-

gung einzelner Bits über ein Übertragungsmedium) andere Anforderungen gestellt als an einen Dienst der Transportschicht (Aufbau und Trennen von Verbindungen, Zerlegen von Daten in Transporteinheiten, Flusssteuerung der Datenströme) oder der Anwendungsschicht (z. B. Versenden von E-Mails oder Übertragen von Dateien) [137].

Das CCITT (Commité Consultatif Internationale Télégrafique et Téléfonique) veröffentlichte daher im Zusammenhang mit der ISDN-Technologie eine möglichst allgemeine Definition von Dienstgüte und entwickelte ein Dienstgütemodell, das die technischen Aspekte eines Dienstes ebenso berücksichtigt wie die Verfügbarkeit und die Bedienung der Endgeräte. In diesem Zusammenhang definiert das CCITT Dienstgüte als [28]:

> *„The collective effect of service performances which determine the degree of satisfaction of a user of the service.“*

Unter „User“ ist hier nicht notwendigerweise ein menschlicher Benutzer zu verstehen, sondern auch elektronische Geräte oder Software. Im Folgenden soll sich das Grundverständnis von Dienstgüte an dieser allgemeinen Definition des CCITT orientieren. Da sich die genauen Kriterien an Dienstgüte jedoch aus dieser allgemeinen Definition nicht erschließen, müssen diese jeweils im Kontext, in welchem der Begriff Dienstgüte verwendet wird, definiert werden. Im Kontext von Netzwerken und Diensten, die Netzwerke benutzen, stehen zumeist die technischen Aspekte der Diensterbringung, wie z. B. die Übertragungsgeschwindigkeit in einem Netzwerk, im Vordergrund. So definiert [123] Dienstgüte wie folgt:

> *„Dienstgüte kennzeichnet das definierte, kontrollierbare Verhalten eines Systems bezüglich quantitativ messbarer Parameter.“*

Diese Art der Dienstgüte kann als technische Dienstgüte bezeichnet werden, wohingegen unter Dienstgüte im weiteren Sinne auch nicht-technische Aspekte von Dienstgüte fallen können, wie z. B. die Reputation des Dienstanbieters.

2.4.2 Dienstgüte für Service-orientierte Workflows

Dienstgüte im Rahmen einer SOA, realisiert z. B. auf Basis der Web Service Technologie, ist im Gegensatz zu Bereichen wie Multimedia, z. B. [95, 101, 131, 133, 134], oder Kommunikationsnetzwerke, z. B. [50, 51, 52, 123], ein noch wenig untersuchtes Gebiet, in dem sich noch keine Standards etablieren konnten. Gerade aber im Umfeld service-orientierter Workflows hat Dienstgüte als Ansatz für das technische Management eine besondere Bedeutung. Denn Unternehmen werden nur dann bereit sein, Services externer Anbieter in unternehmenskritischen Workflows zu verwenden, wenn deren Dienstgüteeigenschaften garantiert werden, z. B. [19, 99, 140].

Web Services können eine Fülle von Funktionalitäten bereitstellen und für die verschiedensten Zwecke in den unterschiedlichsten Umgebungen eingesetzt werden. Zusätzlich ist es möglich, Web Services unter einer Vielzahl von Aspekten zu betrachten, wie z. B. unter dem Gesichtspunkt ihrer Funktionalität, ihrer Eigenschaften als Softwarekomponente oder ihrer Eigenschaften als Komponente eines Netzwerkes. Es ist daher nicht möglich, eine bestimmte

Menge von Dienstgütekriterien für die Spezifikation der Dienstgüte eines Web Service für jeden Einsatzzweck zu definieren. Dementsprechend viele Varianten von Dienstgütekriterien für Web Services lassen sich in der Literatur finden, z. B. [49, 70, 108, 112]. Eine Auswahl der häufigsten genannten Dienstgütekriterien ist in Abbildung 19 grafisch dargestellt und wird im Folgenden diskutiert.

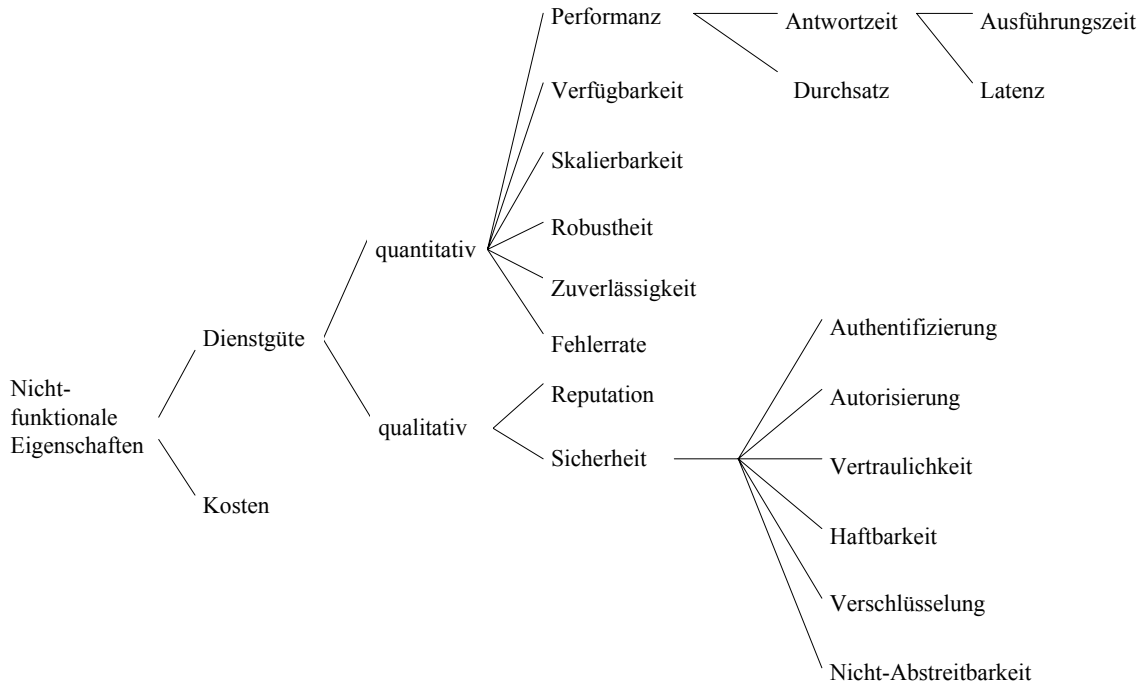


Abbildung 19: Klassifizierung von Dienstgütekriterien (in Anlehnung an [49, 98, 108, 112])

Ein wichtiges Kriterium bei der Auswahl zwischen Web Services gleicher Funktionalität neben den Dienstgüteeigenschaften sind die Kosten der Servicenutzung. Daher werden diese auch als weitere eigenständige nicht-funktionale Eigenschaft eines Service betrachtet [49]. In diesem Zusammenhang werden verschiedene Preismodelle, wie z. B. Flat-Rate- und Pay-per-use-Modelle, unterschieden [49, 93].

Performanz

Die Performanz als ein übergeordnetes Kriterium wird durch verschiedene Dienstgütekriterien bestimmt [108] (siehe Tabelle 1).

Kriterium	Beschreibung
Durchsatz	Anzahl von Anfragen, die in einem bestimmten Zeitintervall verarbeitet werden können [49, 98, 108].
Ausführungszeit	Zeit, die ein Web Service zur Abarbeitung einer Anfrage benötigt. Sie wird gemessen vom Zeitpunkt des Erhalts der Anfrage durch den Web Service bis zum Absenden der Antwort an den Empfänger [171].
Antwortzeit	Zeitspanne, die vom Absenden einer Anfrage an den Web Service bis zum Erhalt einer Antwort vergeht. Diese Zeitspanne beinhaltet die vor allem durch das Netzwerk determinierte Latenz sowie die Ausführungszeit des Web Service [49, 98].

Kriterium	Beschreibung
Latenz	Zeitspanne, die zwischen dem Absenden einer Anfrage und dem Empfang einer Antwort zwangsläufig vergehen muss und unabhängig von der Größe der Anfrage und der Anzahl der auszuführenden Operationen durch den Web Service ist. Üblicherweise wird diese Zeitspanne von der Dauer determiniert, die eine Nachricht minimaler Länge benötigt, um über das Netzwerk vom Nutzer zum Web Service und wieder zurück transportiert zu werden (sog. minimale Round-Trip-Time) [49, 108].

Tabelle 1: Dienstgütekriterium Performanz

Verfügbarkeit [98, 108]

Die Verfügbarkeit eines Web Service bezeichnet die Wahrscheinlichkeit, mit der ein Web Service durch einen Nutzer sofort in Anspruch genommen werden kann. Hierzu muss die Ausführungsumgebung, auf der der Web Service läuft, funktionstüchtig und über das Netzwerk erreichbar sein. Die Verfügbarkeit eines Web Service hängt eng mit dessen Zuverlässigkeit zusammen. Von besonderem Interesse im Zusammenhang mit der Verfügbarkeit ist die Time-to-Repair (TTR), welche die Zeit bezeichnet, die im Falle eines Ausfalls des Web Service benötigt wird, um diesen wieder zur Verfügung zu stellen. Die Verfügbarkeit wird als prozentualer Wert angegeben.

Skalierbarkeit [92, 112]

Skalierbarkeit drückt die Fähigkeit aus, die Verarbeitungskapazität eines Web Service so erhöhen zu können, dass eine gesteigerte Anzahl von Nutzeranfragen erfolgreich verarbeitet werden kann. Die Skalierbarkeit bezieht sich auf die Anzahl der durchführbaren Operationen und Transaktionen durch einen Web Service und steht in Zusammenhang mit dessen Performanz. Um eine gute Skalierbarkeit zu gewährleisten, muss der Web Service Anbieter in der Lage sein, die Ressourcen der an der Ausführung des Web Service beteiligten Systeme erweitern und die Last unter diesen Systemen verteilen zu können.

Robustheit [149]

Mittels der Robustheit wird der Grad an Funktionstüchtigkeit ausgedrückt, die ein Web Service selbst beim Auftreten ungültiger, unvollständiger oder widersprüchlicher Anfragen erbringt. Ein Web Service sollte auch unter solchen Umständen in der Lage sein, deterministisch zu reagieren.

Zuverlässigkeit [25, 49, 108]

Zuverlässigkeit repräsentiert die Fähigkeit eines Web Service, seine Funktionalität unter gegebenen Bedingungen für die Dauer eines bestimmten Zeitintervalls erfolgreich erbringen zu können. Zuverlässigkeit beinhaltet auch, dass während dieses Zeitintervalls sichergestellt ist, dass ankommende Anfragen angenommen und verarbeitet werden sowie korrekte Antworten erzeugt und in der richtigen Reihenfolge erfolgreich ausgeliefert werden. Als Indikator mangelnder Zuverlässigkeit kann das Auftreten von Fehlern angesehen werden. In diesem Fall

sind falsche Rückgabewerte, Ausfallzeiten, aber auch verloren gegangene Anfragen oder Antworten als Fehler anzusehen.

Fehlerrate [49]

Die Fehlerrate bezeichnet die Anzahl der Fehler innerhalb eines bestimmten Zeitintervalls. Grundsätzlich können in diesem Zusammenhang verschiedene Fehlerarten, wie z. B. fehlerhafte Ergebnisse oder Übertragungsfehler, unterschieden werden. Da der Entwickler eines Web Service nicht alle möglichen Fehler, die beim Betrieb eines Web Service entstehen können, vorhersagen kann, ist die Verwendung von Mechanismen zur Ausnahmebehandlung ein hilfreiches Mittel, um die Robustheit des Web Service zu erhöhen.

Sicherheit [39, 53, 98, 108]

Von einem Web Service wird erwartet, Sicherheitsmechanismen gemäß den Sicherheitsbedürfnissen der an der Kommunikation beteiligten Partner zur Verfügung zu stellen. Bei der Übermittlung von Prozessdaten eines verteilten Workflows an die beteiligten Web Services über das Internet ist die Bereitstellung grundlegender Sicherheitsmechanismen nahezu eine Voraussetzung ihrer Nutzung. Sicherheitsmechanismen können hierbei eine Vielzahl von Aspekten adressieren, wie z. B. Authentifizierung, Autorisierung, Vertraulichkeit, Haftbarkeit, Rückverfolgbarkeit/ Prüfbarkeit, Verschlüsselung und Nicht-Abstreitbarkeit. Authentifizierung bezeichnet die Fähigkeit, den Nutzer eines Web Service identifizieren zu können. Mittels der Autorisierung können Zugriffsberechtigungen auf Funktionen und Daten je nach identifiziertem Nutzer unterschiedlich festgelegt werden. Vertrauliche Daten sollten bei ihrem Austausch von und zu den autorisierten Nutzern entsprechend ihrer Vertraulichkeit übertragen und gehandhabt werden. Mittels der Verschlüsselung von Daten kann hierbei gewährleistet werden, dass nur der beabsichtigte Empfänger Daten entschlüsseln und verarbeiten kann. Ein Anbieter von Web Services sollte für die Funktionalität seines Web Service haftbar gemacht werden können. Voraussetzung hierfür ist die Rückverfolgbarkeit und Prüfbarkeit, die sicherstellt, dass Aufrufe von Web Services zurückverfolgt und die Ergebnisse des Aufrufs überprüft werden können. Mechanismen zur Nicht-Abstreitbarkeit garantieren, dass ein Nutzer nicht leugnen kann, einen Web Service aufgerufen und dessen Funktionalität genutzt zu haben.

Reputation [70, 87, 96]

In diesem Zusammenhang sind die bisherigen Erfahrungen mit dem jeweiligen Service bzw. dessen Anbieter zu bewerten. Dieses Kriterium ermöglicht, einen Service höher zu bewerten, wenn in der Vergangenheit schon positive Erfahrungen mit diesem bzw. mit dessen Anbieter gemacht wurden. Durch das Kriterium Reputation können zudem auch Referenzen des Anbieters mit in die Bewertung einbezogen werden. So kann der Anbieter eine Liste von Geschäftspartnern angeben, die diesen oder einen anderen Service des Anbieters nutzen. Die Reputation eines Anbieters ist die einzige nicht-funktionale Eigenschaft eines Web Service, die nicht vom Anbieter selbst angegeben werden kann. Sie muss stattdessen vom Nutzer, z. B. auf Grundlage der Referenzen oder Erfahrungen aus der Vergangenheit, bestimmt werden.

2.4.3 Service Level Agreements (SLAs)

Besonders im Rahmen von organisationsübergreifenden Geschäftsprozessen ist es für die beteiligten Partner wichtig, Vereinbarungen über die jeweils angebotene bzw. geforderte Leistung zu treffen. Zur Definition von Leistungsvereinbarungen werden Service Level Agreements verwendet. SLAs sind formale Dokumente mit vertragsähnlichem Charakter, die eine Vereinbarung zwischen dem Anbieter und dem Nutzer einer Leistung darstellen und die die Art und Weise der Leistungserbringung sowie relevanter Rahmenbedingungen festlegen, d. h. einen Service Level definieren, z. B. [6, 27].

Bei SLAs im Allgemeinen handelt es sich meist um natürlichsprachige Dokumente, für deren Aufbau keine genauen Vorgaben existieren. Je nach Art der Leistung, der Ausgestaltung der Leistungsinanspruchnahme und des Umfangs der Vereinbarungen können sich SLAs im Aufbau stark unterscheiden und mehr oder weniger komplex strukturiert sein. Typischerweise erfolgt die Strukturierung von SLAs ähnlich eines Vertrags mittels Klauseln, die jeweils einzelne Aspekte der Vereinbarung regeln. Übliche Klauseln, wie sie auch im RFC 3198 [157] beschrieben werden, betreffen hierbei die Dienstgüte, mit der eine Leistung erbracht werden muss. So werden die Voraussetzungen und Rahmenbedingungen, unter denen die Qualität garantiert wird, beschrieben. Zudem wird dargestellt, wie die Einhaltung der Qualitätsmerkmale überprüft wird und welche Konsequenzen eintreten, wenn vereinbarte Leistungen nicht eingehalten werden. Auch Supportvereinbarungen, Haftungsbestimmungen, Nutzungsentgelte und Kündigungsbedingungen sind Gegenstand eines SLAs.

Im Zusammenhang mit Service-orientierten Workflows können SLAs dazu verwendet werden, Leistungsvereinbarungen und Dienstgüteeigenschaften bezüglich der verwendeten Services einer SOA zu spezifizieren, z. B. [27, 70]. Die Verwendung von SLAs in der Form natürlichsprachiger Dokumente wäre hier zwar prinzipiell möglich, gestaltet sich aber durch den Mangel an automatischer Verarbeitbarkeit und dem Zwang zur menschlichen Interaktion als nachteilig. Sind an einem Workflow eine Vielzahl von verschiedenen Web Services beteiligt, die dynamisch zu Workflows orchestriert werden sollen, so wird der Einsatz natürlichsprachiger SLAs aufgrund des hohen Bearbeitungs- und Koordinationsaufwandes nahezu unmöglich. Damit SLAs für Web Services eingesetzt werden können, müssen diese in Form elektronischer Dokumente mit einem syntaktisch und semantisch standardisierten Format zur Verfügung stehen. Wesentlicher Bestandteil elektronischer SLAs für Web Services ist die Beschreibung der entsprechenden Leistungsmerkmale und der Kosten für die Servicenutzung. Die Leistungsmerkmale eines Web Service werden hierbei hauptsächlich durch seine Dienstgüte definiert. Es existieren Ansätze, die explizit elektronische, maschinenles- und maschineninterpretierbare SLAs für Services modellieren und eine Umgebung zur Kontrolle der Einhaltung dieser SLAs zur Verfügung stellen. IBM veröffentlichte hierzu ein Framework, welches die XML-basierte Sprache *Web Service Level Agreement (WSLA)* [88] zu Spezifikation von SLAs verwendet. Hewlett Packard entwickelte mit dem *Web Services Management Network* (WSMN) [90] ein Overlay-Netzwerk zum Management von Web Services, welches mit der *Web Services Management Language (WSML)* ebenfalls eine XML-basierte Sprache zur Spezifikation von SLAs verwendet.

Eine standardisierte, maschinenlesbare Form von SLAs schafft die nötigen Voraussetzungen für: (z. B. [43, 70, 117, 118, 122])

- Das automatisierte Erstellen eigener und das automatisierte Auslesen fremder SLAs
- Die automatisierte Aushandlung von SLAs als Übereinkunft zwischen den Anforderungen des Nutzers und dem Angebot des Anbieters
- Die automatisierte Kontrolle der spezifizierten Leistungsmerkmale
- Die automatisierte Vergleichbarkeit von verschiedenen Web Services anhand ihrer Leistungsmerkmale, inklusive der Berechnung einer Bewertungsmetrik als Maß der Übereinstimmung mit individuellen Nutzerpräferenzen

3 WSQoSX – Eine Dienstgüteunterstützende Web Service Architektur

Entscheidend für den weiteren Erfolg des SOA-Paradigmas in Produktivsystemen der Unternehmen ist ein umfassendes Dienstgütemanagement der entsprechenden Services, z. B. [19, 92, 140, 141, 149]. So sind Unternehmen nicht bereit, Services externer Anbieter in den geschäftskritischen Anwendungen und Workflows einzusetzen, wenn keine Garantien über deren Dienstgüteeigenschaften vorliegen. Zukünftig ist davon auszugehen, dass eine Vielzahl von Services gleicher bzw. zumindest ähnlicher Funktionalität zur Verfügung stehen wird, die von unterschiedlichen Anbietern bereitgestellt werden. Das eigentliche Differenzierungsmerkmal solcher funktional weitgehend gleicher Services sind neben den Kosten der Servicenutzung deren Dienstgüteeigenschaften. Das Management der Dienstgüte bedarf einer umfassenden Unterstützung seitens der zugrunde liegenden Architektur.

In diesem Kapitel wird daher als ein Beitrag der vorliegenden Arbeit mit WSQoSX (Web Services Quality of Service Architectural Extension) [10, 11, 12, 13, 129] eine erweiterte Web Service Architektur mit umfassender Dienstgüteunterstützung vorgestellt. WSQoSX ermöglicht die dynamische, dienstgütebasierte Komposition von Web Services zu Workflows. Ein Schwerpunkt liegt auf der Auswahl, Einbindung und Ausführung der beteiligten Web Services. Externe Anbieter müssen das Laufzeitverhalten ihrer Web Services in Form von SLAs garantieren. WSQoSX steuert und überwacht die Dienstgüte der entsprechenden Web Services auf Grundlage eines im Rahmen dieser Arbeit entwickelten Vorgehensmodells, das aus neun Phasen besteht. Als Proof-of-Concept wird eine prototypische Implementierung von WSQoSX vorgestellt.

Dieses Kapitel ist wie folgt gegliedert: Zu Beginn werden Anforderungen beschrieben, die eine erweiterte Service-orientierte Architektur erfüllen muss, um eine umfassende Dienstgüteunterstützung für Web Service Workflows sicherstellen zu können. Im Anschluss daran wird mit WSQoSX eine Architektur vorgestellt, die diese Anforderungen umsetzt. In Abschnitt 3.2 werden die an WSQoSX beteiligten Akteure und deren Zusammenwirken sowie ein mögliches Anwendungsszenario für den Einsatz von WSQoSX vorgestellt. Das Vorgehensmodell zum Dienstgütemanagement im Rahmen von WSQoSX wird in Abschnitt 3.3 dargestellt. Die Architektur von WSQoSX sowie die Funktionsweise der Architekturkomponenten ist Gegenstand von Abschnitt 3.4. Abschnitt 3.5 beinhaltet die Vorstellung der prototypischen Implementierung von WSQoSX. Kapitel 3 endet mit einer zusammenfassenden Darstellung der wesentlichen Leistungsmerkmale von WSQoSX.

3.1 Anforderungen an eine Web Service Architektur

In diesem Abschnitt werden Anforderungen an eine Web Service Architektur definiert, die das Management Service-orientierter Workflows ermöglichen:

- Integration externer Web Services
  Eine Web Service Architektur muss in der Lage sein, auch externe Services zu integrieren, auszuführen und zu verwalten. Dazu ist es wichtig, Ausschreibungen für Services, die extern bezogen werden sollen, zu veröffentlichen. Externen Anbietern muss die Möglichkeit gegeben werden, ihre Services als Reaktion auf solche Ausschreibungen anzubieten.
- Auswahl von Web Services in Abhängigkeit ihrer Dienstgüte
  Sind mehrere Web Services gleicher oder ähnlicher Funktionalität vorhanden, muss seitens des Nutzers entschieden werden, welcher der in Frage kommenden Web Services verwendet werden soll. Hierbei sollte der Nutzer insofern unterstützt werden, als dass ihm durch eine entsprechende Architekturkomponente ein Service zur Ausführung vorgeschlagen wird. Als Entscheidungskriterium werden die nicht-funktionalen Eigenschaften der Web Services herangezogen. Eine Voraussetzung hierfür ist, dass die Nutzerpräferenzen ermittelt und die in Frage kommenden Services anhand dieser Präferenzstruktur bewertet werden.
- Unterstützung von Service Level Agreements
  Ein Unternehmen wird in kritischen Workflows nur dann externe Services verwenden, wenn Garantien bezüglich deren nicht-funktionalen Eigenschaften existieren. Solche Garantien können in SLAs (vgl. Abschnitt 2.4.3) festgeschrieben werden. Die in den SLAs garantierten nicht-funktionalen Eigenschaften sind die Grundlage für die Bewertung der Services. Von einer Web Service Architektur wird daher erwartet, dass sie in der Lage ist, SLAs auszuwerten, zu verwalten und auf Einhaltung zu überprüfen.
- Überwachung des Laufzeitverhaltens von Web Services
  Die Überwachung des Laufzeitverhaltens einzelner Web Services und der durch diese realisierten Workflows ist eine wichtige Anforderung an eine Web Service Architektur. So kann festgestellt werden, ob Services gegen SLAs verstoßen. Die Überwachung des Laufzeitverhaltens ist aber auch insofern wichtig, als dass der Servicenutzer so überprüfen kann, ob er die Vereinbarung, die er seinerseits als Dienstleister gegenüber Kunden eingegangen ist, erfüllen kann. Das Protokollieren des Laufzeitverhaltens im Sinne eines Accounting (vgl. Abschnitt 3.4.5) ermöglicht einerseits eine verursachungsgerechte Kostenzuordnung innerhalb des eigenen Unternehmens, andererseits eine Überprüfung der durch externe Serviceanbieter gestellten Rechnungen.

3.2 Beteiligte Akteure und Anwendungsszenario

3.2.1 Beteiligte Akteure

Im Rahmen von WSQoSX werden, wie bei einer SOA üblich (vgl. Abschnitt 2.1.1), die beiden Rollen Serviceanbieter und Servicenutzer unterschieden. Seitens des Servicenutzers existieren die beiden Ausprägungen Entscheidungsträger und Administrator.

- Serviceanbieter
  Der Serviceanbieter meldet seine Web Services am WSQoSX-Portal des Nutzers an. Hierzu muss er verbindliche Angaben zur Dienstgüte des Web Service sowie zu den Rahmenbedingungen der Servicenutzung in Form eines SLAs machen. Meldet sich ein Anbieter erstmalig an, muss er sich zuvor mit Informationen über sein Unternehmen registrieren (vgl. Abschnitt 3.4.1). Bei Serviceanbietern kann es sich sowohl um externe Anbieter als auch um interne Fachabteilungen handeln.
- Servicenutzer
  Innerhalb eines Unternehmens, das WSQoSX zur Steuerung und Ausführung seiner Web Service Workflows nutzt, können die Rollen Administrator und Entscheidungsträger (siehe Abbildung 20) unterschieden werden. Der Administrator überwacht den technischen Betrieb von WSQoSX. Er wird über auftretende Fehler und SLA-Verletzungen benachrichtigt. Ein Entscheidungsträger innerhalb eines Unternehmens, das WSQoSX einsetzt, übernimmt Aufgaben im Zusammenhang mit der Verwaltung der Kategorien und der Bewertung von Web Services. Der Entscheidungsträger legt fest, welche Kategorien am Portal ausgeschrieben werden und bestimmt für diese die Mindestanforderungen und die Präferenzstruktur. Außerdem entscheidet er, ob für eine Kategorie eine dynamische oder statische Auswahl der Web Services durchgeführt wird (vgl. Abschnitt 3.4.4).

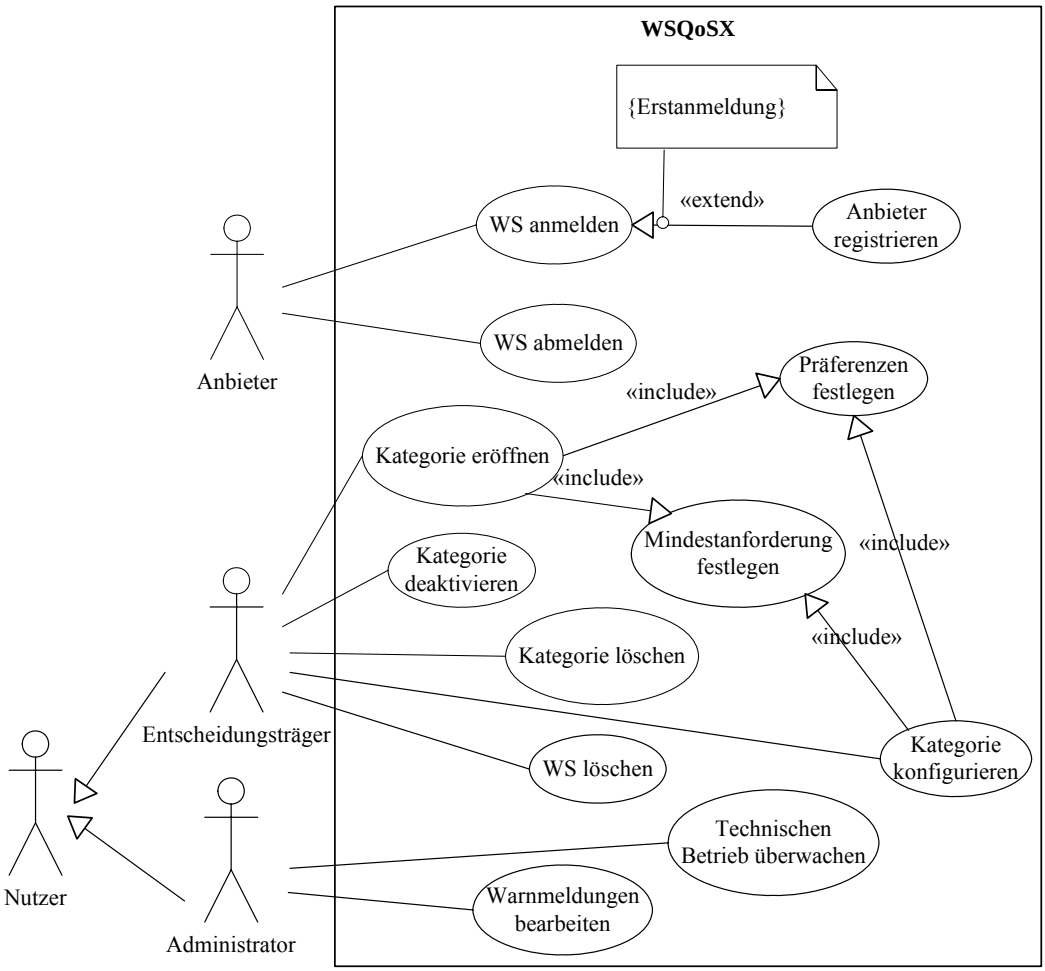


Abbildung 20: Beteiligte Akteure

3.2.2 Anwendungsszenario

Im Folgenden wird die Funktionsweise von WSQoSX anhand des Anwendungsszenarios „Kreditantrag bearbeiten“ näher beschrieben.

Der Geschäftsprozess einer Bank zur Bearbeitung eines Kreditantrags enthält einen Prozessschritt zur Überprüfung der Kreditwürdigkeit des Kunden. Die Funktionalität dieses Prozessschrittes soll durch einen Web Service bereitgestellt werden, der die Kreditwürdigkeit des Kunden in Form eines Ratings zurückliefert. Die Bank ist daran interessiert, diesen Prozessschritt durch den Web Service eines hierfür spezialisierten externen Anbieters ausführen zu lassen. Hierbei soll der Web Service möglichst preiswert sein, jedoch bestimmten Mindestanforderungen an seine Antwortzeit genügen. Die Bank veröffentlicht daher in ihrem WSQoSX-Portal eine Ausschreibung für die Kategorie „Bonitätsprüfung“. Dazu wird im Portal die geforderte Funktionalität textuell beschrieben und die Spezifikation der Schnittstelle in Form einer WSDL-Beschreibung veröffentlicht. Darüber hinaus definieren Entscheidungsträger innerhalb der Bank Mindestanforderungen an in Frage kommende Web Services. Zudem machen die Entscheidungsträger ihre Präferenzstruktur durch die Gewichtung der nichtfunktionalen Eigenschaften, wie z. B. Kosten und Antwortzeit, explizit. Anbieter, die einen Web Service mit der gewünschten Funktionalität in ihrem Portfolio haben, können diesen am WSQoSX-Portal der Bank als Reaktion auf Ausschreibung seitens der Bank anmelden. In einem SLA-Dokument müssen die Anbieter die zugesicherte Dienstgüte sowie weitere Rahmenbedingungen der Nutzung angeben. Bei der späteren Ausführung des Prozessschrittes „Bonitätsprüfung“ ruft WSQoSX automatisch denjenigen Web Service auf, der den Präferenzstrukturen des Entscheidungsträgers, „minimale Kosten verbunden mit einer Mindestanforderung an die Antwortzeit“, am besten genügt. Durch die Protokollierung der Web Service Aufrufe kann die Abrechnung der Nutzungsentgelte mit dem jeweiligen Anbieter kontrolliert werden. Sollte der Web Service zur Überprüfung der Kreditwürdigkeit gegen die durch den Anbieter garantierte Dienstgüte verstoßen, wird der Administrator hiervon in Kenntnis gesetzt.

3.3 Vorgehensmodell

WSQoSX erlaubt ein dienstgütebasiertes Management von Web Services, beginnend mit deren Anmeldung und Registrierung am Portal bis hin zur endgültigen Deaktivierung (siehe Abbildung 21). Im Folgenden wird ein im Rahmen dieser Arbeit entwickeltes, aus neun Phasen bestehendes Vorgehensmodell zum Dienstgütemanagement von Web Services vorgestellt. Im Folgenden wird beschrieben, wie diese Phasen durch WSQoSX unterstützt werden.

1. Phase: Anmeldung am WSQoSX-Portal des Nutzers

Unternehmen, die ihre Web Service Workflows durch WSQoSX steuern, können an ihrem von WSQoSX bereitgestellten Portal Ausschreibungen zu Web Services veröffentlichen, die sie an einen externen Anbieter vergeben wollen. Möchte ein externer Web Service Anbieter als Reaktion auf eine solche Ausschreibung einen Web Service zur Verfügung stellen, so muss er sich zuvor, falls noch nicht geschehen, am WSQoSX-Portal des ausschreibenden Unternehmens registrieren. Das Portal von WSQoSX wird in Abschnitt 3.4.1 beschrieben.

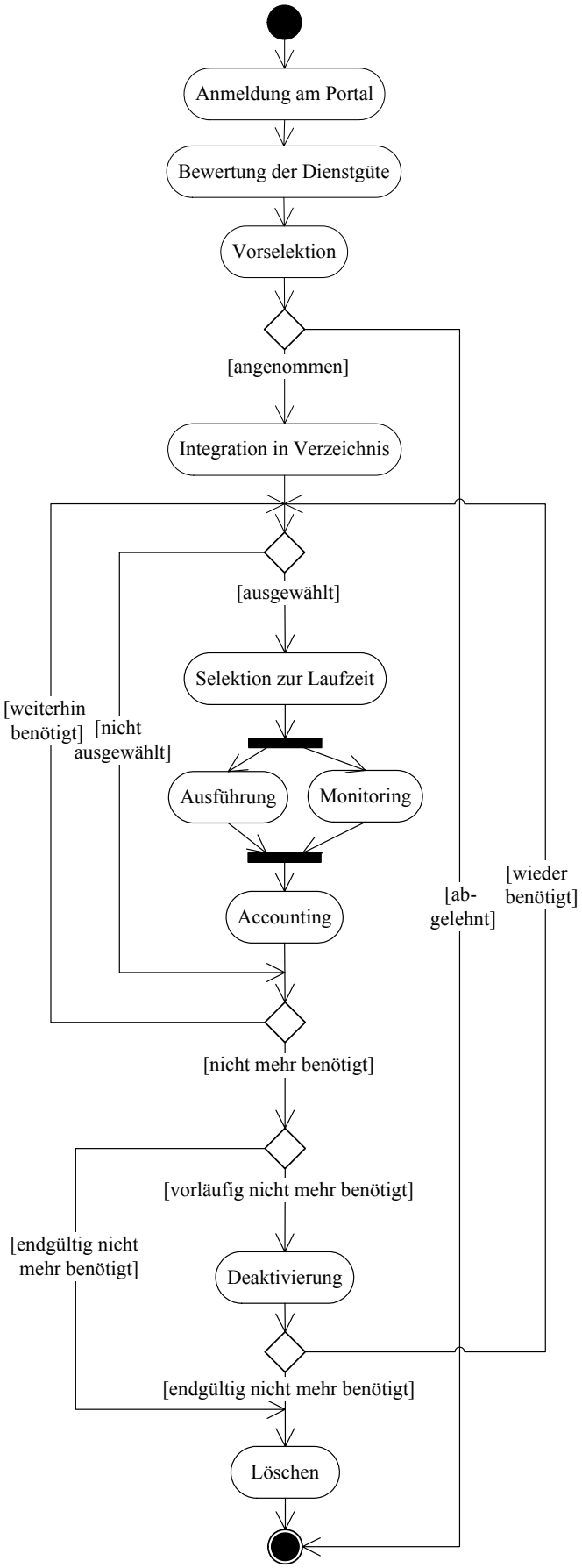


Abbildung 21: Vorgehensmodell zur Dienstgüteunterstützung

2. Phase: Bewertung der Dienstgüte

Nachdem der entsprechende Web Service mit zugehörigem SLA am Portal angemeldet wurde, erfolgt dessen Bewertung anhand der Dienstgütekriterien, die in den SLAs garantiert sind. Die Gewichtung dieser Dienstgütekriterien untereinander wird aus der Präferenzstruktur des Nutzers abgeleitet und geht ebenfalls mit in die Bewertung ein. Eine ausführliche Diskussion, wie die Dienstgüte der Web Services im Rahmen von WSQoSX bewertet wird, erfolgt in Abschnitt 3.4.3.

3. Phase: Vorselektion

In dieser Phase wird auf Grundlage der bisher vorgenommen Bewertung entschieden, ob der Web Service in das interne Verzeichnis übernommen wird. Hierzu können Ausschlussregeln definiert werden, die sich entweder auf die Gesamtbewertung des Web Service oder auf einzelne Teilkriterien beziehen. Eine Ausschlussregel bezüglich der Gesamtbewertung kann beispielsweise lauten: „Web Services, deren Gesamtbewertung kleiner als acht Punkte ist, werden nicht in das interne Verzeichnis übernommen.“ Bezogen auf einzelne Teilkriterien können Ausschlussregeln definiert werden, die z. B. besagen, dass Web Services, deren externe Referenzen mit weniger als vier Punkten bewertet wurden, abgelehnt werden. Es ist auch möglich, Ausschlussregeln miteinander zu verknüpfen. Neben den absoluten Regeln können auch relative definiert werden, die beispielsweise Web Services ausschließen, deren Gesamtbewertung nicht unter den 10 bestplatzierten liegt. Unabhängig von den Bewertungen können in der Phase Vorselektion auch solche Web Services ausgeschlossen werden, mit deren Anbieter der Nutzer, z. B. aus strategischen oder regulatorischen Gründen, nicht zusammenarbeiten möchte bzw. darf.

4. Phase: Integration eines Web Service in das interne Verzeichnis

Web Services, welche die Phase Vorselektion erfolgreich durchlaufen haben, werden in das interne Verzeichnis übernommen und stehen zur Integration in die jeweiligen Geschäftsprozesse zur Verfügung. Die Anbieter der nicht berücksichtigten Web Services werden über die Ablehnung (und ggf. über die Gründe hierfür) benachrichtigt.

5. Phase: Selektion eines Web Service zur Laufzeit

Ausschlaggebend, ob und wie häufig ein Web Service aus dem internen Verzeichnis tatsächlich ausgewählt wird, ist die Bewertung seiner Dienstgüteeigenschaften aus Phase 2. Grundsätzlich wird derjenige Web Service ausgewählt, der die höchste Gesamtbewertung in seiner Kategorie erhalten hat. Im Zusammenhang mit Geschäftsprozessen, an die besonders hohe Anforderungen gestellt werden, können aber auch noch für die in Frage kommenden Web Services zusätzliche Nebenbedingungen definiert werden, welche die Ausschlussregeln aus der Phase Vorselektion verschärfen. So ist es Unternehmen möglich, flexibel auf äußere Einflüsse zu reagieren. Sollte beispielsweise kurzfristig mit einem starken Anstieg von Kundenanfragen für einen bestimmten Web Service zu rechnen sein, wird durch eine zusätzliche Nebenbedingung festgelegt, dass der zum Einsatz kommende externe Web Service einen besonders hohen Durchsatz garantiert. Die Auswahl von Web Services in Abhängigkeit ihrer Dienstgüte wird in 3.4.4 beschrieben.

6. Phase: Ausführung

Derjenige Web Service, der in der vorigen Phase als am besten geeignet bezüglich der vorgegebenen Kriterien ermittelt wurde, wird nun dynamisch zur Laufzeit in den Workflow integriert und ausgeführt. Die Ausführung von Web Services zur Laufzeit durch WSQoSX wird in Abschnitt 3.4.4 vorgestellt.

7. Phase: Monitoring und Accounting

Für extern bezogene Web Services erhält das Unternehmen eine Abrechnung, die auf unterschiedlichen Preismodellen (z. B. nutzungsabhängige und -unabhängige Tarife) basieren kann [49]. Zur Erstellung dieser Abrechnung führt der externe Web Service Anbieter ein Accounting durch. Unter Accounting versteht man die verursachungsgerechte Zurechnung von Aktivitäten eines Informationssystems zu einer Ressource [5]. Aber auch für den Nutzer der Web Services ist ein Accounting von Bedeutung, da so eine verursachungsgerechte Kostenzuordnung zu den internen Organisationseinheiten, die den Service tatsächlich in Anspruch genommen haben, möglich wird. Auch werden die durch das Accounting gewonnenen Daten für die Berechnung der Dienstgütekriterien (z. B. Verfügbarkeit) verwendet. Eine genauere Beschreibung des Accountings im Rahmen von WSQoSX erfolgt in Abschnitt 3.4.5.

8. Phase: Deaktivierung eines Web Service

Wird die von einem Web Service bereitgestellte Funktionalität vorläufig nicht mehr benötigt, so kann der Nutzer diesen deaktivieren. Sollte der Web Service zu einem späteren Punkt wieder benötigt werden, so hat der Nutzer die Möglichkeit, diesen wieder zu aktivieren. Die Deaktivierung eines Web Service wird in Abschnitt 3.4.6 beschrieben.

9. Phase: Endgültige Löschung eines Web Service

Wird ein Web Service seitens des Nutzers endgültig nicht mehr benötigt, kann der Nutzer diesen aus dem Verzeichnis löschen. Es kann auch dann notwendig werden, einen Web Service zu löschen, wenn der Anbieter den entsprechenden Web Service aus seinem Angebot nimmt. Das Löschen eines Web Service wird in Abschnitt 3.4.6 beschrieben.

3.4 Architektur und Funktionsweise

WSQoSX besteht aus den drei Kernkomponenten *WSProxy, WSProxyAdmin* und *WSPortal,* deren Funktionalität in diesem Abschnitt detailliert beschrieben wird (siehe Abbildung 22). Neben diesen Kernkomponenten werden noch ein Datenbankserver sowie eine Workflow-Engine benötigt. Beide sind selbst kein Bestandteil von WSQoSX im engeren Sinne. Die Workflow-Engine nimmt aus der Perspektive von WSQoSX die Rolle eines Client ein, der Web Services aufruft. Allerdings werden im Rahmen von WSQoSX diese Aufrufe auf den *WSProxy* umgeleitet. Durch diesen zusätzlichen Indirektionsschritt beim Aufrufen von Web Services wird ein dynamisches Routing von Web Service Aufrufen möglich.

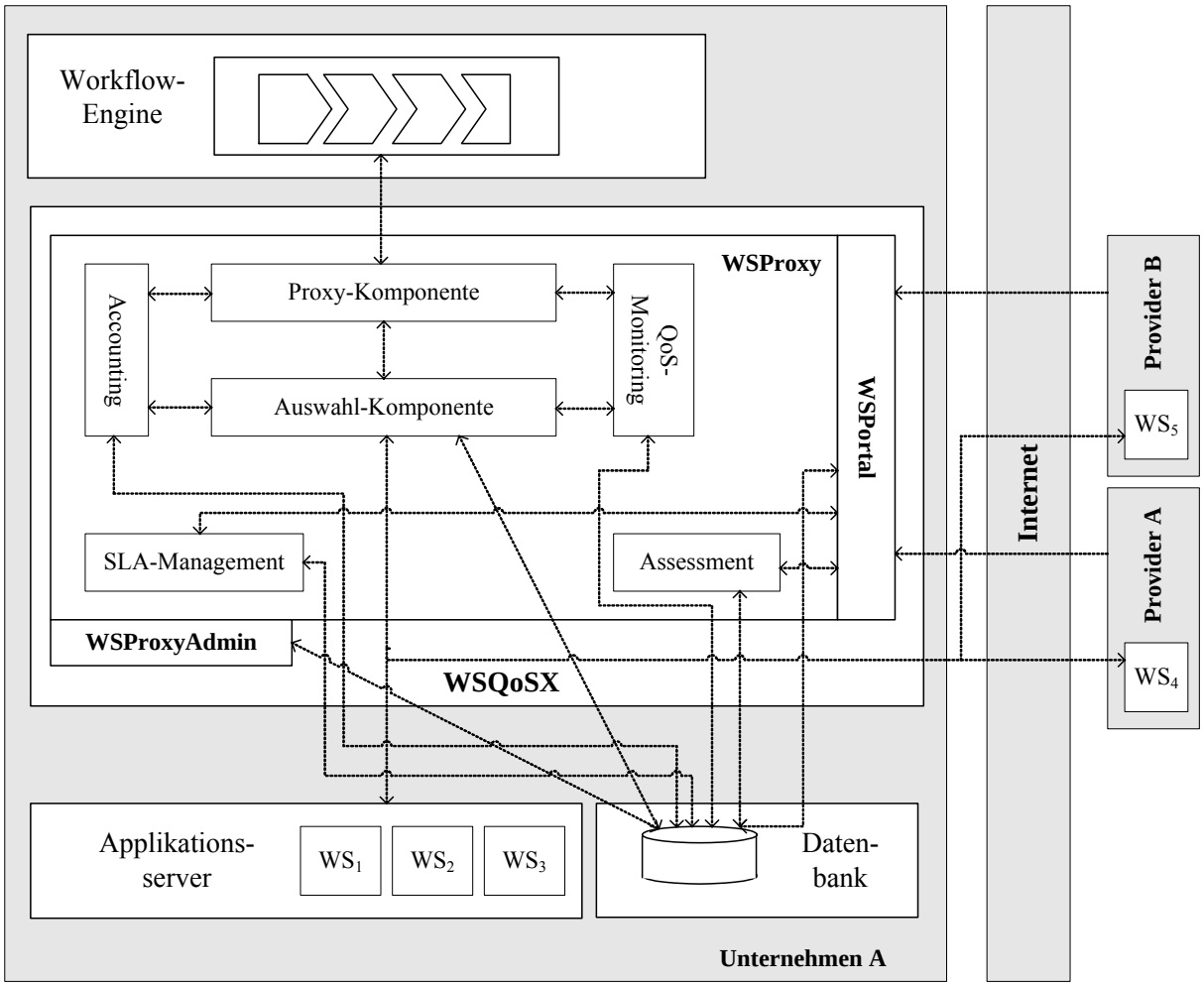


Abbildung 22: Architekturkomponenten von WSQoSX

3.4.1 Portalbasierte Anmeldung

WSQoSX stellt über die Komponente *WSPortal* eine Anbieterschnittstelle in Form eines Web Portals zur Verfügung, über das interne Fachabteilungen und externe Serviceanbieter ihre Web Services zur Nutzung bereitstellen können. Über das Portal kann ein Unternehmen, das WSQoSX verwendet, Ausschreibungen für benötigte Services potenziellen Nutzern bekannt machen. Das Portal kann mittels eines Webbrowsers plattformunabhängig verwendet werden. Bevor ein Anbieter das Portal verwendet, muss er sich einmalig durch Bekanntgabe seiner Kontaktinformationen registrieren. Bei der Registrierung des Anbieters werden Informationen über dessen Unternehmen, wie z. B. Firmennamen, Kontaktperson und Anschrift, erfasst. Diese Daten werden seitens des Nutzers benötigt, um in einem späteren Schritt entscheiden zu können, ob er mit diesem Anbieter zusammenarbeiten möchte bzw. um ihn im Falle von Problemen technischer oder organisatorischer Art erreichen zu können. Bei seiner Registrierung wählt der Anbieter zusätzlich einen Login-Namen und ein Passwort, um sich bei einem späteren Besuch am Portal identifizieren zu können.

Im Portal kann sich ein Anbieter die Liste sog. *Kategorien* (vgl. Abschnitt 3.4.2) anzeigen lassen, die momentan für die Anmeldung von Web Services freigegeben sind. Bei Kategorien handelt es sich um Ausschreibungen für Web Services seitens des Nutzers. Jede Kategorie

enthält Detailinformationen über die in dieser Kategorie geforderten Funktionalitäten und die Mindestanforderungen bezüglich der nicht-funktionalen Web Service Eigenschaften. Die Ausschreibung enthält ebenfalls Informationen darüber, welche Dienstgüteeigenschaften dem Nutzer im konkreten Falle von Bedeutung sind und wie deren Bedeutung untereinander gewichtet wird. Die erforderliche Schnittstelle ist mittels eines referenzierten WSDL-Dokumentes spezifiziert. Verfügt ein Anbieter über einen Web Service, der die benötigte Funktionalität bereitstellt, kann er diesen Web Service anmelden. Dazu muss er verbindliche Angaben zu den Dienstgüteeigenschaften und zur Nutzungsgebühr des von ihm offerierten Web Service in Form eines Service Level Agreements machen (siehe Anhang, Abschnitt A.1.2). Nachdem die Eigenschaften des Web Service bewertet und die Erfüllung von Mindestanforderungen an den Web Service geprüft wurden, wird entschieden, ob der Web Service zur produktiven Verwendung freigegeben wird. Der Anbieter sieht in der Liste seiner angemeldeten Web Services, ob seine Web Services bereits überprüft wurden oder ob eine Überprüfung noch aussteht.

Folgende Funktionalitäten werden dem Anbieter über WSPortal bereitgestellt:

- Anmelden (Login) des Anbieters am Portal
- Aufrufen einer Übersicht von Kategorien, in denen Web Services angemeldet werden können
- Anmelden von Web Services in einer Kategorie
- Aufrufen einer Übersicht von Web Services, die durch den entsprechenden Anbieter bereits am Portal angemeldet wurden
- Löschen von Web Services, die durch den Anbieter bereits am Portal angemeldet wurden
- Bearbeiten der Kontaktdaten des Anbieters
- Abmelden (Logout) des Anbieters vom Portal

Um sich mit einem Web Service in einer Kategorie zu bewerben, muss der Anbieter nach der Auswahl der entsprechenden Kategorie eine Reihe von Informationen in ein Formular eintragen:

- Verwaltungsinformationen
  Der Anbieter muss dem entsprechenden Web Service einen Namen und eine Kurzbeschreibung zuordnen. Diese Angaben erleichtern sowohl dem Anbieter als auch dem Nutzer die Identifikation und das spätere Auffinden des Web Service.
- Schnittstellenspezifikation des Web Service
  Die Angabe der Schnittstellenspezifikation des Web Service erfolgt über die Angabe der URL des entsprechenden WSDL-Dokumentes. Nach dem Absenden der Anmeldeinformationen wird das WSDL-Dokument von der angegebenen URL eingelesen und der Netzwerkendpunkt des Web Service aus den `port`-Elementen des WSDL-Dokumentes extrahiert. Das WSDL-Dokument wird durch die Assessment-

Komponente lokal gesichert, da es einen Teil der Vereinbarung darstellt, die der Anbieter dem Nutzer zum Zeitpunkt der Anmeldung des Web Service verbindlich zusichert.

- Nicht-funktionale Eigenschaften des Web Service
  Nicht-funktionale Eigenschaften des Web Service können bei der Anmeldung auf zweierlei Weise durch den Anbieter übermittelt werden. Entweder trägt der Anbieter die nicht-funktionalen Eigenschaften einzeln in einem Eingabeformular ein oder er gibt die URL zu einem SLA-Dokument an, welches alle Angaben zu den nicht-funktionalen Eigenschaften enthält. Wurde die URL eines SLA-Dokumentes angegeben, so wird das Dokument von der angegebenen URL gelesen und die nicht-funktionalen Eigenschaften daraus extrahiert. Das SLA-Dokument wird, wie auch das WSDL-Dokument, durch die Assessment-Komponente lokal gesichert.

3.4.2 Kategorisierung

Ein Anbieter kann am Portal nicht beliebige Web Services anmelden, sondern nur solche, die vom Nutzer benötigt werden. Welche Web Services benötigt werden, wird durch *Kategorien* festgelegt. Kategorien gruppieren Web Services gleicher bzw. ähnlicher Funktionalität. Jede Kategorie verfügt über eine informelle Beschreibung ihrer Funktionalität sowie eine Schnittstellenspezifikation in Form eines WSDL-Dokumentes. Zu jeder Kategorie werden folgende Daten erfasst:

- Name der Kategorie und Kurzbeschreibung
- Dokument mit ausführlicher textueller Beschreibung der Funktionalität
  Um die funktionalen Eigenschaften einer Kategorie näher zu spezifizieren, wird die URL eines Dokumentes angegeben, welches eine ausführlichen Beschreibung der Funktionalität enthält, die durch alle Web Services dieser Kategorie erbracht werden muss.
- WSDL-Dokument der Schnittstellenspezifikation
  Damit sichergestellt wird, dass eine Anfrage an jeden Web Service innerhalb einer Kategorie versendet werden kann, müssen alle Web Services einer Kategorie dieselbe Schnittstelle implementieren. Hierzu wird die URL zu einem WSDL-Dokument angegeben, welches die Spezifikation der zu implementierenden Schnittstelle der Kategorie enthält.
- Vorlage für ein SLA-Dokument
  Zur Übermittlung der nicht-funktionalen Eigenschaften eines Web Service kann ein vordefiniertes SLA-Dokument verwendet werden. Um dem Anbieter die Erstellung eines solchen SLA-Dokumentes zu erleichtern, wird in jeder Kategorie die URL zu einer Formatvorlage eines SLA-Dokumentes angegeben.
- Präferenzen und Anforderungen an nicht-funktionale Eigenschaften
  In jeder Kategorie können neben den Anforderungen an die funktionalen Eigenschaften eines Web Service auch Mindestanforderungen und Präferenzen bezüglich nicht-funktionaler Eigenschaften definiert werden. Durch Mindestanforderungen werden

Kriterien festgelegt, die ein Web Service mindestens erbringen muss, um in dieser Kategorie als produktiver Web Service verwendet zu werden. Mittels der Präferenzen wird definiert, welche nicht-funktionalen Eigenschaften bei der Auswahl eines Web Service zur Laufzeit mit welcher Gewichtung zu berücksichtigen sind.

3.4.3 Bewertung der Web Services

Die Bewertung der Web Services im Rahmen von WSQoSX basiert auf verschiedenen nicht-funktionalen Eigenschaften der entsprechenden Web Services. Neben den Dienstgüteeigenschaften im engeren Sinne können auch weitere Eigenschaften, wie z. B. die Rahmenbedingungen der Servicenutzung, bisherige Erfahrungen mit dem Service oder dem entsprechenden Anbieter berücksichtigt werden. Die Entscheidungsträger seitens des Web Service Nutzers können gemäß Ihrer Präferenzstruktur die gewünschten Kriterien auswählen, die als Bewertungsgrundlage herangezogen werden. WSQoSX kann des Weiteren durch Hinzufügen zusätzlicher, bisher nicht berücksichtigter, Kriterien erweitert werden. Die nicht-funktionalen Eigenschaften müssen durch den Anbieter bei der Anmeldung des Web Service spezifiziert werden.

Präferenzen und Mindestanforderungen in einer Kategorie

Alle Web Services einer Kategorie stellen die gleiche Funktionalität zur Verfügung, die einer vorgegebenen Schnittstellenspezifikation entspricht. Wird zur Laufzeit die Funktionalität einer Kategorie benötigt, so wird unter allen in dieser Kategorie verfügbaren Web Services derjenige ausgewählt und aufgerufen werden, dessen Bewertung der vorgegebenen Präferenzstruktur am besten entspricht.

Die Definition von Präferenzen für die Auswahl eines Web Service erfolgt über eine Gewichtung der nicht-funktionalen Eigenschaften. Die Gewichtung kann durch den Nutzer über die Kategorieverwaltung des Administrations-Frontends spezifisch für jede Kategorie vorgenommen werden. Für jede nicht-funktionale Eigenschaft wird hierbei eine Gewichtung festgelegt. Je höher das Gewicht einer Eigenschaft, desto stärker wird diese Eigenschaft bei der Bewertung, und damit auch bei der Auswahl, eines Web Service berücksichtigt. Bezeichnet K die Anzahl der nicht-funktionalen Eigenschaften, d. h. der Parameterwerte, der Web Services $WS_{i,j}$ einer Kategorie i, so gilt:

$$\forall_{i=1,\ldots,n}.0 \le w_{i,k} \le 1.\sum_{k=1}^{K} w_{i,k} = 1 \qquad (1)$$

Neben der Definition von Präferenzen bei der Auswahl eines Web Service kann die Assessment-Komponente über eine Vorselektion Web Services, die nicht den Anforderungen seitens des Nutzers entsprechen, von der Nutzung in Produktivsystemen ausschließen. Dadurch wird verhindert, dass in einem kritischen Workflow Web Services verwendet werden, die bestimmten Anforderungen nicht genügen. Zu diesem Zweck erlaubt WSQoSX, für jede Kategorie Mindestanforderungen an Web Services in Form von Regeln zu definieren.

Entsprechende Regeln können durch den Nutzer über die Kategorieverwaltung des Administrations-Frontends definiert werden. Zu jeder Kategorie lassen sich beliebig viele Regeln definieren. Jede Regel besteht hierbei aus drei Teilen: Dem Namen einer nicht-funktionalen Eigenschaft, einem Vergleichsoperator und einem numerischen Vergleichswert. Um zu entscheiden, ob eine Regel durch einen Web Service erfüllt wird, vergleicht die Assessment-Komponente die definierte nicht-funktionale Eigenschaft des Web Service mit dem entsprechenden Wert. Ein Web Service erfüllt genau dann die definierte Mindestanforderung einer Kategorie, wenn er alle definierten Regeln dieser Kategorie erfüllt. Beispiele für Regeln sind:

- „Der selektierte Web Service muss einen Durchsatz von mindestens 1.000 Aufrufen pro Tag bewältigen können."
- „Der selektierte Web Service muss eine Verfügbarkeit von mehr als 99,999% aufweisen."
- „Der selektierte Web Service muss über eine Antwortzeit von höchstens 10.000 ms verfügen."

Es lassen sich jedoch auch Regeln für nicht-funktionale Eigenschaften definieren, die nicht durch numerische Werte beschrieben werden können, wie z. B. für die Eigenschaft „Verschlüsselung". In einem solchen Fall erfolgt der Vergleich nicht mit dem eigentlichen (nicht-numerischen) Eigenschaftswert, sondern mit der numerischen Punktbewertung, die ein Nutzer diesem Eigenschaftswert während der Bewertung des Web Service zugeordnet hat.

Präferenzen und Mindestanforderungen müssen durch den Nutzer bereits zum Zeitpunkt der Erstellung einer Kategorie festgelegt werden, können jedoch jederzeit an die aktuellen Bedürfnisse angepasst werden. So kann z. B. in einer Phase hohen Transaktionsaufkommens in einer Kategorie die Präferenz auf Web Services verschoben werden, die einen hohen Durchsatz sowie eine hohe Verfügbarkeit verbunden mit höheren Kosten bieten. Fällt das Transaktionsaufkommen wieder geringer aus, kann die Präferenz auf preisgünstigere Web Services verlegt werden.

Damit WSQoSX zur Laufzeit feststellen kann, welcher Web Service innerhalb einer Kategorie am besten zur Erbringung der Funktionalität geeignet ist, wird jeder Web Service anhand seiner nicht-funktionalen Eigenschaften bewertet. Die Bewertung erfolgt durch die Berechnung eines numerischen Wertes, der sog. *Score* $s_{i,j}$. Je höher diese Score, desto besser entspricht dieser Web Service den Anforderungen des Nutzers. Die Score eines Web Service kann nicht automatisch nach der Anmeldung eines Web Service berechnet werden, da WSQoSX nicht selbstständig aus allen Dienstgüteeigenschaften eines Web Service eine Bewertung ableiten kann. So können zwar bei quantifizierbaren Dienstgüteeigenschaften automatisiert Vergleiche durchgeführt und Bewertungen errechnet werden, bei qualitativen Eigenschaften ist dies jedoch häufig nicht möglich.

Quantifizierbare Eigenschaften können durch die Assessment-Komponente verglichen werden. Beispielsweise kann die Assessment-Komponente unter den beiden Antwortzeiten

1.000 ms und 5.000 ms selbstständig das bessere Antwortzeitverhalten erkennen und die kürzere Antwortzeit als besser bewerten.

Qualitative Eigenschaften stellen weiche Eigenschaften dar und können nicht automatisch miteinander verglichen werden. Der Assessment-Komponente ist weder bekannt, wie weiche Eigenschaften durch einen Anbieter syntaktisch beschrieben werden noch was diese Angaben semantisch bedeuten bzw. wie diese vom Nutzer im konkreten Einsatzbereich des Web Service beurteilt werden. Die Assessment-Komponente ist beispielsweise nicht in der Lage festzustellen, ob eine Verschlüsselung mittels *DES (Data Encryption Standard)* vorteilhafter ist als eine Verschlüsselung mittels *IDEA (International Data Encryption Algorithm)*.

Manuelle Punktvergabe für qualitative Eigenschaften

Um eine Vergleichbarkeit der qualitativen Eigenschaften im Rahmen der Bewertung eines Web Service zu erreichen, müssen diese durch den Nutzer individuell bewertet werden. Erst nachdem alle weichen Eigenschaften bewertet wurden, kann eine Bewertung des Web Service als Ganzes erfolgen. Die Bewertung der weichen Eigenschaften erfolgt durch den Nutzer über das Administrations-Frontend in der Form von ganzzahligen *Punkten* im Wertebereich zwischen 0 und 10. Hierbei entsprechen 0 Punkte der Bewertung *ungenügend* und 10 Punkte der Bewertung *hervorragend*.

Normalisierung quantifizierbarer Eigenschaften

Die Vergabe von Punkten für quantifizierbare Eigenschaften erfolgt automatisch durch die Assessment-Komponente und wird in Form einer Normalisierung der Eigenschaftswerte unter allen Web Services einer Kategorie vorgenommen. Hierbei erhält der beste Eigenschaftswert innerhalb einer Kategorie 10 Punkte. Alle anderen Eigenschaftswerte erhalten eine Punktzahl (zwischen 0 und 10), welche die relative Größe des Eigenschaftswerts zum besten Eigenschaftswert der Kategorie repräsentiert. Je nach betrachteter Eigenschaft werden entweder kleinere oder größere Werte als bessere Eigenschaftswerte interpretiert. Im Falle der Verfügbarkeit und des Durchsatzes werden höhere Werte besser eingestuft als niedrigere Werte. Eine Normalisierung dieser Eigenschaften erfolgt daher am jeweiligen Maximalwert der Eigenschaften in einer Kategorie. Zur Berechnung der normalisierten Werte $q_{i,j,k}$ wird die entsprechende Dienstgüteeigenschaft $p_{i,j,k}$ des Web Service $WS_{i,j,}$ durch den Maximalwert der entsprechenden Dienstgüteeigenschaften einer Kategorie dividiert:

$$q_{i,j,k} = \frac{p_{i,j,k}}{\max(p_{i,1,k}, p_{i,2,k}, \ldots, p_{i,m,k})} * 10 \qquad (2)$$

Die Antwortzeit und der Preis werden hingegen umso besser eingestuft, je niedriger die spezifizierten Werte sind. Eine Normalisierung erfolgt in diesem Fall an dem jeweiligen Minimalwert in einer Kategorie. Zur Berechnung der normalisierten Werte $q_{i,j,k}$ wird der jeweilige Minimalwert der entsprechenden Dienstgüteeigenschaften einer Kategorie durch die Dienstgüteeigenschaft $p_{i,j,k}$ des jeweiligen Web Service dividiert:

$$q_{i,j,k} = \frac{\min(p_{i,1,k}, p_{i,2,k}, \ldots, p_{i,m,k})}{p_{i,j,k}} * 10 \quad (3)$$

Ein Beispiel für eine Normalisierung kann Tabelle 2 entnommen werden.

	Web Service 1		Web Service 2	
Eigenschaft	**Eigenschaftswert $p_{i,j,k}$**	**Normalisierter Wert $q_{i,j,k}$**	**Eigenschaftswert $p_{i,j,k}$**	**Normalisierter Wert $q_{i,j,k}$**
Verfügbarkeit	98,5 [%]	9,95	99 [%]	10
Durchsatz	20.000 [Aufrufe/ Tag]	10	18.000 [Aufrufe/ Tag]	9
Antwortzeit	10.000 [ms]	8	8.000 [ms]	10
Kosten	0,05 [EUR/ Aufruf]	4	0,02 [EUR/ Aufruf]	10

Tabelle 2: Beispiel für die Normalisierung quantitativer Eigenschaften

Berechnung der Score

Nachdem die normalisierten Werte $p_{i,j,k}$ der quantifizierbaren Eigenschaften von der Assessment-Komponente berechnet wurden und der Nutzer die Punkte zur Bewertung der weichen Kriterien spezifiziert hat, wird durch die Assessment-Komponente für jeden Web Service $WS_{i,j}$ die Score $s_{i,j}$ berechnet. Diese ist die gewichtete Summe aus den normalisierten Werten der quantitativen bzw. den zugewiesenen Punkten der qualitativen Eigenschaften. Zur Gewichtung werden die in der Kategorie definierten Gewichte für die Eigenschaften verwendet. Mit diesen Definitionen ergibt sich die Berechnungsvorschrift der Score $s_{i,j}$ eines Web Service $WS_{i,j}$ als:

$$s_{i,j} = \sum_{k=1}^{K} q_{i,j,k} * w_{i,k} \quad (4)$$

Werden die Gewichte der Eigenschaften in einer Kategorie so festgelegt, dass sie in der Summe 1 ergeben, so impliziert dies, dass die Score ebenfalls einen Wertebereich von 0 bis 10 aufweist.

Wird ein Web Service einer Kategorie hinzugefügt oder aus dieser gelöscht, so können sich neue Maxima oder Minima für Eigenschaftswerte in dieser Kategorie ergeben. Die Punktvergabe durch die Normalisierung der Eigenschaftswerte ist daher bei jedem Hinzufügen oder Löschen eines Web Service nötig und wird von der Assessment-Komponente in der betroffenen Kategorie automatisch durchgeführt. Da sich die normalisierten Werte der quantifizierbaren Eigenschaften ändern können, wird auch die Score für jeden Web Service erneut berechnet.

Überprüfung von Mindestanforderungen

Bevor einem Web Service seine Score zugeteilt wird, erfolgt zunächst noch eine Überprüfung der definierten Mindestanforderungen, die durch alle Web Services einer Kategorie erfüllt werden müssen. Nacheinander werden alle Regeln, durch welche die Mindestanforderungen

definiert sind, überprüft. Kann der Web Service alle Regeln erfüllen, so wird ihm seine berechnete Score zugeteilt und ist damit in das interne Verzeichnis produktiv einsetzbarer Web Services von WSQoSX aufgenommen. Kann auch nur eine der Regeln nicht erfüllt werden, so wird der Web Service abgelehnt und ihm die Score -1 zugeteilt. Anhand der Score lassen sich daher drei Zustände eines Web Service im internen Verzeichnis eindeutig unterscheiden (siehe Tabelle 3)

Score	Status im internen Verzeichnis
Noch nicht vergeben	Die Bewertung des Web Service durch den Nutzer steht noch aus.
≥ 0	Der Web Service wurde durch den Nutzer bewertet und erfüllt die entsprechenden Mindestanforderungen.
-1	Der Web Service wurde durch den Nutzer bewertet, konnte jedoch nicht den Mindestanforderungen genügen.

Tabelle 3: Zusammenhang zwischen Score und Status

3.4.4 Auswahl und Ausführung

Über das Administrations-Frontend *WSProxyAdmin* wird festgelegt, wie der WSProxy die Web Service Aufrufe seitens der Workflow-Engine verarbeiten und weiterleiten soll. WSProxy kann einen Web Service Aufruf entweder statisch an einen bestimmten Web Service weiterleiten oder dynamisch zur Zeit des Aufrufs einen geeigneten Web Service ermitteln. Soll der Web Service Aufruf dynamisch erfolgen, wird eine Kategorie (vgl. Abschnitt 3.4.2) definiert, aus der ein entsprechender Web Service ausgesucht wird. Bei der Selektion eines Web Service aus einer Kategorie ermittelt die Auswahl-Komponente denjenigen Web Service, der innerhalb der entsprechenden Kategorie die höchste Bewertung erhalten hat (vgl. Abschnitt 3.4.3). Hat die Auswahl-Komponente den entsprechenden Web Service identifiziert, ruft sie diesen auf, empfängt dessen Antwort und gibt diese an die Proxy-Komponente zurück. Die Proxy-Komponente reicht die Antwort wiederum an die Workflow-Engine zurück.

Um den dynamischen Aufruf von Web Services zu ermöglichen, wird lediglich bei der Aufnahme der Web Services in das interne Verzeichnis deren WSDL-Datei so modifiziert, dass die physikalische Adresse des eigentlichen Web Service (z. B. `http://www.rating-company-a.de/webservices/ws-a`) durch die Adresse des WSProxy (z. B. `http://192.168.0.10:8081/creditprocess/rating-a`) ersetzt wird. Diese Modifizierung der physikalischen Web Service Adresse wird durch die SLA-Management-Komponente vorgenommen. Für jeden Web Service wird hierbei der Netzwerkendpunkt des WSProxy unterschiedlich parametrisiert (z. B. `http://192.168.0.10:8081/creditprocess/rating-a` oder `http://192.168.0.10:8081/creditprocess/rating-b`). Ein solcher parametrisierter Netzwerkendpunkt wird im Folgenden als *Request-URL* bezeichnet. Alle auf diese Weise umkonfigurierten Web Service Aufrufe werden nun automatisch an den WSProxy gesendet, anstatt sie direkt an den vormals adressierten Web Service zu senden. Die Proxy-Komponente des WSProxy empfängt die Web Service Aufrufe und erkennt anhand der übermittelten Request-URL, welcher Web Service ursprünglich aufgerufen werden sollte. Die

Auswahl-Komponente ermittelt nun durch Abfragen der Routing-Tabelle, ob statisch ein bestimmter Web Service aufgerufen werden soll oder ob zur Laufzeit dynamisch ein Web Service bestimmt wird. Im ersten Fall wird der entsprechende Web Service direkt aufgerufen. Ist hingegen eine dynamische Auswahl gewünscht, wird derjenige Web Service aufgerufen, der über die höchste Bewertung innerhalb der entsprechenden Kategorie verfügt.

Für aufgerufene Web Services ist der Umweg der Aufrufe über den WSProxy vollkommen transparent, d. h., sie erhalten Aufrufe auf dieselbe Art und Weise wie zuvor und antworten auf diese, ohne dass dadurch sich für sie Änderungen ergeben. Der WSProxy fungiert bei einem Web Service Aufruf als Zwischeninstanz zwischen der Workflow-Engine und dem jeweiligen Web Service und übernimmt das *Routing* des Web Service Aufrufs. Der Ablauf des Aufrufvorgangs wird durch das Sequenzdiagramm in Abbildung 23 veranschaulicht.

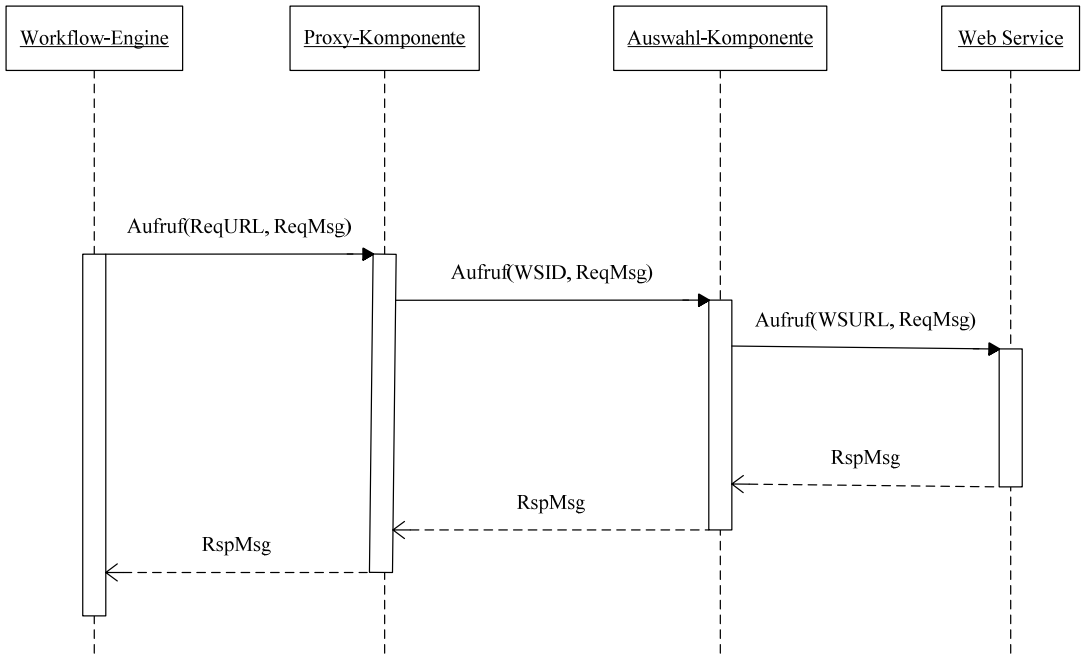


Abbildung 23: Aufruf eines Web Service über WSProxy

Zum Aufruf eines Web Service generiert die Workflow-Engine eine Nachricht `ReqMsg` und sendet diese über die Request-URL `ReqURL` an die Proxy-Komponente des WSProxy. Die Request-URL enthält im parametrisierten Teil Informationen über den aufzurufenden Web Service. In der vom WSProxy verwalteten Datenbank ist für jeden Web Service eine Routing-Methode hinterlegt, mit der die Auswahl-Komponente eine URL (`WSURL`) zum Aufruf des Web Service bestimmt. An diese URL wird die `ReqMsg` gesendet. Der so aufgerufene Web Service sendet eine Antwortnachricht (`RspMsg`) über die Auswahl-Komponente an die Proxy-Komponente zurück. Diese leitet die Antwortnachricht wiederum an die Workflow-Engine weiter.

Aufbau der Request-URL

Das Routing des WSProxy wird durch einen Teil der URL bestimmt, die zum Aufruf des WSProxy verwendet wurde. Da der WSProxy nur über das Protokoll HTTP angesprochen

werden kann, handelt es sich bei diesen URLs ausschließlich um URIs nach dem im RFC 2626 [44] spezifizierten Format:

HTTP-URL = `http://<host>[:<port>][<abs_path>[?<query>]]`

Die Basis-URL zur Adressierung des WSProxy umfasst dabei den Teil `http://<host>:<port>`. In der Basis-URL wird das Protokoll festgelegt, mit welchem der WSProxy angesprochen wird, der Rechner, auf dem der WSProxy als Server ausgeführt wird sowie der Port, unter dem der WSProxy TCP-Verbindungen annimmt. Diese Angaben sind für die Adressierung und den Aufbau einer HTTP-Verbindung zum WSProxy zwingend erforderlich. Alle weiteren Teile, die eine HTTP-URL enthalten kann, wie Pfad- und Parameterangaben, werden beim Aufruf des WSProxy im HTTP-Header übergeben und stehen dem WSProxy zur Weiterverarbeitung zur Verfügung. Derjenige Teil der URL, welcher über die Basis-URL hinausgeht, wird im Folgenden als *Web Service Identifier (WSID)* bezeichnet. Das Schema für URLs zum Aufruf von Web Services über den WSProxy lautet:

Request-URL = `http://<host>:<port><WSID>`

Ein Beispiel: Der WSProxy wurde auf einem Rechner, der im Netzwerk unter der IP-Adresse 192.168.0.10 erreichbar ist, in seinen Standard-Einstellungen installiert und ist daher unter Port 8081 erreichbar. Die Basis-URL des WSProxy lautet in diesem Fall `http://192.168.0.10:8081`. Ruft ein Client einen Web Service mittels der URL `http://192.168.0.10:8081/creditprocess/rating-a` über den WSProxy auf, so interpretiert WSProxy den Teil `/creditprocess/rating-a` als WSID.

Statischer Aufruf und dynamische Weiterleitung

Mittels eines WSIDs teilt der aufrufende Client dem WSProxy mit, welchen Web Service er aufrufen möchte, jedoch nicht, über welche URL dieser Web Service erreicht werden kann. Wie der WSProxy zu einem WSID eine entsprechende URL ermitteln soll, wird über ein *Target* definiert, das dem WSID über einen *Unique Service Identifier (USID)* zugeordnet wird. Ein USID ist in diesem Zusammenhang ein eindeutiger Bezeichner für einen Web Service. Jedem USID wird ein Target zugeordnet, das aus einem Suchverfahren und einem Suchparameter besteht. Um zur Laufzeit die URL eines geeigneten Web Service zur Weiterleitung des Web Service Aufrufs zu bestimmen, wird die im Target spezifizierte Suchmethode mit dem Suchparameter ausgeführt. Die Syntax zur Angabe eines Targets ist:

Target = `<Suchverfahren>:<Suchparameter>`

Als Suchverfahren können *static* und *portal* angegeben werden, wobei die Suchparameter spezifisch für jedes Suchverfahren sind. Das Suchverfahren *static* wird verwendet, um statische Weiterleitungen von Web Service Aufrufen an bestimmte Web Services zu realisieren. Beim Suchverfahren *portal* wird als Suchparameter die Kategorie eines Web Service angegeben. Zur Laufzeit ermittelt das Suchverfahren dynamisch denjenigen Web Service aus der angegebenen Kategorie, der zur Zeit der Suche in dieser Kategorie die höchste Bewertung besitzt. Die URL des ermittelten Web Services wird als Ergebnis der Suche zurückgeliefert.

Die Konfiguration des Routings wird über das Administrations-Frontend (vgl. Anhang, Abschnitt A.3) vorgenommen. Im Beispiel wird dem WSID `/creditprocess/rating-a` das USID `creditprocess.rating-a` zugeordnet. Dem USID `creditprocess.rating-a` wird wiederum das Target `portal:rating-a` zugewiesen. Durch diese beiden Zuordnungen ergibt sich eine Zuordnung des WSIDs `/creditprocess/rating-a` zum Target `portal:Rating-a`. Ist die Basis-URL des WSProxy-Servers beispielsweise `http://192.168.0.10:8081` und die Workflow-Engine sendet die SOAP-Nachricht eines Web Service Aufrufs an die URL `http://192.168.0.10:8081/creditprocess/rating-a`, so ermittelt der WSProxy den Web Service in der Kategorie „Rating“, der zu diesem Zeitpunkt die höchste Bewertung besitzt und sendet den Web Service Aufruf an die URL des ermittelten Web Service. Nach Erhalt der Antwort des Web Service durch den WSProxy wird diese an die Workflow-Engine zurückgegeben.

Dem WSID `/creditprocess/rating-a` wird im Beispiel über den USID `creditprocess.rating-a` das Target `static:http://192.168.0.10:8080/axis/services/rating-a` zugeordnet. Bei einem Web Service Aufruf an die URL `http://192.168.0.10:8081/creditprocess/rating-a` erfolgt daher eine Weiterleitung des Aufrufs an die URL `http://192.168.0.10:8080/axis/services/rating-a`.

3.4.5 Monitoring und Accounting

Während des Aufrufs eines Web Service durch den WSProxy wird dessen tatsächliche Dienstgüte gemessen bzw. anhand von Logging-Informationen berechnet. Die Ergebnisse werden einzeln für jeden Aufruf in einer *Call-History* gespeichert. Zusätzlich wird zu jedem Web Service eine *History-Statistik* geführt, in der alle Informationen über das bisherige Dienstgüteverhalten in Kennzahlen zusammengefasst werden. Die QoS-Monitoring-Komponente sammelt zu jedem Web Service Informationen über die Dienstgütekriterien, wie z. B. Antwortzeit, Verfügbarkeit und Durchsatz.

Bevor der WSProxy die Aufrufnachricht an einen Web Service sendet, wird die Startzeit des Aufrufs protokolliert. Nachdem die Antwort des Web Service vollständig empfangen wurde, wird die Endzeit des Aufrufs protokolliert. Aus der Differenz zwischen Start- und Endzeit wird die Dauer des Aufrufs, und damit die Antwortzeit des Web Service, berechnet. Die ermittelte Antwortzeit wird in der Call-History gespeichert und die durchschnittliche Antwortzeit in der History-Statistik aktualisiert.

Bei jedem Aufruf eines Web Service, für den eine Antwortzeit ermittelt werden kann, wird von der Verfügbarkeit des Web Service ausgegangen, da in diesem Fall die Aufrufnachricht erfolgreich an den Web Service gesendet und eine Antwort empfangen werden konnte. Ein Web Service wird als nicht verfügbar angesehen, falls die Aufrufnachricht nicht an den Web Service übermittelt werden konnte, weil z. B. keine HTTP-Verbindung zum Web Service aufgebaut werden konnte. Kommt es während der Übermittlung der Aufrufnachricht oder während des Empfangs der Antwort zu einem Abbruch der HTTP-Verbindung, gilt der Web Ser-

vice ebenfalls als nicht verfügbar. Bricht die HTTP-Verbindung nicht ab, es trifft jedoch innerhalb der doppelten, in der SLA garantierten Antwortzeit keine Antwort vom Web Service ein, gilt der Web Service ebenso als nicht verfügbar. Die Verfügbarkeit eines Web Service ist unabhängig davon, ob die Übermittlung der Antwort des Web Service an den Aufrufer fehlschlägt. Nachdem der Web Service erfolgreich aufgerufen wurde, ist es für dessen Verfügbarkeit nicht relevant, ob die weitere Verarbeitung seiner Antwort durch den WSProxy oder den Aufrufer fehlschlägt. Nachdem feststeht, ob ein Web Service bei einem Aufruf als verfügbar oder nicht verfügbar angesehen werden kann, wird die Verfügbarkeit in der Call-History gespeichert und die durchschnittliche Verfügbarkeit des Web Service in der History-Statistik aktualisiert. Die durchschnittliche Verfügbarkeit wird hierbei als prozentualer Anteil der Aufrufe des Web Service, bei denen der Web Service als verfügbar angesehen wurde, zu der Anzahl aller Aufrufe des Web Service berechnet. Jeder Aufruf eines Web Service wird in der Call-History protokolliert. Anhand der Angabe zur Verfügbarkeit können erfolgreiche Web Service Aufrufe identifiziert werden. Im Anschluss an jeden erfolgreichen Aufruf wird die Call-History nach weiteren erfolgreichen Aufrufen des soeben ausgeführten Web Services selektiert und nach verschiedenen zeitlichen Intervallen gruppiert. Hieraus wird der maximale bisher durch diesen Web Service erzeugte Durchsatz pro Monat, Tag und Stunde ermittelt und in der History-Statistik des Web Service aktualisiert.

Das QoS-Monitoring beschränkt sich nicht nur auf die Messung von Dienstgüteparametern, sondern schließt auch das Warnen des Nutzers bei kritischen Werten ein. Zu jeder Warnung werden der genaue Zeitpunkt ihres Auftretens, ein Warnungstyp sowie eine Warnmeldung festgehalten. Es lassen sich die drei Warnungstypen „Nicht-Verfügbarkeit“, „SLA-Verletzung“ und „SLA abgelaufen“ unterscheiden.

Logging und Accounting

Während der Abarbeitung eines Aufrufes durch den WSProxy werden neben der Messung der Dienstgüte durch die QoS-Monitoring-Komponente und der eventuellen Ausgabe von Warnungen eine Vielzahl weiterer Informationen in einer *Logging-Tabelle* erfasst. Die hier gesammelten Informationen werden bei dem Auftreten von Fehlern zur Analyse der Ursache verwendet und können als Datenbasis für weitere Auswertungen, z. B. im Zusammenhang mit dem Accounting, gebraucht werden. Die in der Logging-Tabelle zu jedem durch den WSProxy abzuarbeitenden Aufruf gespeicherten Informationen werden wie folgt untergliedert:

- Informationen zum Aufrufer
  Zur Initiierung eines Aufrufs über den WSProxy sendet der Aufrufer eine HTTP-Nachricht an den WSProxy. Gleichzeitig werden die eingehende HTTP-Nachricht, die IP-Adresse und der Port des Clients sowie der Aufrufzeitpunkt protokolliert.
- Informationen zur Bestimmung des Routing-Ziels
  Nachdem die HTTP-Nachricht des Aufrufers empfangen wurde, wird der WSID aus dem HTTP-Header extrahiert. Zu dem übermittelten WSID wird über den verknüpften

USID das zu verwendende Target ermittelt. Das extrahierte WSID, das verknüpfte USID und das verknüpfte Target werden protokolliert.

- Informationen zum Aufruf des Web Service
  Die ursprünglich vom Client empfangene HTTP-Nachricht wird an die ermittelte URL des Web Service gesendet. Hierbei werden die gesendete HTTP-Nachricht sowie der Sendezeitpunkt protokolliert. Nachdem eine Antwort vom aufgerufenen Web Service empfangen wurde, wird der Empfangszeitpunkt sowie die empfangene HTTP-Nachricht protokolliert. Aus der Differenz zwischen Empfangs- und Sendezeitpunkt wird zusätzlich die Antwortzeit des Web Service protokolliert.
- Informationen zur Rücksendung der Antwort an den Client
  Die vom Web Service empfangene HTTP-Nachricht wird an den Client gesendet. Neben der gesendeten HTTP-Nachricht wird der Absendezeitpunkt an den Client protokolliert.
- Generelle Informationen zur Abarbeitung des Aufrufs
  Bei jeder Änderung des Protokolleintrages wird der Zeitpunkt dieser letzten Änderung festgehalten. Spezielle Status-Codes (vgl. Anhang, Abschnitt B.6) protokollieren den momentanen Status der Abarbeitung eines Web Service Aufrufs. Dadurch ist jederzeit ersichtlich, in welcher Phase der Abarbeitung durch den WSProxy sich ein Web Service Aufruf befindet. Sollte die Verarbeitung durch das Auftreten von unvorhergesehenen Fehlern unkontrolliert abbrechen, kann anhand des Statuscodes ersehen werden, in welcher Phase die Verarbeitung unerwartet unterbrochen wurde. Wird eine HTTP-Fehlermeldung an den Client gesendet, z. B. weil das gesendete WSID unbekannt ist oder der Aufruf des Web Service durch den WSProxy fehlgeschlagen ist, wird der gesendete HTTP-Fehlercode protokolliert. Anhand des HTTP-Fehlercodes werden auch bei einer erfolgreichen Verarbeitung eines Web Service Aufrufes über alle Phasen hinweg aufgetretene Fehler im Zusammenhang mit dem aufrufenden Client oder dem aufgerufenen Web Service erkannt.
- Accounting
  Durch das Festhalten von Detailinformationen zu jedem einzelnen durch den WSProxy verarbeiteten Web Service Aufruf in der Logging-Tabelle und der Call-History entsteht eine umfangreiche Datenbasis, die zu den verschiedensten Zwecken ausgewertet werden kann, wie z. B. zur Kostenabrechnung.
- Kostenabrechnung
  Intern, d.h. im Netzwerk des Nutzers, kann ein Aufrufer eines Web Service durch seine IP-Adresse identifiziert werden. Die durch einen Aufruf des Web Service entstehenden Kosten können z. B. über ein SLA gewonnen werden. Kosten, die durch die Web Service Aufrufe entstehen, können so über identifizierte Clients in die Kostenstellenrechnung des Unternehmens integriert werden. Ist für die Ermittlung der Kosten von Web Service Aufrufen auch das hierdurch erzeugte Transfervolumen relevant, können über die in der Logging-Tabelle protokollierten, gesendeten und empfangenen

HTTP-Nachrichten die Transfervolumina im internen und externen Netz ermittelt und verrechnet werden.

Externe Anbieter, die Web Services über das WSQoSX-Portal angemeldet haben, haben Kosten für die Nutzung ihres Web Services hinterlegt. In regelmäßigen Abständen werden daher die Anbieter den Nutzer für die Inanspruchnahme der angebotenen Web Services belasten. Um die abgerechneten Beträge zu kontrollieren, werden die erfolgreichen Aufrufe des Web Service im Abrechnungszeitraum ermittelt und das zu entrichtende Nutzungsentgelt berechnet. Hieran können leicht Diskrepanzen mit den Abrechungen der Anbieter erkannt werden. Sollte der Web Service im Abrechnungszeitraum gegen die garantierte Dienstgüte verstoßen haben, so sind hier entsprechende Abzüge einzurechnen bzw. Zahlungen gänzlich zu verweigern.

3.4.6 Deaktivierung und Löschung

Aus der Sicht des Nutzers können Gründe für die Löschung eines Web Service darin bestehen, dass der Web Service nicht mehr benötigt wird, die Zusammenarbeit mit dem Anbieter eingestellt werden soll oder die Gültigkeitsdauer des Angebots abgelaufen ist. Aus der Sicht eines Anbieters kann ein Grund für die Löschung sein, dass der Web Service in Zukunft nicht mehr angeboten werden soll. Ein Anbieter kann die von ihm angebotenen Web Services über das Portal löschen. Der Nutzer verwendet zum Löschen von Web Services das Administrations-Frontend. Im Administrations-Frontend besteht für den Nutzer zusätzlich die Möglichkeit, ganze Kategorien einschließlich aller Web Services oder Anbieter inklusiver aller angebotenen Web Services zu löschen. Durch eine Löschung eines Web Service werden alle Informationen bezüglich des Web Service aus WSQoSX entfernt. Durch die Löschung eines Web Service aus einer Kategorie können sich die minimalen und maximalen Eigenschaftswerte der quantifizierbaren Eigenschaften in dieser Kategorie verändern. Daher müssen die Bewertungspunkte für die quantifizierbaren Eigenschaften durch eine erneute Normalisierung neu berechnet werden. Da durch die Änderung der Punkte die bisherige Score eines Web Service ungültig wird, ist auch die Score aller Web Services in dieser Kategorie neu zu berechnen.

In der Praxis existiert eine Vielzahl von Fällen, in denen ein Web Service nicht endgültig gelöscht, sondern nur vorübergehend deaktiviert werden soll. Ein Beispiel hierfür ist z. B. das akute Auftreten von Problemen bei der Nutzung eines Web Service. Durch eine Deaktivierung des Web Service kann die Funktionalität an einen anderen Web Service derselben Kategorie übertragen werden. Die vorübergehende Deaktivierung eines Web Service ist dem Anbieter nicht möglich, da dies eine vorübergehende Aussetzung seiner Garantie bezüglich der Dienstnutzung bedeuten würde. Ein Nutzer kann jedoch über das Administrations-Frontend Web Services vorübergehend deaktivieren. Im Gegensatz zu einer Löschung müssen bei einer Deaktivierung eines Web Service nicht die Bewertungen aller Web Services in der betroffenen Kategorie neu berechnet werden. Der deaktivierte Web Service bleibt bei seiner Deaktivierung Bestandteil einer Kategorie und wird lediglich innerhalb dieser Kategorie nicht mehr produktiv verwendet. Da sich hierdurch keine Änderung der Minima und Maxima bei den

quantifizierbaren Eigenschaften einer Kategorie ergibt, bleibt die Gültigkeit aller zuvor kalkulierten Werte in der Kategorie erhalten.

3.5 Umsetzung und Implementierung

Zu den Kernkomponenten von WSQoSX zählen der ProxyServer *WSProxy*, das Portal *WSPortal* und das Administrations-Frontend *WSProxyAdmin* (siehe Abbildung 24). Mit Ausnahme des WSProxyAdmin und einem Client zum Aufruf des Geschäftsprozesses, die in .NET entwickelt wurden, basiert die Entwicklung des WSProxy auf Java 2 Platform, Standard Edition (J2SE) in der Version 1.4.1. Als Applikations-Server bzw. Servlet-Container, wird Apache Tomcat 5.0.28 verwendet. Tomcat wird von dem Portal WSPortal, der SOAP-Engine Apache AXIS und der Workflow-Engine BPWS4J als Ausführungsumgebung genutzt. Der in BPEL4WS formulierte Geschäftsprozess wird durch BPWS4J ausgeführt, Web Services werden über AXIS bereitgestellt und WSPortal bietet durch JavaServer Pages eine Web-Schnittstelle für Anbieter von Web Services. Zur Datenhaltung verwenden die Kernkomponenten den Datenbankserver MySQL 4.1.9.

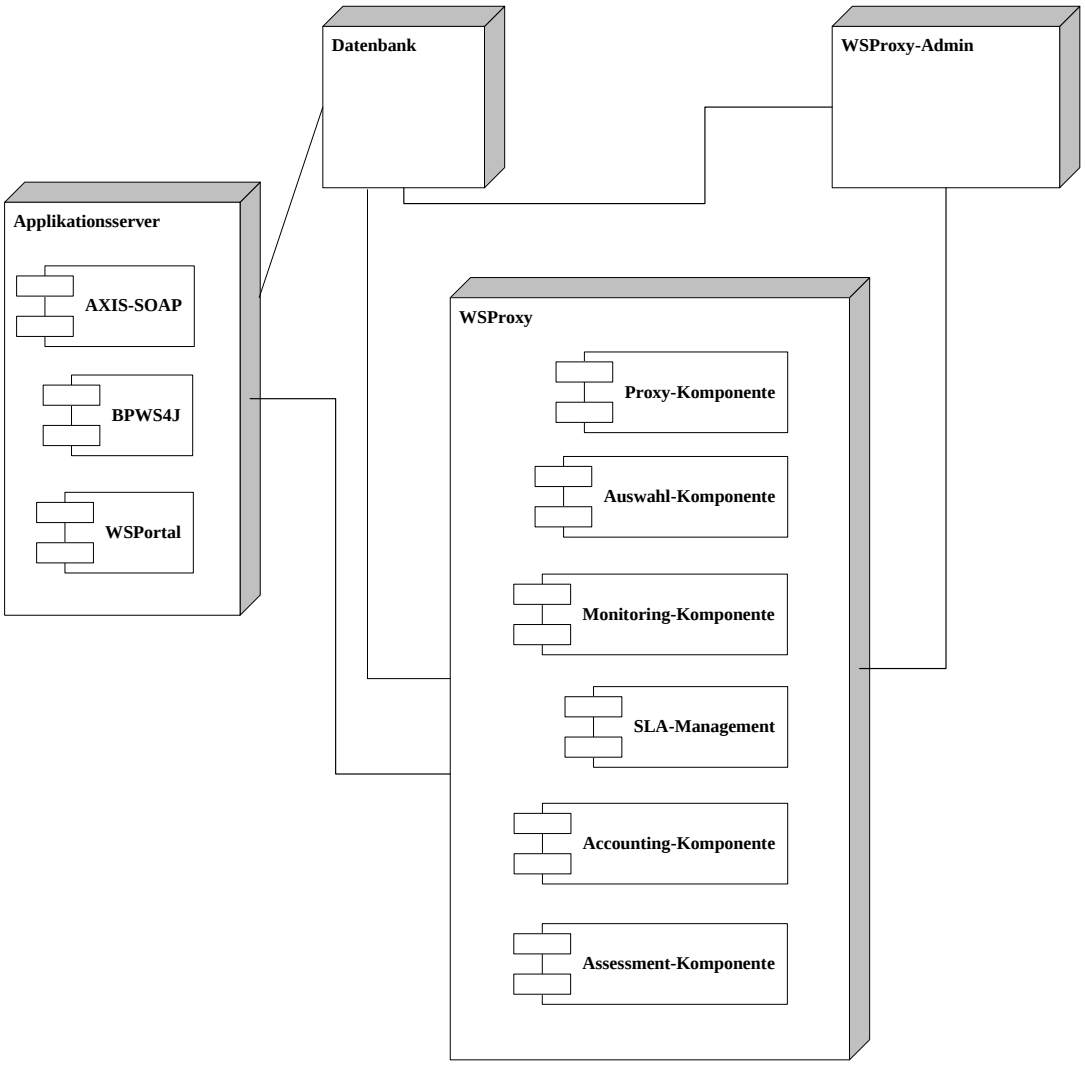


Abbildung 24: Verteilung der WSQoSX-Komponenten

WSProxy bildet den Kern von WSQoSX. Aufgabe des WSProxy ist es:

- Web Service Aufrufe zu empfangen
- den ursprünglichen gewünschten Web Service aus den übermittelten Informationen des Aufrufes zu ermitteln
- das zugeordnete Routing-Verfahren zu bestimmen
- anhand des Routing-Verfahrens den entsprechenden Web Service zu identifizieren sowie aufzurufen
- die Antwort des Web Service zu empfangen
- diese Antwort an die Workflow-Engine zurückzuliefern

Hierbei wird durch den WSProxy jeder Aufruf mitsamt den ermittelten Dienstgüteparametern protokolliert und Statistiken bezüglich der Dienstgüteparameter verwendeter Web Services erstellt und aktualisiert. Weichen die Dienstgüteparameter eines Aufrufes oder im durchschnittlichen Verhalten von den zugesicherten Dienstgüteparametern in einer SLA ab, so werden Warnungen generiert.

Im Folgenden wird ein typischer Ablauf der Verarbeitung eines Web Service Aufrufs durch den WSProxy beschrieben (siehe Abbildung 25). Hierbei werden die wichtigsten Klassen und die durch sie bereitgestellte Funktionalität näher erläutert. Eine Übersicht der Klassen kann Tabelle 15 im Anhang, Abschnitt A.1.1 entnommen werden.

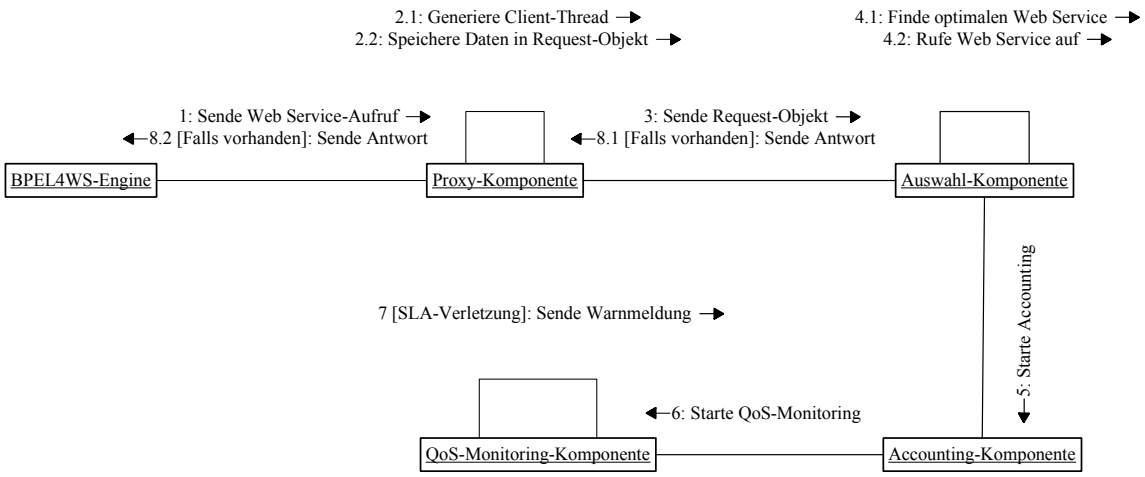


Abbildung 25: Interaktion der WSQoSX-Komponenten

Start des WSProxy

Zum Start des WSProxy wird die Klasse `de.tud.kom.dynws.proxy.WSProxy` ausgeführt. Hierbei wird ein neuer Thread der Klasse `WSProxyDaemonThread` als Daemon-Thread gestartet. Die Aufgabe des Daemon-Threads ist es, auf dem per Kommandozeilenparameter spezifizierten Port in einer Endlosschleife auf eingehende TCP-Verbindungen zu warten. Solange der Anwendungs-Thread noch nicht beendet wurde, ist der Daemon-Thread aktiv und wartet auf eingehende TCP-Verbindungen. Wird eine eingehende Verbindung festgestellt, so wird

ein neuer Thread der Klasse `WSProxyClientThread` gestartet und das Socket mit der angenommenen Verbindung an diesen Thread übergeben. Die Aufgabe des `WSProxyClientThread` ist es, die weitere Verarbeitung der angenommenen Verbindung nebenläufig zu übernehmen. Der Daemon-Thread steht direkt nach dem Start des `WSProxyClientThread` wieder für die Annahme von neuen Verbindungen zur Verfügung.

Empfang des HTTP-Requests

Durch den `WSProxyClientThread` wird die Nachricht, die ein Client über die angenommene Verbindung sendet, empfangen. Hierbei wird eine Nachricht in Form eines HTTP-Requests gemäß dem HTTP-Protokoll 1.0 [18] oder 1.1 [44] erwartet. Zunächst wird der HTTP-Header zeilenweise aus dem eingehenden Strom von ASCII-Zeichen gelesen. Es wird erwartet, dass die Nachricht wie folgt aufgebaut ist:

- 1. Zeile: `<http command> <request uri> <http version>`
- Folgende Zeilen: `<header property>: <header value>`
- Leerzeile: Separator zwischen HTTP-Header und HTTP-Body
- Alle folgenden Zeichen: HTTP-Body

Wird als HTTP-Kommando weder `GET` noch `POST` übermittelt, so wird die Verarbeitung des HTTP-Requests abgebrochen und der HTTP-Fehlercode 501 (*Not implemented*) an den Client gesendet. Wurde als HTTP-Kommando `GET` oder `POST` übermittelt, werden alle Header-Zeilen bis zum Auftreten einer Leerzeile eingelesen. Alle Zeichen nach der Leerzeile werden als Body des HTTP-Requests interpretiert und Byte für Byte binär ausgelesen. Hat der Client zuvor im Header über den `Content-Length-Header`[3] die Größe des Body in Bytes angegeben, wird der Body in einem Stück gelesen, bis die angegebene Anzahl an Bytes empfangen wurde. Wurde durch den Client zuvor im HTTP-Header über den *Transfer-Encoding-Header* das Übermittlungsverfahren *`chunked`*[4] definiert, wird der Body in mehreren, speziell markierten Teilen (*`Chunks`*) eingelesen. In diesem Fall steht die Größe des Body bis zum Erhalt des letzten *`Chunks`* nicht fest. Der empfangene HTTP-Request wird in einem Objekt der Klasse `WSProxyHTTPRequest` abgelegt.

Routing des HTTP-Requests

Zur Weiterverarbeitung des empfangenen HTTP-Requests wird eine Instanz der Klasse `WSRouter` erzeugt und daher der empfangene HTTP-Request in Form eines `WSProxyHTTPRequest`-Objekts im Konstruktor übergeben. Durch den Aufruf der Methode `route` des `WSRouter`-Objekts wird das Routing des HTTP-Requests ausgelöst. Die erste Zeile des HTTP-Requests enthält die durch den Request beim Server adressierte URI. Im Falle des WSProxy handelt es sich bei dieser URI um den WSID des adressierten Web Service. Zu dem übermittelten WSID wird nun in der Datenbank über den zugeordneten USID ein

[3] Die Verwendung des *Content-Length-Headers* entspricht der HTTP-Version 1.0.

[4] Das Übermittlungsverfahren *chunked* steht erst seit der HTTP-Version 1.1 zur Verfügung.

Target ermittelt, welches angibt, wie nach einem Ziel für den HTTP-Request gesucht werden soll. Kann ein solches Target nicht ermittelt werden, ist das im Target angegebene Suchverfahren unbekannt oder kann über das angegebene Suchverfahren keine Endpunkt-URL für die Weiterleitung ermittelt werden, wird eine `WSRouterUnroutableException` ausgelöst und an den Client der HTTP-Fehlercode 404 (*Not found*) zurückgegeben. Kann ein Target ermittelt werden, wird eine Instanz der Klasse `WSSearchFactory` erzeugt und über die Methode `getSearchForTarget(target)` ein Objekt zu diesem Target instanziiert, welches das Interface `WSSearch` implementiert. Für jedes unterstützte in einem Target spezifizierbare Suchverfahren existiert eine Klasse, welche die Schnittstelle `WSSearch` implementiert und das spezifische Suchverfahren nach einer Endpunkt-URL über die Funktion `getEndpointURL` bereitstellt (z. B. `WSSearchPortal` oder `WSSearchStatic`). Neue Suchverfahren können daher leicht über weitere Klassen, die das Interface `WSSearch` implementieren, ergänzt werden. Das Suchverfahren *static* wird durch die Klasse `WSSearchStatic` implementiert und liefert im Target die Endpunkt-URL zurück.

Das Suchverfahren *portal* wird durch die Klasse `WSSearchPortal` implementiert und durchsucht die durch den Suchparameter spezifizierte Kategorie nach dem Web Service mit der höchsten Score. Die Endpunkt-URL des gefundenen Web Services wird als Ergebnis der Suche zurückgegeben. Neben der Methode `getEndpointURL` zur Ermittlung der Endpunkt-URL stellt jede Klasse, die das Interface `WSSearch` implementiert, auch die Funktion `getResponseTime` zur Verfügung. Nach der Ermittlung der Endpunkt-URL liefert diese Funktion die zu erwartende Antwortzeit des Web Service unter der ermittelten Endpunkt-URL.

Dynamischer Aufruf von Web Services durch WSProxy

Nachdem die Endpunkt-URL zur Weiterleitung des HTTP-Requests ermittelt wurde, wird der vom Client empfangene HTTP-Request über die Funktion `call` des `WSRouter`-Objekts an diese Endpunkt-URL gesendet. Vor dem Versenden des HTTP-Requests werden die Header des Requests überprüft und ggf. angepasst. Existiert im Header des Requests ein ***`Host-Header`***, verweist dieser noch auf die Basis-URL des WSProxy. Vor dem Versenden wird daher der ***`Host-Header`*** so modifiziert, dass dieser den in der Endpunkt-URL spezifizierten Rechner adressiert. Enthielt der weiterzuleitende HTTP-Request einen Body und verwendete der Client das Übermittlungsverfahren ***`chunked`*** zu dessen Übermittlung, wird der entsprechende ***`Transfer-Encoding-Header`*** entfernt und durch einen ***`Content-Length-Header`*** ersetzt, da der `WSRouter` bei der Weiterleitung des HTTP-Requests den Body stets in einem Stück und nicht in einzelnen Chunks versendet.

Zum Versenden des HTTP-Requests an die Endpunkt-URL wird eine modifizierte Version des *Apache Jakarta Commons HTTP Client* verwendet. Dabei wurde die Klasse `org.apache.commons.httpclient.HttpMethodBase` so überarbeitet, dass ein *User-Agent-Header* nicht zwingend in den HTTP-Headern vorhanden sein muss. Die Klasse `org.apache.commons.httpclient.HttpConnection` wurde um das Wrapping aller Eingabe- und Ausgabe-Streams in ein Byte-Array erweitert. Hierdurch ist die Protokollierung

jedes Bytes möglich, das durch den HTTP-Client auf TCP-Ebene gesendet und empfangen wird. Diese wird intern für das Logging aller gesendeten und empfangenen HTTP-Nachrichten auf Byteebene verwendet. Zum Versenden des HTTP-Requests wird ein HTTP-Client als Objekt der Klasse `org.apache.commons.httpclient.HttpClient` erzeugt. Mit Methoden des Objekts wird anschließend das durch den HTTP-Request vorgegebene HTTP-Kommando `GET` oder `POST` gesetzt, die ermittelte Endpunkt-URL als Ziel des Aufrufs festgelegt, die angepassten Header des HTTP-Requests bestimmt sowie die Timeout-Zeit für den bevorstehenden Aufruf gesetzt.

Sind alle Aufrufparameter im HTTP-Client-Objekt gesetzt, wird die aktuelle Systemzeit festgehalten und der HTTP-Request durch einen Aufruf der Funktion `executeMethod` durchgeführt. Der HTTP-Request wird an die Endpunkt-URL des Web Service gesendet und die Antwort des Web Service empfangen. Nachdem die Antwort des Web Service vollständig empfangen wurde, wird erneut die Systemzeit ermittelt und aus der Differenz zur Startzeit des Aufrufs die Antwortzeit des aufgerufenen Web Services ermittelt und protokolliert.

Die vom aufgerufenen Web Service erhaltene Antwort wird vom `WSRouter` an den `WSProxyClientThread` in Form eines Byte-Array zurückgegeben und von diesem über die noch bestehende TCP-Verbindung zum Client zurückgesendet. Sollten während des Aufrufs des Web Service Fehler aufgetreten sein, wird der `WSProxyClientThread` in Form von Exception-Meldungen hiervon in Kenntnis gesetzt. Dem Client wird in diesem Fall je nach aufgetretenem Fehler eine HTTP-Fehlernachricht gesendet. Ist während des Aufrufs ein Timeout aufgetreten, wird eine `WSRouterRemoteTimeoutException` ausgelöst und der HTTP-Fehlercode 504 (*Gateway timeout*) an den Client gesendet. Ist während des Aufrufs ein Fehler bei einer Ein-/Ausgabeoperation aufgetreten, wird der HTTP-Fehlercode 502 (*Bad Gateway*) an den Client gesendet. Sollte ein anderer nicht erwarteter Fehler während des Aufrufes aufgetreten sein, wird der HTTP-Fehlercode 500 (*Internal server error*) an den Client übermittelt. Konnte die Antwort vollständig an den Client übermittelt werden, so wird die TCP-Verbindung zum Client beendet.

Nachbearbeitung eines Aufrufes und QoS-Monitoring

Ist der eigentliche Web Service Aufruf abgearbeitet, beginnt dessen Nachbearbeitung. Hierzu zählen das Eintragen der gemessenen Dienstgüteparameter in die Call-History, das Aktualisieren der Statistik des aufgerufenen Web Services in der History-Statistik und das Generieren von Warnungen, sollten Dienstgüteparameter in einem kritischen Bereich festgestellt worden sein.

Zur Nachbearbeitung des Aufrufs wird ein Objekt der Klasse `WSHistory` instantziiert und der soeben verarbeitete HTTP-Request als `WSProxyHTTPRequest`-Objekt im Konstruktor übergeben. Durch den Aufruf der Methode `update` auf dem Objekt wird die Nachbearbeitung vollständig ausgeführt. Die Generierung von Warnungen geschieht hierbei spezifisch für das Suchverfahren, welches verwendet wurde, um die Endpunkt-URL zur Weiterleitung des HTTP-Requests zu ermitteln. Das Interface `WSWarnings` definiert eine gemeinsame Schnitt-

stelle für alle Klassen, die Warnmeldungen generieren können. Innerhalb der Klasse `WSHistory` wird die Klasse `WSWarningsFactory` verwendet, um eine zum verwendeten Suchverfahren passende Implementierung von `WSWarnings` zu instanziieren (z. B. `WSWarningsPortal` und `WSWarningsStatic`). Durch den Aufruf der Methode `createWarnings` auf dem so erzeugten Objekt werden die Aufrufdaten entsprechend analysiert und, falls nötig, Warnungen generiert und in der Datenbank abgelegt.

Beendigung der Verarbeitung des HTTP-Requests

Mit dem Abschluss der Nachbearbeitung des Aufrufs ist dieser vollständig durch den `WSProxyClientThread` abgearbeitet und der Thread wird beendet. Sobald der Status eines HTTP-Requests über die Methode `setStatus` des `WSProxyHTTPRequest`-Objekts geändert wird, wird der neue Status sofort persistent in der Logging-Tabelle aktualisiert. Der Status spiegelt die Phase der Verarbeitung wieder in der sich ein HTTP-Request befindet. Eine Übersicht der Status-Codes kann Tabelle 16 im Anhang, Abschnitt A.1.3 entnommen werden.

3.6 Zusammenfassung

In diesem Kapitel wurde die Konzeption und Umsetzung von WSQoSX, einer Service-orientierte Dienstgütearchitektur zur Steuerung von Workflows, beschrieben, welche die Dienstgüte der beteiligten Services basierend auf einem in dieser Arbeit entwickelten Vorgehensmodell umfassend unterstützt. WSQoSX stellt ein Portal bereit, an dem sich auch externe Anbieter registrieren und ihre Web Services anmelden können. Diese werden aufgefordert, in einer SLA die Dienstgüte der entsprechenden Web Services und deren Nutzungsbedingungen zu definieren. Web Services gleicher Funktionalität werden in Kategorien gruppiert. Zu jeder Kategorie werden die Präferenzstruktur eines Entscheidungsträgers bzgl. verschiedener Dienstgüteeigenschaften sowie entsprechende Mindestanforderungen definiert. Erfüllt der Web Service alle Mindestanforderungen, wird eine Gesamtbewertung errechnet und der Web Service wird in die Menge der durch WSQoSX verwendeten Web Services aufgenommen. Bei der Ausführung eines durch WSQoSX gesteuerten Workflows kann dynamisch zur Laufzeit derjenige Web Service aus der entsprechenden Kategorie ausgeführt werden, der mit der definierten Präferenzstruktur des Entscheidungsträgers am besten übereinstimmt. Alle Web Service Aufrufe über WSQoSX werden protokolliert und die Einhaltung der vom Anbieter in Form von SLAs garantierten Dienstgüteeigenschaften kontrolliert.

4 Optimierung von Web Service Workflows

Die in Kapitel 3 vorgestellte Web Service Architektur WSQoSX ermöglicht u. a. die Auswahl von Web Services unter Berücksichtigung von Dienstgüteeigenschaften. In diesem Kapitel wird als konsequente Weiterentwicklung des im Rahmen von WSQoSX umgesetzten dienstgütebasierten Auswahlverfahrens die Zuordnung konkreter Web Services zu abstrakten Prozessschritten als ein *Optimierungsproblem* formuliert. Dies hat den Vorteil, dass die Präferenzen des Nutzers und seine Anforderungen explizit modelliert werden können. Ein Beispiel für die explizite Modellierung der Nutzerpräferenzen in diesem Zusammenhang kann lauten: „Selektiere Web Services mit möglichst geringen Antwortzeiten unter der Nebenbedingung, dass die durch die Nutzungen der Web Services entstehenden Kosten einen bestimmten Betrag nicht überschreiten“. Zudem wird bei der Lösung des Optimierungsproblems sichergestellt, dass diese Präferenzen und Anforderungen bezogen auf den gesamten Workflow berücksichtigt und umgesetzt werden. Dieser Ansatz ermöglicht somit die globale Optimierung eines Web Service Workflows.

Das Kapitel ist wie folgt gegliedert: Nach einer detaillierten Darstellung der Problemstellung in Abschnitt 4.1 wird in Abschnitt 4.2 eine Einführung in das Themengebiet Optimierungsmodelle gegeben. Wie die Zuordnung von Web Services zu abstrakten Prozessschritten unter Berücksichtigung der Nutzerpräferenzen und Anforderungen an einen Web Service Workflow in Form eines mathematischen Optimierungsmodells beschrieben werden kann, wird in Abschnitt 4.3 vorgestellt. Das Optimierungsproblem, einen Ausführungsplan für einen Web Service Workflow zu finden, der sowohl alle definierten Restriktionen erfüllt als auch hinsichtlich der Nutzerpräferenzen optimal ist, ist ein *NP-schweres* Optimierungsproblem [22, 168, 172]. In Abschnitt 4.4 werden heuristische Verfahren vorgestellt, die dieses NP-schwere Optimierungsproblem mit vertretbarem Rechenaufwand lösen. Die Implementierung der heuristischen Lösungsverfahren wird in Abschnitt 4.5 diskutiert. Das Laufzeitverhalten und die Lösungsgüte dieser Heuristiken werden in Abschnitt 4.6 evaluiert. Abschnitt 4.7 fasst die wesentlichen Ergebnisse dieses Kapitels zusammen.

4.1 Problemstellung

In Abschnitt 4.1.1 werden die Begrifflichkeiten *Workflow*, *Kategorie* und *Ausführungsplan* im Zusammenhang mit der Optimierung von Web Service Workflows vorgestellt. Wie der Nutzen eines Ausführungsplans bestimmt wird, ist Gegenstand von Abschnitt 4.1.2. Ein Beispielszenario wird in Abschnitt 4.1.3 beschrieben.

4.1.1 Workflow, Kategorie und Ausführungsplan

Bei der Ausführung eines Web Service Workflows ist es die Aufgabe der Workflow-Engine, jedem Prozessschritt einen konkreten Web Service zuzuordnen und auszuführen. Es wird angenommen, dass zur Ausführung eines Prozessschrittes verschiedene funktional identische Web Services, sog. *Prozessschrittkandidaten,* zur Verfügung stehen, die sich jedoch in ihren nicht-funktionalen Eigenschaften unterscheiden. Funktional identische Web Services werden in *Kategorien* gruppiert. Durch die Zuordnung konkreter Web Services zu den abstrakten Prozessschritten des Workflow Modells entsteht ein *Ausführungsplan* (siehe Abbildung 26).

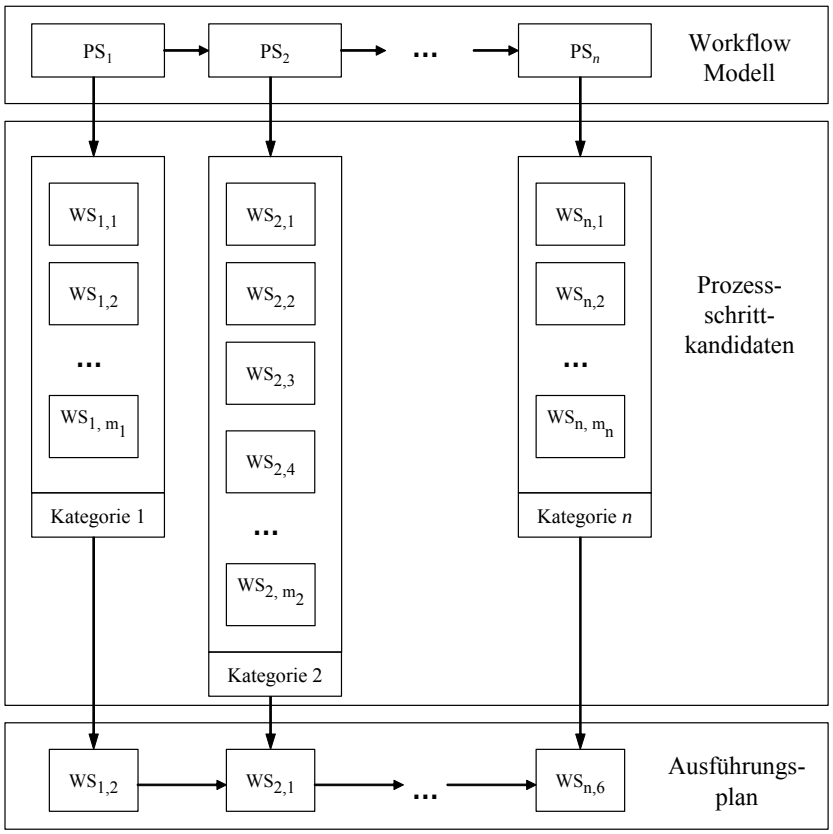


Abbildung 26: Beispiel für die Zuordnung von Prozessschritten zu Kategorien

Zur Differenzierung können diese Web Services durch eine beliebige Anzahl von nicht-funktionalen Eigenschaften beschrieben werden, wie z. B. Dienstgütekriterien oder Kosten der Servicenutzung. Jede dieser Eigenschaften wird durch einen Parameter und einen numerischen Parameterwert dargestellt.

Das Ziel ist es, bei der Ausführung des Workflows für jeden Prozessschritt aus der entsprechenden Kategorie einen Web Service so zu wählen, dass der gesamte Workflow bestimmte Eigenschaften hinsichtlich seiner nicht-funktionalen Eigenschaften erfüllt, wie z. B.: „Die Gesamtantwortzeit muss kleiner sein als zwei Minuten“. Zugleich wird der durch den Ausführungsplan gestiftete *Nutzen*, der die Präferenzen des Entscheidungsträgers reflektiert, z. B. „Minimiere die Gesamtkosten, die durch die Servicenutzung entstehen“, maximiert (vgl. Abschnitt 4.1.2).

Um die Komplexität auf ein handhabbares Maß zu reduzieren, werden nur solche Workflows betrachtet, die sich mit dem Kontrollkonstrukt der sequenziellen Ausführung definieren lassen. Ein sequenzieller Workflow besteht aus einer endlichen Anzahl von Prozessschritten, die in einer fest definierten Reihenfolge hintereinander ausgeführt werden. Diese Einschränkung ist auch insofern gerechtfertigt, da die gängigen komplexen Workflowstrukturen (z. B. Verzweigungen und Schleifen) auf sequenzielle Workflows reduziert werden können. Hierfür geeignete Verfahren werden u. a. in den Arbeiten von [169] vorgestellt. Enthält ein Workflow z. B. Verzweigungen, so wird der Pfad optimiert, der in der Vergangenheit am häufigsten ausgeführt wurde.

Aus einem Ausführungsplan, der die konkret zur Ausführung eines Workflows aufzurufenden Web Services festlegt, lassen sich die Parameterwerte berechnen, die für die Ausführung des Workflows als Ganzes gelten. Um diese Eigenschaften eines konkreten Ausführungsplans für einen Workflow zu ermitteln, werden die Parameterwerte der einzelnen verwendeten Web Services zu entsprechenden *Gesamtparameterwerten* aggregiert. Hierbei können drei Typen von Parametern unterschieden werden, die sich jeweils in der Art der Aggregation der Einzelparameterwerte zum Gesamtparameterwert unterscheiden:

- Additive Parameter
- Multiplikative Parameter
- Minimaloperator-Parameter

Additive Parameter werden durch Summierung aggregiert (z. B. Summierung der Antwortzeiten der einzelnen Web Services zur Gesamtantwortzeit), *multiplikative Parameter* durch Multiplikation (z. B. Multiplikation der einzelnen Verfügbarkeiten stochastisch unabhängiger Web Services zur Gesamtverfügbarkeit) und *Minimaloperator-Parameter* durch Anwendung des Minimaloperators (z. B. Maximaler Gesamtdurchsatz als Minimum des maximalen Durchsatzes aller verwendeten Web Services).

4.1.2 Nutzen eines Ausführungsplans

Für einen Workflow und eine gegebene Menge an Prozessschrittkandidaten existiert eine Vielzahl möglicher Ausführungspläne. Im Zuge der Optimierung muss die Workflow-Engine aus allen diesen möglichen Ausführungsplänen den optimalen Ausführungsplan ermitteln. Um zu definieren, welcher Ausführungsplan als optimal anzusehen ist, wird jedem Ausführungsplan ein *Nutzen* zugewiesen, der als numerischer Wert ausgedrückt wird. Der Nutzen ist je höher, desto stärker ein Ausführungsplan zuvor definierte Präferenzen eines Entscheidungsträgers erfüllt.

Durch *Gewichte* wird definiert, wie hoch der Nutzen ist, den die Eigenschaften bzw. Gesamtparameterwerte eines Ausführungsplans stiften. Jedem Gesamtparameterwert des Ausführungsplans (z. B. Gesamtantwortzeit, Gesamtverfügbarkeit, Maximaldurchsatz) wird ein Gewicht zugeordnet, das definiert, wie viele Nutzeneinheiten durch eine Einheit des Gesamtparameterwertes gestiftet werden. Wird dem Maximaldurchsatz beispielsweise ein Gewicht von 0,1 zugewiesen, so bedeutet dies, dass eine Zunahme des Maximaldurchsatzes um eine Einheit, einen Nutzenzuwachs von 0,1 Nutzeneinheiten bewirkt. Ein Nutzenzuwachs von genau einer Nutzeneinheit wird durch einen Zuwachs von 10 Einheiten des Maximaldurchsatzes erreicht. Wird der Gesamtverfügbarkeit ein Gewicht von 50 zugewiesen, so bewirkt eine Erhöhung der Gesamtverfügbarkeit um eine Einheit einen Nutzenzuwachs um 50 Nutzeneinheiten bzw. ein Nutzenzuwachs von genau einer Nutzeneinheit wird durch einen Zuwachs von 0,02 Einheiten der Gesamtverfügbarkeit erreicht. Im geschilderten Fall stiftet eine Zunahme des Maximaldurchsatzes um 10 Einheiten und eine Zunahme der Gesamtverfügbarkeit um 0,02 Einheiten jeweils den gleichen Zusatznutzen und ist daher bzgl. der Optimierung gleichwertig. Wird ein Gesamtparameterwert mit 0 gewichtet, so stiftet er keinen Nutzen und ist

daher für die Berechnung des Nutzens und die Optimierung des Ausführungsplans unerheblich.

Zur Berechnung des Gesamtnutzens eines Ausführungsplans werden zunächst alle Gesamtparameterwerte durch Multiplikation mit ihrem Gewicht in Nutzeneinheiten transformiert und anschließend summiert. Hierbei ist zu beachten, dass Gesamtparametern, bei denen höhere Gesamtparameterwerte als schlechter angesehen werden (z. B. die Gesamtantwortzeit), ein negatives Gewicht zugeordnet wird.

Um unter allen möglichen Ausführungsplänen denjenigen zu finden, der den höchsten Nutzen stiftet und zugleich Restriktionen seitens des Nutzers bezüglich bestimmter Gesamtparameterwerte erfüllt, wird die Zuordnung von Web Services zu Prozessschritten als ein *Optimierungsproblem* modelliert. Die zu maximierende Zielfunktion dieses Optimierungsproblems ist der Nutzen eines Ausführungsplans und die Nebenbedingungen des Optimierungsproblems spiegeln die Restriktionen seitens des Entscheidungsträgers wider [16, 130].

4.1.3 Anwendungsszenario

Ein sequenzieller Workflow besteht aus drei Prozessschritten. Für die Ausführung jedes Prozessschrittes stehen in der jeweiligen Kategorie i jeweils drei funktional identische Web Services $WS_{i,j}$ zur Verfügung, welche sich in ihren nicht-funktionalen Eigenschaften $p_{i,j,k}$ wie folgt unterscheiden (siehe Tabelle 4):

	Web Service $WS_{i,j}$		
Kategorie $i = 1$	$j = 1$	$j = 2$	$j = 3$
Antwortzeit $p_{1,j,1}$ [ms]	1.000	1.500	500
Verfügbarkeit $p_{1,j,2}$ [%]	99,50	99,00	99,90
Max. Durchsatz $p_{1,j,3}$ [Aufrufe/ Min]	2.500	5.000	1.000

	Web Service $WS_{i,j}$		
Kategorie $i = 2$	$j = 1$	$j = 2$	$j = 3$
Antwortzeit $p_{2,j,1}$ [ms]	250	500	750
Verfügbarkeit $p_{2,j,2}$ [%]	99,90	99,60	99,10
Max. Durchsatz $p_{2,j,3}$ [Aufrufe/ Min]	1.250	1.500	2.000

	Web Service $WS_{i,j}$		
Kategorie $i = 3$	$j = 1$	$j = 2$	$j = 3$
Antwortzeit $p_{3,j,1}$ [ms]	3.000	2.000	4.000
Verfügbarkeit $p_{3,j,2}$ [%]	99,95	99,99	99,90
Max. Durchsatz $p_{3,j,3}$ [Aufrufe/ Min]	4.500	3.000	6.000

Tabelle 4: Kategorien funktional identischer Web Services

Ein möglicher Ausführungsplan des Workflows kann zur Ausführung des Prozessschrittes 1 den Web Service 1 aus Kategorie 1, für Prozessschritt 2 den Web Service 3 aus Kategorie 2 und für Prozessschritt 3 den Web Service 2 aus Kategorie 3 (Kurznotation 1/1→3/2→2/3) festlegen. Die Antwortzeit ist ein additiver Parameter, weshalb sich die Gesamtantwortzeit x^+ dieses Ausführungsplans durch Addition der Antwortzeiten der gewählten Web Services berechnet als x^+ = 1.000 ms + 750 ms + 2.000 ms = 3.750 ms. Da die Verfügbarkeit ein multiplikativer Parameter ist, ergibt sich die Gesamtverfügbarkeit $x^\bullet$ des Ausführungsplans, d. h. die Wahrscheinlichkeit mit der bei einer Ausführung des Workflows jeder auszuführende Web Service verfügbar ist, durch Multiplikation der Verfügbarkeiten der gewählten Web Services: $x^\bullet$ = 0,995 * 0,991 * 0,9999 ~ 98,59%. Der Maximaldurchsatz ergibt sich aus dem Minimum der Maximaldurchsätze aller gewählten Web Services x^{min} = Min(2.500, 2.000, 3.000) = 2.000 Aufrufe pro Minute.

Wird im betrachteten Beispiel festgelegt, dass für die Optimierung des Ausführungsplans eine Verbesserung der Antwortzeit um 32 ms, eine Verbesserung der Verfügbarkeit um 0,02% und eine Verbesserung des Maximaldurchsatzes um 10 Aufrufe pro Minute äquivalent sind, so ergibt sich hieraus eine Gewichtung von w^+ = -1*1/32 = -0,03125 für die Antwortzeit, eine Gewichtung von $w^\bullet$ = 1/0,0002 = 5.000 für die Verfügbarkeit und eine Gewichtung von w^{min} = 1/10 = 0,1 für den Maximaldurchsatz. Der im Beispiel betrachtete Ausführungsplan 1/1→3/2→2/3 besitzt die Gesamtparameter x^+ = 3.750 ms, $x^\bullet$ = 98,59% und x^{min} = 2.000 Aufrufe pro Minute. Mit der angegebenen Gewichtung der Gesamtparameterwerte ergibt sich für diesen Ausführungsplan ein Nutzen *U* in Höhe von *U* = 3.750 * -0,03125 + 0,9859 * 5.000 + 2.000 * 0,1 ≈ 5.012,31 Nutzeneinheiten. Für diese Gewichtung der Gesamtparameter existiert im Beispiel kein anderer Ausführungsplan, der einen höheren Nutzen aufweist. Bei dem betrachteten Ausführungsplan handelt es sich daher um den optimalen Ausführungsplan für die durch die Gewichte ausgedrückte Präferenzstruktur.

Wird im Beispiel eine Restriktion eingeführt, welche die Gesamtantwortzeit des Workflows auf c^+ = 3.000 ms beschränkt, so verstößt der zuvor als optimal ermittelte Ausführungsplan durch seine Gesamtantwortzeit x^+ = 3.750 ms gegen diese Restriktion und ist daher nicht mehr zulässig. Unter allen für das veränderte Optimierungsproblem zulässigen Ausführungsplänen besitzt nun der neue Ausführungsplan 3/1→1/2→2/3 mit *U* = 5.003,5625 Nutzeneinheiten den höchsten Nutzen und ist damit unter der eingeführten Restriktion der optimale Ausführungsplan (Gesamtantwortzeit x^+ = 2.750 ms, Gesamtverfügbarkeit $x^\bullet$ = 99,79%, Maximaldurchsatz x^{min} = 1.000 Aufrufe pro Minute).

4.2 Optimierungsmodelle und Lösungsverfahren

In diesem Abschnitt wird ein Überblick zum Thema Optimierungsmodelle gegeben, da dies zum weiteren Verständnis der Arbeit von Bedeutung ist. Es werden *lineare Optimierungsmodelle* (siehe Abschnitt 4.2.2) und *ganzzahlig lineare Optimierungsmodelle* (siehe Abschnitt 4.2.3) vorgestellt sowie mögliche Lösungsverfahren skizziert. Heuristiken als Lösungsverfah-

ren, die vor allem im Zusammenhang von NP-schweren Optimierungsproblemen zum Einsatz kommen, werden in Abschnitt 4.2.4 diskutiert. In Abschnitt 4.2.5 wird mit *Simulated Annealing* eine Metaheuristik vorgestellt, die im weiteren Verlauf dieser Arbeit verwendet wird.

4.2.1 Einführung

Generell werden durch den Begriff *Optimierungsproblem* alle jene Entscheidungsprobleme bezeichnet, bei denen die im Hinblick auf die zu verfolgenden Ziele die günstigste realisierbare Lösung auszuwählen ist. Ein *Optimierungsmodell* ist die formale Darstellung eines Optimierungsproblems, z. B. [36, 76]. Es enthält neben allen bei der Entscheidungsfindung zu berücksichtigenden Ursache-Wirkungs-Zusammenhängen auch Zielrelationen zur Bewertung und Auswahl von Handlungsmöglichkeiten. Um Optimierungsprobleme effizient lösen zu können, erfolgt die Darstellung des entsprechenden Optimierungsmodells in Form eines quantitativen (mathematischen) Modells, in welchem sämtliche im Modell abgebildeten Aspekte des realen Entscheidungsproblems durch kardinal messbare Größen beschrieben werden. Elemente des realen Systems werden durch Variablen dargestellt und in Form von Gleichungen oder Ungleichungen auf strukturerhaltende Weise miteinander verknüpft.

4.2.2 Lineare Optimierungsmodelle

Ein lineares Optimierungsmodell besitzt eine lineare Zielfunktion und lineare Nebenbedingungen. Die Variablen dürfen reelle Werte annehmen. Ein Optimierungsproblem, das durch ein lineares Optimierungsmodell abgebildet wird, wird als lineares Optimierungsproblem (LOP) bezeichnet. Jedes LOP lässt sich hierbei in die folgende Form bringen:

$$\text{Maximiere } F(x_1,...,x_n) = \sum_{j=1}^{n} w_j x_j \tag{5}$$

unter den Nebenbedingungen

$$\sum_{j=1}^{n} b_{i,j} x_j \le c_i \qquad \forall i = 1,...,m \tag{6}$$

$$x_j \ge 0 \qquad \forall j = 1,...,n \tag{7}$$

In Matrixschreibweise lässt sich das LOP wie folgt beschreiben:

$$\text{Maximiere } F(\vec{x}) = \vec{w}^T \vec{x} \tag{8}$$

unter den Nebenbedingungen

$$B\vec{x} \le \vec{c} \tag{9}$$

$$\vec{x} \ge \vec{0} \tag{10}$$

Hierbei sind $\vec{w}$ und $\vec{x}$ n-dimensionale Vektoren, $\vec{c}$ ist ein m-dimensionaler Vektor und B eine (m x n)-Matrix.

Für LOPe existiert mit dem *Simplex-Algorithmus* ein universelles Lösungsverfahren, z. B. [36, 76]. Der Simplex-Algorithmus löst ein lineares Optimierungsproblem nach endlich vielen Schritten exakt oder stellt die Unlösbarkeit des Problems fest.

4.2.3 Ganzzahlige lineare Optimierungsmodelle

Ein ganzzahliges lineares Optimierungsmodell ist ein lineares Optimierungsmodell, in dem mindestens eine Variable nur ganzzahlige Werte annehmen darf. Werden nur einige Variablen auf den Wertebereich der ganzen Zahlen beschränkt, so spricht man auch von gemischt ganzzahligen linearen Optimierungsmodellen. Das zugehörige Optimierungsproblem wird als (gemischt) ganzzahliges lineares Optimierungsproblem (GLOP) bezeichnet. Wird ein GLOP durch Fallenlassen der Ganzzahligkeitsrestriktion in ein LOP überführt, spricht man von einer *LP-Relaxation*.

GLOPen können nicht mit dem Simplex-Algorithmus gelöst werden, stattdessen kommen u. a. *Entscheidungsbaumverfahren* zum Einsatz. Diese führen eine möglichst begrenzte Enumeration des Lösungsraumes durch, wobei versucht wird, so früh wie möglich größtmögliche Teilmengen des Lösungsraums zu identifizieren, die nicht mehr zu einem optimalen Ergebnis führen. Diese werden dann von der weiteren Enumeration des Lösungsraums ausgeschlossen. Ein universell einsetzbares Entscheidungsbaumverfahren ist *Branch and Bound (B&B)* [36, 76]. Das *Branching* zerlegt hierbei ein zu lösendes Problem P_0 vollständig in k Teilprobleme $P_1, ..., P_k$, d. h. die Vereinigung der Lösungsmengen der Teilprobleme ergibt wieder die Lösungsmenge des Ausgangsproblems P_0. Die Zerlegung in Teilprobleme ist möglichst so vorzunehmen, dass eine vollständig disjunkte Zerlegung erreicht wird, d. h., dass der paarweise Schnitt der Lösungsmengen aller Teilprobleme leer ist. Die Teilprobleme $P_1, ..., P_k$ lassen sich analog zu diesem Verfahrens wiederum zerlegen, wodurch ein (Lösungs-)Baum des Problems mit der Wurzel P_0 entsteht [36].

$$X(P_0) = \bigcup_{i=1}^{k} X(P_i) \tag{11}$$

$$X(P_i) \bigcap X(P_j) = \varnothing \quad \forall i \neq j \tag{12}$$

Das *Bounding* beschreibt ein Prinzip zur Beschränkung des Verzweigungsprozesses des Branchings im Rahmen des Lösungsprozesses. Hierzu werden Schranken für Zielfunktionswerte berechnet, mit deren Hilfe entschieden werden kann, ob Teilprobleme verzweigt werden müssen oder nicht. Zu Beginn des Verfahrens ist die untere Schranke $\underline{F}$ des Zielfunktionswertes des Problems P_0 unbekannt und wird daher auf $-\infty$ gesetzt. Nun wird das Problem P_0 durch das Branching zerlegt. Nach jeder Zerlegung eines Problems in Teilprobleme, kann für jedes Teilproblem P_i, z. B. mittels einer LP-Relaxation, ein vereinfachtes Problem P_i' formuliert werden, welches durch ein exaktes Verfahren, z. B. dem Simplex-Algorithmus, gelöst werden kann. Der ermittelte Zielfunktionswert der optimalen Lösung von P_i' stellt eine obere Grenze $\overline{F_i}$ für den Zielfunktionswert von P_i dar. Ein Problem P_i heißt *ausgelotet* und braucht nicht weiter betrachtet, d. h. nicht weiter verzweigt zu werden, falls einer der folgenden, sich gegenseitig ausschließenden Fälle eintritt:

- $X(P_i') = \varnothing$. Das Teilproblem Pi' besitzt keine zulässige Lösung
- $\overline{F_i} \leq \underline{F}$: Die optimale Lösung des Teilproblems P_i' ist schlechter als die bisher beste bekannte Lösung des Gesamtproblems
- $\overline{F_i} > \underline{F}$: Die optimale Lösung des Teilproblems P_i' ist zulässig für P_i. Da die optimale Lösung von P_i' auch zulässig für P_0 ist und da durch $\overline{F_i} > \underline{F}$ diese Lösung zugleich auch besser als die bisher beste bekannte Lösung für P_0 ist, wird $\underline{F}$ auf $\overline{F_i}$ gesetzt.

Ein Teilproblem wird weiter zerlegt, falls es nicht auslotbar ist. Das Verfahren endet, sobald alle Teilprobleme ausgelotet sind. Die exakte optimale Lösung des Ursprungsproblems P_0 ist die Lösung des Teilproblems P_i', die im Verlauf des Verfahrens zur letzten Anpassung der unteren Schranke $\underline{F}$ führt. Wird die untere Schranke $\underline{F}$ im Verlauf des Verfahrens nie angepasst, ist das Ursprungsproblem nicht lösbar.

B&B ist ein *Metasuchverfahren*, da es lediglich ein Prinzip zur Reduzierung des effektiv durchsuchten Problemlösungsraumes beschreibt. B&B legt nicht explizit fest, wie ein Problem in Teilprobleme zerlegt wird, wie eine Relaxation der Teilprobleme erfolgt und wie die relaxierten Teilprobleme gelöst werden. Die Effizienz eines konkreten B&B-Verfahrens hängt davon ab, ob es durch ein geschicktes Branching gelingt, möglichst viele und große Teilprobleme vorzeitig auszuloten. Da das Ausloten von Teilproblemen auf einen Vergleich von Schranken zurückgeführt wird, hängt die Effizienz auch von der Qualität des Bounding-Verfahrens ab.

Die Anzahl der Lösungen eines GLOPs steigt aufgrund der Kombinatorik in der Regel exponentiell mit der Problemgröße. Dieses exponentielle Anwachsen der Lösungsmenge führt dazu, dass die meisten GLOPe NP-schwere Probleme sind. In der Praxis auftretende GLOPe werden aufgrund ihrer Größe und Problemkomplexität daher oft nicht exakt, sondern nur näherungsweise mittels Heuristiken gelöst.

4.2.4 Heuristiken

Im Hinblick auf die meist sehr hohen Rechenzeiten, die von exakten Lösungsverfahren gerade bei der Lösung von NP-schweren Problemen benötigt werden, muss man sich in der Praxis oftmals damit zufriedengeben, statt optimaler Lösungen lediglich gute zulässige Lösungen zu bestimmen. Zu diesem Zweck entwickelte Verfahren werden heuristische Verfahren, oder kurz *Heuristiken*, genannt [36, 76]. Sie bieten keine Garantie, dass ein Optimum gefunden bzw. als solches erkannt wird. Dem Verlust an Genauigkeit steht meist ein erheblicher Laufzeitvorteil im Vergleich zu exakten Lösungsmethoden gegenüber. Die Güte der erzielten Lösung ist von der Ausgestaltung des Verfahrens abhängig. Heuristiken stellen daher stets einen Kompromiss zwischen benötigter Laufzeit und erzielter Lösungsgüte dar. Heuristiken lassen sich primär unterteilen in *Eröffnungs- und Verbesserungsverfahren*.

Eröffnungsverfahren dienen der Bestimmung einer (ersten) zulässigen Lösung des betrachteten Problems. Wird eine Lösung schrittweise konstruiert und in jedem Verfahrensschritt nach

der größtmöglichen Verbesserung des Zielfunktionswertes der bisherigen Teillösung gestrebt, bezeichnet man diese Verfahren als greedy (dt.: gierig) oder myopisch (dt. kurzsichtig). Den Gegensatz dazu bilden vorausschauende Verfahren, die in jedem Schritt abschätzen, welche Auswirkungen z. B. eine Variablenfixierung auf die in nachfolgenden Schritten noch erzielbare Lösungsgüte besitzt.

Verbesserungsverfahren starten zumeist mit einer zulässigen Lösung des Problems, die entweder zufällig oder durch Anwendung eines Eröffnungsverfahrens bestimmt wird. In jeder Iteration wird von der gerade betrachteten Lösung $\vec{x}$ zu einer Lösung aus der Nachbarschaft NB($\vec{x}$) fortgeschritten. NB($\vec{x}$) enthält alle Lösungen, die sich aus $\vec{x}$ durch eine einmalige Anwendung einer Transformationsvorschrift ergeben. Typische Transformationsvorschriften verändern die Lösung meist an genau einer Stelle oder verschieben bzw. vertauschen Elemente der Lösung innerhalb ihrer Reihenfolge. Die Durchführung einer solchen Transformation wird *Zug* genannt. Die einzelnen Verfahren unterscheiden sich bzgl. der Strategien zur Untersuchung der Nachbarschaft und der Auswahl von Zügen. Reine Verbesserungsverfahren enden, sobald in einer Iteration keine verbessernde Nachbarlösung existiert bzw. gefunden werden konnte. Die beste der erhaltenen Lösungen stellt für die betrachtete Nachbarschaft ein lokales Optimum dar, deren Zielfunktionswert jedoch deutlich schlechter sein kann als der des globalen Optimums. Um das globale Optimum zu erreichen, verwendet man sog. *Metaheuristiken*. Diese steuern Verbesserungsverfahren so, dass lokale Optima verlassen werden können. Hierzu müssen Züge erlaubt werden, die zwischenzeitlich zu einer Verschlechterung des Zielfunktionswertes führen. Beispiele für derartige Metaheuristiken sind *Simulated Annealing*, z. B. [82], oder *genetische Algorithmen*, z. B. [100]. Da Simulated Annealing in dieser Arbeit Anwendung findet, wird es im folgenden Abschnitt näher beschrieben.

4.2.5 Simulated Annealing

Die Bezeichnung Simulated Annealing nimmt Bezug auf einen physikalischen Effekt, der bei der Abkühlung von Metallen auftritt. Bei hoher Temperatur bewegen sich die Atome in einem Metallstück stark hin und her. Kühlt das Metallstück ab, nimmt der Bewegungsradius der Atome ab. Tritt die Abkühlung langsam ein, ordnen sich die Atome so an, dass sie eine Anordnung mit möglichst niedrigem Energieniveau einnehmen. Tritt allerdings eine zu schnelle Abkühlung ein, haben die Atome nicht die nötige Zeit eine Anordnung mit tatsächlich minimalem Energieniveau einnehmen zu können und das Metallstück verbleibt in einem energetisch gesehen lokalen Minimum. Die Idee, zufällige Veränderungen (Atombewegungen) zuzulassen, aber deren akzeptierte Größe langsam zu reduzieren (abzukühlen), damit sich ein gewisser Zustand (globales Minimum) einstellt, liegt Simulated Annealing zugrunde.

Simulated Annealing bestimmt, ausgehend von einer zulässigen Startlösung $\vec{x}$, eine zulässige Nachbarlösung $\vec{x}'$. Führt die Nachbarlösung $\vec{x}'$ zu einer Verbesserung des Zielfunktionswertes, so wird sie als neue Lösung $\vec{x}$ akzeptiert. Führt die Nachbarlösung hingegen zu einer Verschlechterung des Zielfunktionswertes, wird sie nur mit einer bestimmten Wahrscheinlichkeit *p* als neue Lösung akzeptiert. Die Wahrscheinlichkeit *p* ist abhängig von der Höhe Δ der Verschlechterung und einem sog. Temperaturparameter λ. Je niedriger der Temperaturpa-

rameter, desto geringer wird die Wahrscheinlichkeit, mit der eine schlechtere Nachbarlösung akzeptiert wird. Im Laufe des Optimierungsprozesses geht der Temperaturparameter λ gegen Null. Dies führt zu dem Effekt, dass zu Beginn des Verfahrens schlechtere Nachbarlösungen auch bei einer hohen Verschlechterung Δ noch mit einer hohen Wahrscheinlichkeit *p* akzeptiert werden. Mit Fortschreiten des Optimierungsprozesses reduziert sich jedoch diese Wahrscheinlichkeit und es werden zunehmend nur noch solche Nachbarlösungen akzeptiert, die entweder nur zu einer geringen Verschlechterung Δ führen oder zu einer Verbesserung. Am Ende des Optimierungsprozesses werden nahezu nur noch Nachbarlösungen akzeptiert, die zu einer Verbesserung führen.

Die Funktion $p(\Delta,\lambda)$ ist hierbei so zu gestalten, dass sehr geringe Verschlechterungen mit einer sehr hohen Wahrscheinlichkeit akzeptiert werden und sehr große Verschlechterungen mit einer sehr geringen Wahrscheinlichkeit. Zudem muss die Akzeptanzwahrscheinlichkeit für Verschlechterung mit abnehmendem Temperaturparameter λ geringer werden und am Ende des Verfahrens schließlich nahezu 0 betragen. Eine Funktion, welche die genannten Anforderungen erfüllt ist, z. B. [75]:

$$p(\Delta,\lambda) = e^{-\frac{\Delta}{\lambda}}. \text{ Es gilt } \lim_{\Delta\to 0} p(\Delta,\lambda) = 1 \text{ und } \lim_{\Delta\to\infty} p(\Delta,\lambda) = 0 \tag{13}$$

Maßgeblich für die Konvergenz und die Laufzeit von Simulated Annealing ist jedoch nicht nur die Berechnung der Annahmewahrscheinlichkeit *p*, sondern auch die Art und Weise der Reduktion des Temperaturparameters λ im Laufe des Optimierungsverfahrens, d. h. die genaue Gestaltung des Abkühlungsprozesses sowie die Definition eines geeigneten Abbruchkriteriums für das Verfahren.

4.3 Mathematisches Modell

Im Folgenden wird beschrieben, wie die dienstgütebasierte Auswahl von Web Services gemäß einer gegebenen Präferenzstruktur und definierter Nebenbedingungen als Optimierungsproblem modelliert wird.

Workflow, Kategorien und Ausführungsplan

Ein sequenzieller Workflow besteht aus *n* Prozessschritten (Aktivitäten). Prozessschritt PS_i (i = 1, ..., n) wird vor Prozessschritt $PS_{i'}$ (i' = 1, ..., n) des Workflows ausgeführt, wenn $i < i'$. Prozessschritt PS_i entspricht der Position i im Ausführungsplan. Für jeden Prozessschritt PS_i stehen m_i alternative Web Services (sog. Prozessschrittkandidaten) zur Verfügung, welche die benötigte Funktionalität des Prozessschrittes erbringen können und die in der Kategorie i zusammengefasst werden. Für alle m_i Web Services j (j = 1, ..., m_i) der Kategorie i wird eine binäre Entscheidungsvariable $x_{i,j}$ eingeführt. Ist $x_{i,j} = 1$, so bedeutet dies, dass für den Prozessschritt i der j-te Web Service aus der Kategorie i verwendet wird. Ist $x_{i,j} = 0$, so wird der entsprechende Web Service nicht verwendet. Damit nur jeweils ein Web Service pro Prozessschritt ausgeführt wird, muss stets gelten:

$$\sum_{j=1}^{m_i} x_{i,j} = 1 \quad \forall i = 1,\ldots,n \tag{14}$$

Parameterwerte

Die Parameterwerte eines Web Service $WS_{i,j}$ werden mit $p_{i,j,k}$ bezeichnet, wobei k (k = 1,..., K) der Index für den Parameterwert ist. Um die Gesamtparameterwerte des Ausführungsplans zu erhalten, ist es nötig, die Parameterwerte aller zur Ausführung ausgewählten Web Services zu dem jeweiligen Gesamtparameterwert zu aggregieren. Die Art und Weise der Aggregation unterscheidet sich je nach Parametertyp und ist für additive Parameter, multiplikative Parameter und Minimaloperator-Parameter unterschiedlich im Modell abzubilden. Da die Aggregation jedoch für alle Parameter des gleichen Typs identisch ist, wird im Folgenden auf den Index k der Parameterwerte $p_{i,j,k}$ verzichtet und stattdessen der Typ des Parameters wie folgt angegeben: Parameterwerte additiver Parameter werden mit $p_{i,j}^{+}$, Parameterwerte multiplikativer Parameter mit $p_{i,j}^{\bullet}$ und Parameterwerte von Minimaloperator-Parametern mit $p_{i,j}^{\min}$ bezeichnet.

Additive Parameter

Zur Aufnahme der Gesamtparameterwerte additiver Parameter wird für jeden additiven Parameter eine zusätzliche Variable x^{+} eingeführt. Ein Gesamtparameterwert x^{+} ergibt sich durch Addition der Parameterwerte $p_{i,j}^{+}$ des jeweiligen additiven Parameters über alle zur Ausführung ausgewählten Web Services.

$$x^{+} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^{+} x_{i,j} \tag{15}$$

Um einen Gesamtparameterwert x^{+} mittels einer Restriktion zu beschränken, kann eine Nebenbedingung der Form

$$x^{+} \leq c^{+} \text{ oder } x^{+} \geq c^{+} \tag{16}$$

eingeführt werden, wobei c^{+} eine Konstante ist, die den Gesamtparameterwert x^{+} beschränkt.

Multiplikative Parameter

Zur Aufnahme der Gesamtparameterwerte multiplikativer Parameter wird für jeden multiplikativen Parameter eine zusätzliche Variable $x^{\bullet}$ eingeführt. Ein Gesamtparameterwert $x^{\bullet}$ wird durch Multiplikation der Parameterwerte $p_{i,j}^{\bullet}$ des jeweiligen multiplikativen Parameters über alle zur Ausführung ausgewählten Web Services berechnet.

$$x^{\bullet} = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^{\bullet} x_{i,j} \tag{17}$$

Ein Produkt über Entscheidungsvariablen lässt sich jedoch nicht in ein lineares Optimierungsmodell übernehmen, da dort Entscheidungsvariablen lediglich über Additions- und Subtraktionsoperationen miteinander verbunden sein dürfen. Durch eine Logarithmierung der Gleichung lässt sich jedoch das Produkt in eine Summe überführen.

$$x^{\bullet} = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^{\bullet} x_{i,j} = \prod_{i=1}^{n} \prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \tag{18}$$

$$\ln(x^{\bullet}) = \ln\left(\prod_{i=1}^{n} \prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \right) = \sum_{i=1}^{n} \left(\ln\left(\prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \right) \right) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln\left(p_{i,j}^{\bullet\, x_{i,j}} \right) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} x_{i,j} \ln\left(p_{i,j}^{\bullet} \right) \tag{19}$$

Da hierbei jedoch auch $x^{\bullet}$ logarithmiert wird, wird durch die Summe statt des Gesamtparameterwertes der logarithmierte Gesamtparameterwert berechnet. Zur Berechnung des Nutzenbeitrags, den der Gesamtparameterwert stiftet, ist es in der Zielfunktion jedoch nötig, den unlogarithmierten Gesamtparameterwert mit dem Gewicht des Gesamtparameterwertes zu multiplizieren. Den exakten unlogarithmierten Gesamtparameterwert durch eine Summe auszudrücken, wie sie im linearen Optimierungsmodell benötigt wird, ist jedoch nicht möglich.

Unter der Voraussetzung, dass alle Parameterwerte $p_{i,j}^{\bullet}$ sehr nahe an 1 sind, ist es jedoch möglich, den Gesamtparameterwert durch eine Summe anzunähern. Es gilt dann folgende Approximation [50]:

$$x^{\bullet} = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^{\bullet} x_{i,j} \approx 1 - \sum_{i=1}^{n} \left(1 - \sum_{j=1}^{m_i} p_{i,j}^{\bullet} x_{i,j} \right) \tag{20}$$

Die Verwendung dieser Approximation liegt in der Tatsache begründet, dass multiplikative Parameter im konkreten Einsatzgebiet des entwickelten Optimierungsverfahrens vor allem zur Abbildung von Wahrscheinlichkeiten verwendet werden, die nahe an 1 liegen (z. B. eine Verfügbarkeit von mehr 0,99999). In diesem Falle ergibt sich bei der Berechnung der Gesamtverfügbarkeit eines Ausführungsplans eines Workflows mit 40 Prozessschritten eine Abweichung von weniger als 10^{-4} zwischen exaktem Wert und der Approximation.

Um einen Gesamtparameterwert $x^{\bullet}$ mittels einer Restriktion zu beschränken, kann eine Nebenbedingung der Form

$$x^{\bullet} \leq c^{\bullet} \text{ oder } x^{\bullet} \geq c^{\bullet} \tag{21}$$

eingeführt werden, wobei $c^{\bullet}$ eine Konstante ist, die den Gesamtparameterwert $x^{\bullet}$ beschränkt. Diese Form der Abbildung einer Restriktion für einen multiplikativen Gesamtparameter auf eine Nebenbedingung besitzt jedoch den Nachteil, dass der Gesamtparameterwert $x^{\bullet}$ nur approximiert ist. Über die zuvor betrachtete Linearisierung eines Produkts durch eine Logarithmierung kann die Restriktion jedoch exakt in das Modell integriert werden, denn es gilt:

$$c^{\bullet} \leq \prod_{i=1}^{n}\prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \Leftrightarrow \ln\left(c^{\bullet}\right) \leq \sum_{i=1}^{n}\sum_{j=1}^{m_i} \ln\left(p_{i,j}^{\bullet}\right) x_{i,j} \tag{22}$$

$$c^{\bullet} \geq \prod_{i=1}^{n}\prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \Leftrightarrow \ln\left(c^{\bullet}\right) \geq \sum_{i=1}^{n}\sum_{j=1}^{m_i} \ln\left(p_{i,j}^{\bullet}\right) x_{i,j} \tag{23}$$

Statt Restriktionen der Form $x^{\bullet} \leq c^{\bullet}$ oder $x^{\bullet} \geq c^{\bullet}$ werden daher für multiplikative Gesamtparameter Restriktionen in der folgenden Form verwendet:

$$\ln\left(c^{\bullet}\right) \leq \sum_{i=1}^{n}\sum_{j=1}^{m_i} \ln\left(p_{i,j}^{\bullet}\right) x_{i,j} \text{ oder } \ln\left(c^{\bullet}\right) \geq \sum_{i=1}^{n}\sum_{j=1}^{m_i} \ln\left(p_{i,j}^{\bullet}\right) x_{i,j} \tag{24}$$

Minimaloperator-Parameter

Zur Aufnahme der Gesamtparameterwerte von Minimaloperator-Parametern wird für jeden dieser Parameter eine zusätzliche Variable $x^{\min}$ eingeführt. Ein Gesamtparameterwert $x^{\min}$ wird durch Anwenden des Minimaloperators auf die Parameterwerte $p_{i,j}^{\min}$ des jeweiligen Parameters aller ausgewählten Web Services berechnet.

$$x^{\min} = \underset{i=1}{\overset{n}{Min}}\left(\sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j}\right) \tag{25}$$

Um die Minimaloperation im linearen Optimierungsmodell abbilden zu können, wird für jeden Prozessschritt *i* eine Nebenbedingung eingeführt, welche besagt, dass der Gesamtparameter $x^{\min}$ höchstens so groß sein darf, wie der Parameterwert $p_{i,j}^{\min}$ des zur Ausführung des Prozessschrittes *i* gewählten Web Services.

$$x^{\min} \leq \sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j} \forall i = 1,...,n \tag{26}$$

Durch diese Nebenbedingungen ist der Gesamtparameter $x^{\min}$ nach oben durch den kleinsten Parameterwert $p_{i,j}^{\min}$ aller zur Ausführung von Prozessschritten ausgewählten Web Services beschränkt. Ein Summand der Zielfunktion des linearen Optimierungsproblems besteht aus dem Produkt des Gesamtparameters $x^{\min}$ und seinem Gewicht und gibt den Nutzenbeitrag an, der durch den Gesamtparameter $x^{\min}$ gestiftet wird. Da das Gewicht positiv ist und im Zuge der Optimierung der Nutzen maximiert wird, wird auch der Gesamtparameter $x^{\min}$ maximiert. Insgesamt nimmt daher $x^{\min}$ den größtmöglichen Wert an, der den kleinsten Parameterwert $p_{i,j}^{\min}$ aller zur Ausführung von Prozessschritten ausgewählten Web Services nicht übersteigt und ist daher exakt das Minimum der Parameterwerte $p_{i,j}^{\min}$ der zur Ausführung von Prozessschritten ausgewählten Web Services.

Um einen Gesamtparameterwert $x^{\min}$ mittels einer Restriktion nach unten zu beschränken, kann eine Nebenbedingung der Form

$$x^{\min} \geq c^{\min} \tag{27}$$

eingeführt werden, wobei $c^{\min}$ eine Konstante ist, die den Gesamtparameterwert $x^{\min}$ nach unten beschränkt.

An dieser Stelle erwähnenswert ist, dass die Variable $x^{\min}$ nicht in allen Fällen im Optimierungsmodell exakt den Gesamtparameterwert des Minimaloperator-Parameters annimmt. Wird der Gesamtparameterwert $x^{\min}$ in der Zielfunktion nicht gewichtet, wird $x^{\min}$ während der Optimierung nicht maximiert und nimmt daher nicht unbedingt das Minimum der Parameterwerte $p_{i,j}^{\min}$ der zur Ausführung von Prozessschritten ausgewählten Web Services an. Im Optimierungsmodell wird der exakte Gesamtparameterwert $x^{\min}$ jedoch lediglich in der Zielfunktion zur Berechnung des Nutzenbeitrags benötigt. Da aber der Nutzenbeitrag im Falle der Nichtgewichtung des Gesamtparameterwerts $x^{\min}$ unabhängig des Wertes von $x^{\min}$ konstant bei 0 liegt, ist der Wert des Gesamtparameterwerts $x^{\min}$ für die Zielfunktion unerheblich. Im Falle einer Restriktion, die den Gesamtparameterwert $x^{\min}$ nach unten beschränkt, wird der exakte Wert des Gesamtparameterwertes $x^{\min}$ ebenfalls nicht benötigt. Durch die Nebenbedingungen zur Realisierung des Minimaloperators wird $x^{\min}$ nach oben beschränkt und durch die Nebenbedingung zur Realisierung der Restriktion nach unten. $x^{\min}$ kann sich daher nur in den durch die Nebenbedingungen definierten oberen und unteren Schranken bewegen und sorgt hierdurch dafür, dass nur solche Lösungen im Optimierungsmodell generiert werden können, welche die definierten Restriktionen erfüllen.

Zielfunktion

Der zu maximierende Zielfunktionswert des linearen Optimierungsproblems ist der Gesamtnutzen, der durch die Gesamtparameterwerte des Ausführungsplans gestiftet wird. Der Gesamtnutzen berechnet sich als gewichtete Summe der Gesamtparameterwerte des gewählten Ausführungsplans. Die Gewichte der Gesamtparameterwerte von additiven Parametern werden mit w^{+} bezeichnet, die Gewichte der Gesamtparameter von multiplikativen Parametern mit $w^{\bullet}$ und die Gewichte der Gesamtparameterwerte von Minimaloperator-Parametern mit $w^{\min}$. Existieren im linearen Optimierungsmodell k^{+} additive Parameter, $k^{\bullet}$ multiplikative Parameter und $k^{\min}$ Minimaloperator-Parameter, ergibt sich die zu maximierende Zielfunktion $F(\vec{x})$ in Abhängigkeit vom Vektor $\vec{x}$ der Entscheidungsvariablen als:

$$F(\vec{x}) = \sum_{l=1}^{k^{+}} w_l^{+} x_l^{+} + \sum_{l=1}^{k^{\bullet}} w_l^{\bullet} x_l^{\bullet} + \sum_{l=1}^{k^{\min}} w_l^{\min} x_l^{\min} \tag{28}$$

Die Multiplikation eines Gewichts mit einem Gesamtparameterwert transformiert den Gesamtparameterwert in Nutzeneinheiten. Sind für einen Parameter höhere Parameterwerte besser, so ist das Gewicht w positiv. Sind hingegen für einen Parameter niedrigere Parameter-

werte besser, so ist das Gewicht w negativ und die Nutzeneinheiten gehen negativ in die Zielfunktion ein. Das Gewicht definiert den Zusatznutzen (bzw. Nutzenverlust), der durch eine zusätzliche Einheit des Gesamtparameterwertes erzeugt wird. Durch die Gewichte wird festgelegt, wie viele Einheiten der verschiedenen Gesamtparameter den gleichen Nutzen stiften. Hierdurch legen die Gewichte die Bedeutung der einzelnen Parameterwerte bzw. Gesamtparameterwerte während des Optimierungsvorgangs fest, d. h., sie definieren, nach welchen Kriterien der konkrete Ausführungsplan eines Workflows optimiert wird. Der Zielfunktionswert (Gesamtnutzen) entsteht als gewichtete Summe der Gesamtparameterwerte und wird während des Optimierungsvorgangs maximiert.

4.4 Heuristiken zur Optimierung von Web Service Workflows

4.4.1 Motivation

Mit zunehmender Anzahl von Prozessschritten und Prozessschrittkandidaten steigt die Anzahl möglicher Ausführungspläne schnell auf eine Menge an, deren vollständige Enumeration und Auswertung nicht mehr mit einem vertretbaren Zeitaufwand möglich ist. Wird die dienstgütebasierte Auswahl von Web Services unter Berücksichtigung von Nutzerpräferenzen als lineares Optimierungsmodell, wie in Abschnitt 4.3 vorgestellt, formuliert, können zwar exakte Methoden zur Lösung verwendet werden, allerdings ist das vorgestellte Optimierungsproblem nach Arbeiten von [22, 168, 172] NP-schwer. Diese Arbeiten haben ferner gezeigt, dass exakte Verfahren, wie z. B. Branch&Bound, einen hohen Rechenaufwand zur Lösungsfindung erfordern, der für den Einsatz zur Laufzeit in einer Workflow-Engine nicht mehr akzeptabel ist. Daher werden in diesem Abschnitt heuristische Lösungsmethoden entwickelt, die das Optimierungsproblem in angemessener Zeit lösen und deren Lösung nur geringfügig vom Optimum abweicht.

Das Eröffnungsverfahren H1_RELAX_IP (siehe Abschnitt 4.4.2) besteht aus zwei Schritten. Im 1. Schritt wird das relaxierte Problem exakt gelöst, auf dessen Grundlage im 2. Schritt durch einen Backtracking-Algorithmus eine gültige Lösung berechnet wird. Zudem werden die Verbesserungsverfahren H2_SWAP (siehe Abschnitt 4.4.3) und H3_SIMUL_ANNEAL vorgestellt (siehe Abschnitt 4.4.4).

4.4.2 Eröffnungsverfahren H1_RELAX_IP

Um einen zulässigen Ausführungsplan für einen Workflow zu konstruieren, muss an jeder Position des Ausführungsplans ein Web Service zur Ausführung der benötigten Funktionalität des entsprechenden Prozessschrittes gewählt werden. Der so entstehende Ausführungsplan muss alle definierten Restriktionen hinsichtlich seiner Gesamtparameter erfüllen und zugleich einen möglichst hohen Zielfunktionswert erreichen. Bei der Konstruktion eines solchen Ausführungsplans stellt sich die Frage, in welcher Reihenfolge Web Services an den jeweiligen Positionen des Ausführungsplans eingesetzt werden sollen, um möglichst schnell einen Ausführungsplan zu erhalten, der alle Restriktionen erfüllt und zudem einen hohen Zielfunktionswert besitzt. Wurde ein zulässiger Ausführungsplan konstruiert und der zugehörige Zielfunktionswert bestimmt, so stellt sich weiterhin die Frage, welche *Lösungsgüte* der kon-

struierte Ausführungsplan besitzt, d. h. wie nahe der durch den Ausführungsplan erreichte Zielfunktionswert bereits am Optimum, dem maximalen Zielfunktionswert, liegt.

Um einen zulässigen Ausführungsplan mit möglichst hoher Lösungsgüte konstruieren zu können, werden Abschätzungen benötigt, welche Web Services an den einzelnen Positionen des Ausführungsplans tendenziell besser geeignet sind, um eine zulässige Lösung mit hoher Lösungsgüte zu konstruieren. Zudem ist in diesem Zusammenhang wichtig zu wissen, in welcher Größenordnung sich der Zielfunktionswert der optimalen Lösung bewegt. Mithilfe dieser Informationen können bei der Konstruktion eines zulässigen Ausführungsplans gezielt zuerst solche Web Services eingesetzt werden, bei denen die Wahrscheinlichkeit sehr hoch ist, einen zulässigen Ausführungsplan zu erhalten, dessen Lösungsgüte nahe am Optimum ist. Hierdurch kann bereits nach kurzer Rechenzeit ein Ausführungsplan konstruiert werden, der sowohl zulässig ist als auch über eine hohe Lösungsgüte verfügt. Die Lösungsgüte lässt sich mittels der Information über die Größenordnung des Zielfunktionswerts der optimalen Lösung abschätzen. So wird vermieden, dass eine unnötige weitere Optimierung der Lösung unternommen wird, obwohl die Lösung bereits hinreichend gut oder gar optimal ist.

Die Heuristik H1_RELAX_IP macht sich diese Vorüberlegungen zu Nutze. Im ersten Schritt wird die LP-Relaxation des zugrunde liegenden GLOPs gelöst. Im zweiten Schritt wird basierend auf dieser Lösung mithilfe eines Backtracking-Verfahrens eine gültige Lösung konstruiert.

4.4.2.1 Schritt 1: Lösung des relaxierten Problems

Informationen darüber, welche Web Services an den einzelnen Positionen des Ausführungsplans mit einer hohen Wahrscheinlichkeit zu einer zulässigen Lösung mit möglichst hoher Lösungsgüte führen, können durch die exakte Lösung des relaxierten Problems ermittelt werden. Diese wird auch zur Abschätzung des Zielfunktionswertes der optimalen Lösung verwendet. Das relaxierte Optimierungsproblem lässt sich sehr schnell mit einem Verfahren, wie z. B. dem Simplex-Algorithmus, exakt lösen.

Durch die Relaxation ändert sich die Bedeutung der Entscheidungsvariablen $x_{i,j}$. Vor der Durchführung der Relaxation bedeutete $x_{i,j} = 1$, dass an Position i des Ausführungsplans der Web Service j der Kategorie i ausgeführt und ein Wert von $x_{i,j} = 0$, dass der entsprechende Web Service an dieser Position nicht verwendet wird. Nach der Durchführung der Relaxation können in einem Ausführungsplan, zumindest gedanklich, an einer Position mehrere Web Services anteilig ausgeführt werden. Es ist so z. B. möglich, an einer Position des Ausführungsplans einen Web Service zu 75% auszuführen und einen anderen zu 25%. In die Berechnung der Gesamtparameterwerte des Ausführungsplans gehen die Parameterwerte der anteilig ausgeführten Web Services gemäß ihrem Anteil an der Ausführung ein. Dies entspricht der Ausführung von fiktiven, nicht real existenten Web Services an den Positionen des Ausführungsplans, deren Parameterwerte durch die Linearkombination der Parameterwerte der realen Web Services gemäß ihrer anteiligen Ausführung entstehen.

Als Abschätzung der Wahrscheinlichkeit, mit welcher ein Web Service Bestandteil der optimalen Lösung des unrelaxierten Problems ist, wird der Anteil verwendet, mit der ein Web Service in der optimalen Lösung des relaxierten Problems ausgeführt wird. Im genannten Beispiel der anteiligen Ausführung eines Web Service zu 75% und eines anderen Web Service zu 25% an derselben Position, kann davon ausgegangen werden, dass der Web Service mit der anteiligen Ausführung von 75% mit einer höheren Wahrscheinlichkeit Bestandteil der optimalen Lösung des unrelaxierten Optimierungsproblems ist, als jener mit der anteiligen Ausführung von lediglich 25%.

Die optimale Lösung des relaxierten Problems erlaubt darüber hinaus eine Abschätzung des Zielfunktionswertes der optimalen Lösung des unrelaxierten Problems. Da durch die Relaxation der Ausführungsplan aus fiktiven Web Services besteht, die aus realen Web Services so konstruiert wurden, dass sie alle Restriktionen möglichst exakt erfüllen und den Zielfunktionswert maximieren, kann der Zielfunktionswert der optimalen Lösung des unrelaxierten Problems keinen höheren Wert aufweisen. Schließlich werden im unrelaxierten Problem nur reale, nicht im Sinne der Optimierung künstlich konstruierte Web Services verwendet. Diese können daher höchstens einen so hohen Zielfunktionswert erreichen, wie es den fiktiven Web Services des relaxierten Problems möglich ist. Der Zielfunktionswert der optimalen Lösung des relaxierten Problems stellt deshalb eine obere Schranke für den Zielfunktionswert der optimalen Lösung des unrelaxierten Problems dar.

Da die Höhe des Unterschiedes zwischen dem Zielfunktionswert der optimalen Lösung des relaxierten und des unrelaxierten Problems unbekannt ist, kann die Güte eines zulässigen Ausführungsplans für das unrelaxierte Problem nicht exakt beurteilt werden. Mithilfe der oberen Schranke des Zielfunktionswerts ist es jedoch möglich, eine Bedingung zur Terminierung der nachgelagerten Verbesserungsverfahren H2_SWAP und H3_SIMUL_ANNEAL zu formulieren. Unterschreitet die Abweichung zwischen dem Zielfunktionswert eines konstruierten Ausführungsplans und der oberen Schranke für den Zielfunktionswert einen gewissen Wert, z. B. 5%, so wird die Güte des konstruierten Ausführungsplans als hinreichend hoch eingestuft, sodass keine Verbesserungsverfahren mehr ausgeführt werden.

Ein weiterer Fall, in welchem unnötige Optimierungsschritte vermieden werden können, tritt beim Erkennen der Unlösbarkeit des relaxierten Optimierungsproblems auf. Existiert für das relaxierte Optimierungsproblem keine Lösung, so impliziert dies, dass für das unrelaxierte Optimierungsproblem ebenfalls keine Lösung existieren kann.

4.4.2.2 Schritt 2: Konstruktion einer zulässigen Lösung

Im ersten Schritt von H1_RELAX_IP wurde durch die Lösung des relaxierten Optimierungsproblems Informationen darüber gewonnen, welche Web Services an den einzelnen Positionen des Ausführungsplans mit einer hohen Wahrscheinlichkeit zu einer zulässigen und möglichst optimalen Lösung führen. Außerdem wurde eine obere Schranke für den optimalen Zielfunktionswert errechnet. Im zweiten Schritt wird nun ein zulässiger Ausführungsplan durch einen Backtracking-Algorithmus erzeugt.

Ein Backtracking-Algorithmus [158] ist ein Verfahren zur schrittweisen systematischen Lösungssuche, der in der Lage ist, nicht erfolgreiche Ansätze der Lösungskonstruktion zu erkennen, zu verwerfen und an einer zurückliegenden Stelle der Lösungssuche erneut mit einem alternativen Ansatz fortzufahren. Der Backtracking-Algorithmus endet mit dem Auffinden der ersten zulässigen Lösung oder dem Erkennen, dass keine zulässige Lösung existiert.

Abbildung 27 zeigt den Pseudo-Code des im Rahmen von H1_RELAX_IP verwendeten Backtracking-Verfahrens, das im Folgenden näher beschrieben wird.

Die Konstruktion eines vollständigen Ausführungsplans erfolgt durch die schrittweise Zuordnung von Web Services zu den jeweiligen Positionen des Ausführungsplans. Erfolgt die Festlegung der Web Services in den Positionen des Ausführungsplans in der Reihenfolge ihrer Ausführung, wird zunächst der Web Service für Position 1 des Ausführungsplans festgelegt. Danach wird der Web Service für Position 2, usw. zugeordnet, bis schließlich der letzten Position n ein Web Service zugewiesen wurde und damit ein vollständiger Ausführungsplan vorliegt. An jeder Position i ($i = 1, ..., n$) kann aus den m_i alternativen Web Services der Kategorie i gewählt werden.

Das Backtracking-Verfahren beginnt an Position $i = 1$ mit einem leeren Ausführungsplan, in dem noch keiner der n Positionen ein Web Service zugewiesen wurde. Da der aktuellen Position noch kein Web Service zugewiesen wurde, wird an dieser Position der erste der m_i alternativen Web Services der Kategorie i gesetzt. Nach der Festsetzung des Web Service werden die Gesamtparameterwerte des aktuellen (und bisher unvollständigen) Ausführungsplans ermittelt und auf die Einhaltung der definierten Restriktionen überprüft. Werden alle Restriktionen erfüllt, handelt es sich um einen bisher zulässigen Ausführungsplan und es wird zur nächsten Position ($i = i+1$) des Ausführungsplans fortgeschritten. An dieser Position wird wieder der erste Web Service der entsprechenden Kategorie gesetzt, die Einhaltung der Restriktionen überprüft und im Falle der Erfüllung wieder zur Position $i+1$ fortgeschritten. Gelingt es, ohne die Restriktionen zu verletzten, bis zur Position n fortzuschreiten, so wurde ein vollständiger, zulässiger Ausführungsplan erzeugt und das Backtracking-Verfahren wird beendet.

Wird jedoch bei der Prüfung der Restriktionen nach der Besetzung einer Position i festgestellt, dass mindestens eine Restriktion verletzt wird, ist es nicht nötig, weitere Positionen i' mit $i' > i$ des Ausführungsplans zu besetzen. Schließlich verletzen alle Ausführungspläne, die den aktuellen Ausführungsplan als Teillösung enthalten, ebenfalls die Restriktion(en) und sind damit keine zulässigen Ausführungspläne. Da der aktuelle Ausführungsplan nicht zu einem zulässigen, vollständigen Ausführungsplan ergänzt werden kann, muss die Entscheidung, für Position i den Web Service $WS_{i,j}$ zu verwenden, revidiert werden. An der betreffenden Position i wird dieser Web Service solange durch bisher noch nicht gewählte Web Services aus dieser Kategorie ersetzt, bis der aktuelle Ausführungsplan entweder wieder alle Restriktionen erfüllt oder alle m_i alternativen Web Services der Kategorie i erfolglos ausprobiert wurden. In letzterem Fall lässt sich durch keinen der m_i alternativen Web Services an Position i ein zulässiger Ausführungsplan erzeugen und es erfolgt ein Rückschritt zur vorherigen Posi-

tion ($i = i$-1) des Ausführungsplans. An dieser Position wird analog vorgegangen. Ist die Besetzung der Position durch einen Web Service in der Art möglich, dass alle Restriktionen erfüllt werden, wird wieder zur nächsten Position ($i = i$+1) fortgeschritten. Kann eine solche Besetzung nicht gefunden werden, erfolgt wieder ein Rückschritt zur vorhergehenden Position ($i = i$-1).

Das Verfahren endet entweder mit einem vollständigen, zulässigen Ausführungsplan an Position n oder aber an Position 1 mit einem leeren Ausführungsplan und der Erkenntnis, dass es keinen zulässigen Ausführungsplan gibt und das Optimierungsproblem daher unlösbar ist.

```
i = 1                       // Aktuelle Position im Ausführungsplan
AP = (0, 0, ..., 0)         // Anfänglicher Ausführungsplan
ende = false
while (not ende) {
       repeat {
              // Falls nicht alle Web Services in Kategorie i betrachtet
              // wurden, nächsten Web Service an Position i setzen.
              if (AP[i]<m_i) AP[i]++
       } until (Ausführungsplan AP zulässig oder AP[i] = m_i)
       if (Ausführungsplan AP unzulässig) {
              AP[i] = 0
              if (i>1)
                     i--           // Schritt zurück, falls nicht Anfang
              else
                     ende = true   // Abbruch, da keine zulässige Lösung
       } else {
              if (i<n)
                     i = i+1       // Schritt vor, falls nicht Ende
              else
                     ende = true   // Abbruch, da zulässige Lösung
       }
}
```

Abbildung 27: Pseudo-Code des Backtracking-Verfahrens

Baumorientierte Betrachtung des Verfahrens

Das Vorgehen bei der Lösungssuche durch den Backtracking-Algorithmus soll im Folgenden anhand eines Lösungsbaums verdeutlicht werden. Abbildung 28 zeigt den Lösungsbaum des Backtracking-Verfahrens für einen Workflow mit 3 Prozessschritten, wobei für Prozessschritt 1 $m_1 = 2$ alternative Web Services zu dessen Durchführung existieren, für Prozessschritt 2 $m_2 = 3$ alternative Web Services und für Prozessschritt 3 $m_3 = 2$ alternative Web Services.

Eine Ebene i im Lösungsbaum entspricht der Festsetzung eines Web Service an Position i des Ausführungsplans. Der konkret gewählte Web Service wird durch einen Knoten repräsentiert. Der Grad eines Knotens in Ebene i entspricht der Anzahl m_{i+1} der für die nächste Position i+1 verfügbaren alternativen Web Services der Kategorie i+1. Unter jedem Knoten ist der beim Erreichen des Knotens aktuelle Ausführungsplan als geordnete Menge aller bereits im Ausführungsplan gesetzten Web Services angegeben. Die Blätter entsprechen vollständigen Aus-

führungsplänen. Die Wurzel repräsentiert die Ausgangskonfiguration des Backtracking-Verfahrens mit einem leeren Ausführungsplan.

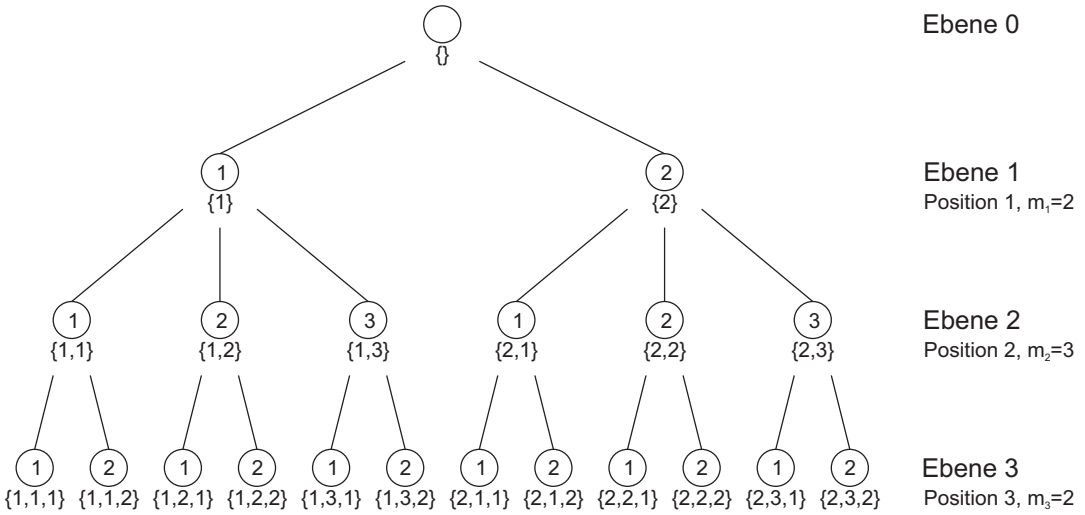


Abbildung 28: Lösungsbaum des Backtracking-Verfahrens

Dem Vorgehen des zuvor beschriebenen Backtracking-Verfahrens entspricht eine Tiefensuche im Lösungsbaum. In jedem Knoten wird der aktuelle Ausführungsplan auf die Einhaltung aller Restriktionen geprüft. Werden alle Restriktionen erfüllt, so wird zu dem Kindknoten mit dem kleinsten Index verzweigt, welcher noch nicht besucht wurde. Verletzt der aktuelle Ausführungsplan eines Knotens mindestens eine Restriktion, wird zum Vaterknoten verzweigt. Das Verfahren endet mit dem Erreichen eines Blattes, dessen Ausführungsplan alle Restriktionen erfüllt oder dem neuerlichen Erreichen der Wurzel nach dem Besuch aller Kindknoten. Im ersten Fall wurde eine zulässige Lösung gefunden, die dem aktuellen Ausführungsplan des Blattes entspricht. Im zweiten Fall existiert kein zulässiger Ausführungsplan.

Bezüglich der Laufzeit des Verfahrens stellt ein Optimierungsproblem ohne Restriktionen den günstigsten Fall dar. Da durch die Wahl eines Web Service keine Restriktion verletzt werden kann, endet das Verfahren stets nach *n* Zuweisungen von Web Services an die entsprechenden Positionen des Ausführungsplans im ersten Blatt des Lösungsbaums. In diesem Fall ist die Laufzeit linear zur Länge *n* des Workflows.

Das schlechteste Laufzeitverhalten wird im Falle eines unlösbaren Optimierungsproblems (impliziert das Vorhandensein von Restriktionen) erreicht, bei welchem die Unzulässigkeit eines Ausführungsplans erst in den Blättern des Lösungsbaums erkannt werden kann. In diesem Fall werden bei der Suche nach einer zulässigen Lösung alle Blätter des Lösungsbaums traversiert, d. h. alle möglichen vollständigen Ausführungspläne für das Optimierungsproblem erzeugt und auf Zulässigkeit überprüft. Da in jeder Position *i* des Ausführungsplans m_i alternative Web Services verwendet werden können, ergibt sich die Anzahl der besuchten Blätter bzw. die Anzahl der konstruierten und untersuchten, vollständigen Ausführungspläne als $m_1 \cdot m_2 \cdot \ldots \cdot m_n = \prod_{i=1}^{n} m_i$. Das Backtracking-Verfahren besitzt daher im schlechtesten Fall eine exponentielle Laufzeit. Daher wird im Folgenden eine Modifizierung von

H1_RELAX_IP vorgestellt, die darauf abzielt durch Vorsortierung möglichst schnell einen zulässigen Ausführungsplan mit hoher Lösungsgüte zu erhalten.

Modifizierung von H1_RELAX_IP

Die Auswahl der Web Services für die einzelnen Positionen des Ausführungsplans geschieht im geschilderten Verfahren in aufsteigender Reihenfolge ihrer Indizes in der jeweiligen Kategorie. Dieser Ansatz berücksichtigt weder, ob ein Web Service in besonderer Weise dazu geeignet ist, bestehende Restriktionen zu erfüllen noch welcher Nutzen bzw. Zielfunktionsbeitrag durch diesen Web Service gestiftet wird. Dies führt dazu, dass der konstruierte zulässige Ausführungsplan weder einen besonders hohen Zielfunktionswert aufweist noch dass er im Falle von bestehenden Restriktionen besonders schnell gefunden wird. Da jedoch gerade im Falle von bestehenden Restriktionen das Laufzeitverhalten exponentiell sein kann, ist es nötig, die benötigte Rechenzeit zur Konstruktion eines zulässigen Ausführungsplans zu reduzieren. Aus diesen Gründen ist eine Optimierung des Verfahrens in zweierlei Hinsicht erstrebenswert. Einerseits sollte nicht ein beliebiger zulässiger Ausführungsplan konstruiert werden, sondern ein Ausführungsplan, der dem optimalen möglichst nahe kommt. Andererseits sollten bei der Konstruktion des Ausführungsplans eventuell bestehende Restriktionen berücksichtigt werden, sodass durch geschickte Auswahl von Web Services möglichst schnell ein zulässiger Ausführungsplan konstruiert werden kann.

Durch die Lösung des relaxierten Optimierungsproblems im ersten Schritt der Heuristik wurden Informationen darüber gewonnen, welche Web Services an den einzelnen Positionen des Ausführungsplans mit einer höheren Wahrscheinlichkeit zu einer zulässigen und möglichst optimalen Lösung führen. Diese Informationen lassen sich nun verwenden, um eine Verbesserung des Backtracking-Verfahrens zu ermöglichen. Diese Verbesserung wird dadurch erreicht, dass sowohl die Web Services in den einzelnen Kategorien als auch die Kategorien selbst vor Anwendung des Backtracking-Verfahrens gemäß diesen Informationen sortiert werden. Dies wird im Folgenden beschrieben.

Der erste Schritt besteht in der Sortierung der Web Services in absteigender Reihenfolge ihrer anteiligen Ausführung $x_{i,j}$ in der optimalen Lösung des relaxierten Problems. Werden zwei Web Services zu gleichen Teilen ausgeführt, so werden diese nach ihrem Beitrag zur Zielfunktion absteigend sortiert (siehe Tabelle 5). Dieser Zielfunktionsbeitrag eines Web Service ist eine imaginäre Größe und berechnet sich als Zielfunktionswert eines fiktiven Ausführungsplans, der an jeder Position den Web Service ausführt, dessen Zielfunktionsbeitrag berechnet wird.

$WS_{i,j}$	$x_{i,j}$	Zielfunktionsbeitrag
$WS_{i,1}$	0,75	5
$WS_{i,2}$	0,25	3
$WS_{i,3}$	0	7
$WS_{i,4}$	0	4
$WS_{i,5}$	0	2

Tabelle 5: Sortierung der Web Services

Die anteilige Ausführung $x_{i,j}$ eines Web Service $WS_{i,j}$ wird hierbei als Abschätzung der Wahrscheinlichkeit, mit welcher ein Web Service Bestandteil der optimalen Lösung des unrelaxierten Problems ist, verwendet. Werden im Verlauf des Backtracking-Verfahrens die Positionen des Ausführungsplans in der Reihenfolge aufsteigender Indizes j mit Web Services $WS_{i,j}$ besetzt, so werden aufgrund der Sortierung zuerst solche Ausführungspläne konstruiert, die Web Services verwenden, die zu einer hohen Wahrscheinlichkeit Bestandteil des optimalen Ausführungsplans sind. Da bei der Lösung des relaxierten Problems auch evtl. bestehende Restriktionen berücksichtigt wurden, führt die Sortierung darüber hinaus auch dazu, dass zuerst solche Ausführungspläne konstruiert werden, die nicht nur zu einem hohen Zielfunktionswert führen, sondern auch in besonderer Weise dazu geeignet sind, die Restriktionen zu erfüllen. Somit gelingt es sehr schnell, einen zulässigen Ausführungsplan zu finden. Bei gleicher anteiliger Ausführung $x_{i,j}$ werden solche Web Services zuerst verwendet, deren Verwendung einen möglichst hohen Zielfunktionsbeitrag erwarten lässt.

In einem zweiten Optimierungsschritt erfolgt eine aufsteigende Sortierung der Kategorien nach der Anzahl der Web Services, die in der jeweiligen Kategorie in dem Ausführungsplan der optimalen Lösung des relaxierten Problems anteilig ausgeführt werden, d. h. nach der Anzahl enthaltener Web Services mit $x_{i,j} \neq 0$. Enthalten mehrere Kategorien die gleiche Anzahl von Web Services mit $x_{i,j} \neq 0$, werden diese absteigend nach dem maximalen Wert $x_{i,j}$ der in ihnen enthaltenen Web Services sortiert. Ein Beispiel für eine solche Sortierung kann Tabelle 6 entnommen werden. Die Sortierreihenfolge ist dort durch die Abfolge der Kategorien von links nach rechts dargestellt.

	Kategorie				
Web Service	***i = 2***	***i = 4***	***i = 5***	***i = 1***	***i = 3***
$WS_{i,1}$	1	1	0,90	0,75	0,80
$WS_{i,2}$	0	0	0,10	0,25	0,15
$WS_{i,3}$	0	0	0	0	0,05
$WS_{i,4}$	0	0	0	0	0
$WS_{i,5}$	0	0	0	0	0

Tabelle 6: Sortierung der Kategorien

Entgegen dem eingangs beschriebenen grundlegenden Backtracking-Verfahren weist das optimierte Backtracking-Verfahren den Positionen des Ausführungsplans die Web Services nicht mehr in der Reihenfolge ihrer Ausführung zu (*PS* 1, *PS* 2, ..., *PS n*), sondern in der Reihenfolge, die durch die Sortierung der Kategorien erzeugt wurde (im Beispiel: *PS* 2, *PS* 4, *PS* 5, *PS* 1, *PS* 3). Die anteilige Ausführung $x_{i,j}$ eines Web Service wird als Abschätzung der Wahrscheinlichkeit verwendet, mit welcher ein Web Service Bestandteil der optimalen Lösung des unrelaxierten Problems ist. Dies führt dazu, dass zuerst den Positionen des Ausführungsplans Web Services zugewiesen werden, für die mit hoher Wahrscheinlichkeit der Web Service bekannt ist, der im optimalen Ausführungsplan enthalten sein wird. Später besetzte Positionen verfügen über eine höhere Anzahl von Web Services, die im relaxierten Modell anteilig ausgeführt werden. Das bedeutet, dass an diesen Positionen die Wahl der Web Services unsicherer ist und im Verlauf des Backtracking-Verfahrens mit einer höheren Wahrscheinlichkeit wieder revidiert werden muss.

Für das Laufzeitverhalten des Backtracking-Verfahrens ist es günstiger, unsichere Entscheidungen erst möglichst spät zu treffen, weil dadurch die Anzahl zu traversierender Knoten minimiert wird, bis ein zulässiger Ausführungsplan gefunden werden kann. Im schlimmsten Fall kann sonst eine falsche Entscheidung bereits durch den Besuch des ersten Kindknotens der Wurzel im Lösungsbaum getroffen werden, die jedoch erst in einem Blatt des entsprechenden Teilbaums erkannt wird. Bis die falsche Entscheidung durch den Besuch eines anderen Kindknotens der Wurzel revidiert würde, müssten alle Blätter des Teilbaums traversiert werden. Weitaus günstiger ist es daher, eine falsche Entscheidung erst mit dem Besuch eines Blattes zu treffen und zu erkennen. In diesem Fall genügt ein Rückschritt zum Vaterknoten des Blattes, um die falsche Entscheidung zu revidieren. Dies ist der Grund, warum H1_RELAX_IP eine Vorsortierung der Kategorien durchführt. So wird die Anzahl der Konstruktionsschritte zum Auffinden einer zulässigen Lösung minimiert. Dies bedeutet vor allem in Anbetracht der evtl. exponentiellen Laufzeit im Falle des Vorhandenseins von Restriktionen ein hohes Einsparpotenzial für die benötigte Rechenzeit und rechtfertigt den zusätzlichen Aufwand der Vorsortierung der Web Services und Kategorien vor dem Start des Backtracking-Verfahrens.

4.4.3 Verbesserungsverfahren H2_SWAP

Weicht der Zielfunktionswert des durch H1_RELAX_IP ermittelten Ausführungsplans stärker als zuvor definiert von einer oberen Schranke ab, wird davon ausgegangen, dass es sich nicht bereits um einen hinreichend guten Ausführungsplan handelt. Um den suboptimalen Ausführungsplan zu verbessern, wird durch die Heuristik H2_SWAP versucht, diesen derart zu verändern, dass möglichst schnell ein lokales Optimum des Zielfunktionswertes erreicht wird. Dafür wird im aktuellen Ausführungsplan an einer zufälligen Position ein Web Service durch einen anderen, zufällig gewählten Web Service der entsprechenden Kategorie ersetzt. Ergibt sich eine Verbesserung des Zielfunktionswertes, wird die Änderung in den Ausführungsplan übernommen, andernfalls wird sie verworfen. Dieser Schritt wird solange wiederholt, bis sich in einer bestimmten Anzahl aufeinanderfolgender Schritte keine weitere Verbesserung mehr erzielen lässt. Abbildung 29 zeigt den Pseudo-Code des Verfahrens H2_SWAP.

```
maxNoImp = v*n      // Maximale Anzahl Iterationen ohne Verbesserung
curNoImp = 0        // Aktuelle Anzahl Iterationen ohne Verbesserung
lastPos = 0         // Letzte gewählte Position
while (curNoImp < MaxNoImp) {
       i = Zufällige Position aus [1;n] ohne lastPos
       lastPos = i
       curWS = AP[i]    // Aktuell an Position gesetzter Web Service
       curZF = Zielfunktionswert des Ausführungsplans AP
       WSRandomOrder[] =   Web Services aus Kategorie i in zufälliger
                           Reihenfolge ohne curWS
       for (j = 1; j< = WSRandomOrder.length; j++) {
              AP[i] = WSRandomOrder[j]
              if (Ausführungsplan AP zulässig) break
       }
       if (Ausführungsplan AP zulässig) {
              if (Zielfunktionswert des Ausführungsplans AP <= curZF) {
                     AP[i] = curWS
                     curNoImp++
              } else
                     curNoImp = 0
       } else {
              curNoImp++
       }
}
```

Abbildung 29: Pseudo-Code H2_SWAP

Zunächst wird die maximale Anzahl erfolgloser Optimierungsschritte festgelegt. Sie wird in Abhängigkeit von der Länge des Ausführungsplans festgesetzt, da bei längeren Ausführungsplänen mehr Positionen und damit auch Positionen für potenziell erfolglose Optimierungsschritte vorhanden sind. Konkret ergibt sich die maximale Anzahl erfolgloser Optimierungsschritte aus der Multiplikation der Länge des Ausführungsplans mit einem Faktor *v*.

Zu Beginn der Ausführung von H2_SWAP wird eine zufällige Position im Ausführungsplan bestimmt, die im Folgenden modifiziert werden soll. Es wird dabei ausgeschlossen, dass es sich um dieselbe Position im Ausführungsplan handelt, die bereits im letzten Optimierungsschritt gewählt wurde. Um eine evtl. erfolglose Veränderung des Ausführungsplans rückgängig machen zu können, wird der momentan an der gewählten Position gesetzte Web Service gespeichert. Zudem wird der Zielfunktionswert des aktuellen Ausführungsplans abgelegt, um nach einer Veränderung des Ausführungsplans beurteilen zu können, ob sich der Zielfunktionswert verbessert hat. Bei der eigentlichen Ersetzung werden der Reihe nach die alternativen Web Services in der zufälligen Reihenfolge an der aktuellen Position des Ausführungsplans eingesetzt. Nach jeder Ersetzung wird geprüft, ob der so entstandene neue Ausführungsplan zulässig ist. Die Ersetzung mit Alternativen endet, sobald der erste zulässige Ausführungsplan erzeugt wurde oder alle Alternativen an der Position eingesetzt wurden, ohne dass hierbei ein zulässiger Ausführungsplan erzeugt werden konnte. Im zweiten Fall wird an der aktuellen Position wieder der ursprünglich gesetzte Web Service eingesetzt, die Anzahl der erfolglosen Optimierungsschritte erhöht und der aktuelle Optimierungsschritt beendet. Im ersten Fall wird der Zielfunktionswert des neuen zulässigen Ausführungsplans bestimmt und mit dem des ur-

sprünglichen Ausführungsplans verglichen. Konnte der Zielfunktionswert gesteigert werden, wird die Änderung im Ausführungsplan beibehalten und der aktuelle Optimierungsschritt beendet. Konnte der Zielfunktionswert hingegen nicht gesteigert werden, wird an der aktuellen Position wieder der ursprünglich gesetzte Web Service eingesetzt, die Anzahl der erfolglosen Optimierungsschritte erhöht und der aktuelle Optimierungsschritt beendet.

4.4.4 Verbesserungsverfahren H3_SIMUL_ANNEAL

Analog zum Beginn von H2_SWAP erfolgt zu Beginn der Ausführung von H3_SIMUL_ANNEAL zunächst eine Prüfung, ob der durch den vorhergehenden Schritt entstandene Ausführungsplan einen Zielfunktionswert besitzt, der nicht mehr als ein zuvor definiertes Maß von der oberen Schranke des theoretisch erreichbaren Zielfunktionswertes abweicht. Ist die Abweichung geringer, wird der aktuelle Ausführungsplan als hinreichend gut eingestuft und die Optimierung beendet. Ansonsten wird die Heuristik H3_SIMUL_ANNEAL ausgeführt.

Die in Abschnitt 4.4.3 vorgestellte Heuristik H2_SWAP läuft Gefahr, bei der Optimierung einen einmal erreichten Bereich um ein lokales Maximum der Zielfunktion nicht mehr verlassen zu können. Um zu verhindern, nur ein lokales Maximum des Zielfunktionswertes zu erreichen, ist es nötig, ein Optimierungsverfahren anzuwenden, welches Schritte erlaubt, die zu einer vorübergehenden Verschlechterung des Zielfunktionswertes führen. Eine Metaheuristik, welche diesen Effekt berücksichtigt und eine zwischenzeitliche Verschlechterung des Zielfunktionswertes explizit vorsieht, ist *Simulated Annealing* (vgl. Abschnitt 4.2.5). Simulated Annealing wird daher durch Ausführen der Heuristik H3_SIMUL_ANNEAL verwendet, um den aktuellen, als suboptimal eingestuften, Ausführungsplan weiter zu verbessern. Das konkret verwendete Verfahren wird im Folgenden näher erläutert.

Das bei der Heuristik H3_SIMUL_ANNEAL verwendete Simulated-Annealing-Verfahren ist in zwei Phasen unterteilt. In der ersten Phase wird eine Mindestanzahl von Iterationen durchgeführt, wohingegen die zweite Phase solange Iterationen durchführt, bis für eine bestimmte Anzahl aufeinanderfolgender Iterationen keine Verbesserung mehr erzielt werden kann. In jeder Phase wird eine Simulated-Annealing-Iteration auf die gleiche Weise durchgeführt, jedoch unterscheiden sich die beiden Phasen hinsichtlich der Reduktion des Temperaturparameters λ und in der Festlegung der jeweils durchzuführenden Anzahl von Iterationen. Die Anzahl der durchzuführenden Iterationen in Phase 1 ergibt sich aus der Länge *n* des Ausführungsplans, multipliziert mit einem Faktor v_1. Während dieser v_1**n* Iterationen wird der Temperaturparameter λ von einem Startwert λ_1 auf einen Endwert λ_2 gleichmäßig reduziert, d. h. in jeder Iteration nimmt λ um den Betrag $\frac{\lambda_1 - \lambda_2}{v_1 \cdot n}$ ab. Im konkret angewendeten Verfahren wurde $v_1 = 4$, $\lambda_1 = 1$ und $\lambda_2 = 0{,}1$ gewählt. Dies bedeutet, dass in Phase 1 viermal so viele Iterationen ausgeführt werden, wie der Ausführungsplan Positionen aufweist, wobei der Temperaturparameter λ gleichmäßig von dem anfänglichen Wert 1 auf den Wert 0,1 sinkt. In Phase 2 werden so viele Iterationen durchgeführt, bis eine maximale Anzahl aufeinanderfolgender Iterationen ohne Verbesserung des Zielfunktionswertes erreicht wird. Die maximale

Anzahl aufeinanderfolgender Iterationen ohne Verbesserung des Zielfunktionswertes ergibt sich durch Multiplikation der Länge *n* mit einem Faktor v_2. In jeder Iteration wird der Temperaturparameter λ durch Multiplikation mit einem Faktor b_λ $(0 < b_\lambda < 1)$ um einen gewissen Prozentsatz gesenkt. Dadurch strebt der Temperaturparameter λ stetig, aber mit wachsender Anzahl von Iterationen betragsmäßig langsamer, gegen 0. Im konkret angewendeten Verfahren wurde $v_2 = 2$ und $b_\lambda = 0{,}95$ verwendet. Dies bedeutet, dass solange Iterationen durchgeführt werden, bis die Anzahl aufeinanderfolgender Iterationen ohne Verbesserung des Zielfunktionswertes den Wert der doppelten Länge des Ausführungsplans erreicht hat, wobei in jeder Iteration der Temperaturparameter λ um 5% gesenkt wird.

In beiden Phasen wird eine Iteration auf die gleiche Art und Weise durchgeführt. Eine Iteration des Simulated-Annealing-Verfahrens entspricht einem Optimierungsschritt des bei der Heuristik H2_SWAP verwendeten Verfahrens. Es unterscheidet sich jedoch im Falle der Nichtverbesserung des Zielfunktionswertes durch die bedingte Akzeptanz der Verschlechterung in Abhängigkeit von der Höhe der Verschlechterung Δ und dem Temperaturparameter λ. Zu Beginn jeder Iteration wird eine Position im aktuellen Ausführungsplan zufällig ausgewählt. Um die Variabilität in den Positionen zu erhöhen, wird sichergestellt, dass bei aufeinanderfolgenden Iterationen nicht dieselbe Position ausgewählt wird. Für die gewählte Position *i* wird nun ein zum aktuell an dieser Position gesetzten Web Service alternativer Web Service aus der entsprechenden Kategorie *i* ausgewählt, um mit diesem eine neue zulässige Nachbarlösung zu erzeugen. Aus der Kategorie *i* werden alle alternativen Web Services in einer zufälligen Reihenfolge nacheinander an der aktuellen Position des Ausführungsplans gesetzt. Es wird jeweils geprüft, ob der entstehende Ausführungsplan zulässig ist. Sobald ein zulässiger Ausführungsplan gefunden wurde, wird dieser als zu betrachtende Nachbarlösung festgehalten. Wurden jedoch alle alternativen Web Services der Kategorie *i* erfolglos eingesetzt, ohne dass ein zulässiger Ausführungsplan erzeugt werden konnte, wird die Anzahl der aufeinanderfolgenden Iterationen ohne Verbesserung des Zielfunktionswertes erhöht und die aktuelle Iteration abgebrochen.

Konnte eine zulässige Nachbarlösung $\vec{x}'$ ermittelt werden, wird der Zielfunktionswert $F(\vec{x}')$ der Nachbarlösung mit dem Zielfunktionswert $F(\vec{x})$ der aktuellen Lösung $\vec{x}$ verglichen. Besitzt die Nachbarlösung einen höheren Zielfunktionswert ($F(\vec{x}') > F(\vec{x})$), wird die Nachbarlösung als neue aktuelle Lösung $\vec{x}$ übernommen und die Anzahl aufeinanderfolgender Iterationen ohne Verbesserung des Zielfunktionswertes auf Null gesetzt.

Besitzt die Nachbarlösung hingegen einen niedrigeren Zielfunktionswert, wird die Anzahl aufeinanderfolgender Iterationen ohne Verbesserung des Zielfunktionswertes um den Wert eins erhöht und die Nachbarlösung nur mit einer bestimmten Wahrscheinlichkeit als neue aktuelle Lösung $\vec{x}$ übernommen. Um die Akzeptanzwahrscheinlichkeit $z(\Delta, \lambda)$ zu berechnen, wird zunächst die Höhe Δ der Verschlechterung als prozentualer Unterschied der Zielfunktionswerte mit $\Delta = \frac{F(\vec{x}) - F(\vec{x}')}{|F(\vec{x})|}$ bestimmt. Anschließend kann die Akzeptanzwahrscheinlichkeit

z mit dem aktuellen Temperaturparameter λ durch $z(\Delta,\alpha) = e^{-\frac{\Delta}{\lambda}}$ berechnet werden. Um zu entscheiden, ob die Akzeptanzwahrscheinlichkeit z zu einer Annahme oder einer Ablehnung der Nachbarlösung $\vec{x}'$ führt, wird eine gleichverteilte Zufallszahl z' aus dem Intervall [0;1] gezogen. Ist $z' \leq z$, wird die Nachbarlösung $\vec{x}'$ als neue aktuelle Lösung $\vec{x}$ akzeptiert. Ist hingegen $z' > z$, so wird die Nachbarlösung $\vec{x}'$ verworfen und die aktuelle Lösung $\vec{x}$ bleibt unverändert.

Das Simulated-Annealing-Verfahren terminiert nach Erreichen der maximal zulässigen Anzahl von Iterationen ohne Verbesserung des Zielfunktionswertes in Phase 2. Da das Verfahren auch Verschlechterungen des Zielfunktionswertes zulässt, ist es möglich, dass der am Ende erreichte Zielfunktionswert unter den anfänglichen Zielfunktionswert gesunken ist, der bereits zuvor vorlag. In diesem Fall wird die aktuelle Lösung auf die ursprüngliche Lösung zurückgesetzt.

Abbildung 30 zeigt den Pseudo-Code der Heuristik H3_SIMUL_ANNEAL, der im Folgenden näher erläutert wird. Die `while`-Schleife des Pseudo-Codes umfasst eine Iteration des Simulated-Annealing-Verfahrens. Sie wird solange wiederholt, bis die maximale Anzahl aufeinanderfolgender Iterationen ohne Verbesserung des Zielfunktionswertes `maxNoImp` erreicht wird. Die aktuelle Anzahl der aufeinanderfolgenden Iterationen ohne Verbesserung des Zielfunktionswertes `curNoImp` wird nur in Phase 2 des Verfahrens erhöht, d. h. nur dann, nachdem die Anzahl aller Iterationen `curIterPhase1` die in Phase 1 durchzuführenden Iterationen `iterPhase1` überschritten hat. In jeder Iteration wird mittels der Anweisung $\vec{x}'$ `= Nachbarlösung(`$\vec{x}$`)` eine zulässige Nachbarlösung von $\vec{x}$ ermittelt, die durch den Austausch eines Web Service an einer zufälligen Stelle im Ausführungsplan entsteht. Bei zwei aufeinanderfolgenden Aufrufen der Funktion „Nachbarlösung“ wird nie dieselbe Position des Ausführungsplans verändert. Sollte an der gewählten Position kein anderer Web Service zu einem zulässigen Ausführungsplan führen, wird ein unzulässiger Ausführungsplan bzw. eine unzulässige Nachbarlösung $\vec{x}'$ zurückgeliefert. Wird eine zulässige Nachbarlösung $\vec{x}'$ zurückgeliefert, wird durch den Vergleich der Zielfunktionswerte der aktuellen Lösung F($\vec{x}$) und der Nachbarlösung F($\vec{x}'$) festgestellt, ob eine Steigerung des Zielfunktionswertes erreicht wurde. Eine Verbesserung des Zielfunktionswertes wird in jedem Fall akzeptiert und daher die Nachbarlösung $\vec{x}'$ zur neuen aktuellen Lösung $\vec{x}$. Im Fall einer Verschlechterung erfolgt die Annahme der Nachbarlösung $\vec{x}'$ nur mit der in Abhängigkeit von der Verschlechterung Δ und dem Temperaturparameter λ berechneten Annahmewahrscheinlichkeit z. Am Ende jeder Iteration erfolgt die Reduktion des Temperaturparameters λ, je nach Phase des Verfahrens entweder um den Betrag Δ_λ (Phase 1) oder mittels Multiplikation mit dem Faktor b_λ (Phase 2). Nachdem in Phase 2 die maximale Anzahl aufeinanderfolgender Iterationen ohne Verbesserung des Zielfunktionswertes `maxNoImp` erreicht wurde und damit die `while`-Schleife verlassen wird, erfolgt abschließend ein Vergleich der ursprünglichen Anfangslösung $\vec{x}''$ mit der aktuellen Lösung $\vec{x}$. Hat sich der Zielfunktionswert der aktuellen Lösung gegenüber dem Zielfunktionswert der ursprünglichen Lösung verschlechtert, wird die ursprüngliche Lösung wiederhergestellt.

```
x⃗ = AP                          // Ausgangslösung ist aktueller Ausführungsplan
x⃗″ = x⃗                          // Sicherung Ausgangslösung für abschließenden Vergleich
λ = λ1                          // Starttemperatur
Δλ = (λ1 − λ2) / (v1 · n)       // Reduktion von λ um Δλ in Iteration von Phase 1
iterPhase1 = v1*n               // Max. Anzahl Iterationen in Phase 1
maxNoImp = v2*n                 // Max. Anzahl von Iterationen ohne Verbesserung in Phase 2
curNoImp = 0                    // Aktuelle Anzahl von Iterationen ohne Verbesserung
curIterPhase1 = 0
Phase1 = true                   // Aktuelle Phase

while (curNoImp < maxNoImp) {
       x⃗′ = Nachbarlösung( x⃗ )
       if (zulässig( x⃗′ )) {
              if ( F( x⃗′ ) <= F( x⃗ ) ) {    // Nachbarlösung ist nicht besser
                     if not Phase1
                            curNoImp++
                     Δ = (F(x⃗) − F(x⃗′)) / |F(x⃗)|
                     z = e^(−Δ/λ)
                     z' = gleichverteilte Zufallszahl aus [0;1]
                     if (z' < z)
                            x⃗ = x⃗′
              } else {
                     curNoImp = 0        // Nachbarlösung ist besser
                     x⃗ = x⃗′ }
       } else {
              if (not Phase1)            // Es existiert keine zulässige
                     curNoImp++          // Nachbarlösung an dieser Position
       }
       if (Phase1)
              λ = λ - Δλ                 // Reduktion von λ in Phase 1
              curIterPhase1= curIterPhase1 + 1
              If (curIterPhase1 = iterPhase1) Phase1 = false
       else
              λ = λ * cλ                 // Reduktion von λ in Phase 2
}
if ( F( x⃗ ) < F( x⃗″ ) )       // Falls akt. Lösung x⃗ schlechter als Aus-
       x⃗ = x⃗″                 // gangslösung x⃗″, x⃗ auf x⃗″ zurücksetzen.
```

Abbildung 30: Pseudo-Code H3_SIMUL_ANNEAL

4.5 Implementierung

Dieser Abschnitt beschreibt die Umsetzung und Implementierung der in Abschnitt 4.4 vorgestellten heuristischen Lösungsverfahren. Die Implementierung der Heuristiken H1_RELAX_IP, H2_SWAP und H3_SIMUL_ANNEAL wird in den Abschnitten 4.5.1, 4.5.2

und 4.5.3 beschrieben. Die Implementierung der im Rahmen dieser Arbeit entwickelten Simulationsumgebung *WorkflowOptimizer* wird im Abschnitt B des Anhangs vorgestellt.

Die Heuristiken sind mit Java 2 Platform Standard Edition (J2SE) in der Version 5.0 implementiert. Zur Lösung des linearen Optimierungsproblems wird der Open-Source-Solver *lp_solve* 5.1.1.3 verwendet.

4.5.1 Eröffnungsverfahren H1_RELAX_IP

Die Ausführung der Heuristik beginnt mit der exakten Lösung des relaxierten Optimierungsproblems durch den Solver. Anschließend kann auf Basis dieser Lösung durch das Backtracking-Verfahren eine erste zulässige Lösung für das unrelaxierte Optimierungsproblem erzeugt werden. Eine Lösung des Optimierungsproblems, d. h., ein zulässiger Ausführungsplan zum Workflow, wird durch die Klasse `ExecutionPlan` gekapselt, welche die einzelnen Prozessschritte und den dort jeweils auszuführenden Web Service in einer Liste von `ExecutionPlanItem`-Objekten ablegt. Um einen leeren Ausführungsplan passender Länge zu erstellen, der im Laufe der Heuristik mit konkreten Web Services zur Ausführung der Prozessschritte gefüllt wird, kann die Funktion `createCurrentExecutionPlan` der `ExecutionPlan`-Klasse verwendet werden. Dieser wird als Argument das `Workflow`-Objekt übergeben, das den zu optimierenden Workflow repräsentiert. Die Schritte eins und zwei der Heuristik, die einen ersten zulässigen Ausführungsplan erzeugen, sind in der Klasse `H1RelaxIP` zusammengefasst. Durch Ausführung der Methode `solve` des erzeugten Objekts werden diese ersten beiden Schritte der Heuristik ausgeführt und der hierbei erzeugte Ausführungsplan in jenem `ExecutionPlan`-Objekt abgelegt, welches im Konstruktor übergeben wurde. Die Schritte eins und zwei der Heuristik werden hierbei, wie im Abschnitt 4.4.2 beschrieben, durchgeführt.

4.5.1.1 Realisierung des ersten Schrittes der Heuristik

Zur Umsetzung des ersten Schrittes der Heuristik wird das für den Solver aufgebaute mathematische Optimierungsmodell, dass durch ein `MIPModel`-Objekt gekapselt wird, zunächst relaxiert und anschließend durch den Solver gelöst. Die Relaxation wird durch einen Aufruf der Methode `relax` des `MIPModel`-Objekts durchgeführt und geschieht durch die Aufhebung der Ganzzahligkeitsbedingung auf den Entscheidungsvariablen $x_{i,j}$. Der Wertebereich der Entscheidungsvariablen $x_{i,j}$ bleibt auf das Intervall [0;1] beschränkt, jedoch kann eine Entscheidungsvariable $x_{i,j}$ nach der Relaxation einen beliebigen nicht-ganzzahligen Wert aus diesem Intervall annehmen. Durch den Aufruf der Funktion `solve` des `MIPModel`-Objekts wird das relaxierte Optimierungsproblem mithilfe des Solvers gelöst. Aufgrund der Relaxation ist der Solver in der Lage, die optimale Lösung des Optimierungsproblems durch den Simplex-Algorithmus zu bestimmen. Da eine Anwendung des aufwendigen Branch&Bound-Verfahrens aufgrund des Fehlens ganzzahliger Entscheidungsvariablen nicht nötig ist, kann die optimale Lösung des relaxierten Problems sehr performant erzeugt werden. Kann keine Lösung für das relaxierte Optimierungsproblem ermittelt werden, so bedeutet dies, dass auch das unrelaxierte Optimierungsproblem keine Lösung besitzen kann und die Methode `solve` des `H1RelaxIP`-Objekts wird verlassen, ohne den ursprünglich übergebenen, leeren Ausfüh-

rungsplan zu verändern. Konnte hingegen eine Lösung für das relaxierte Optimierungsproblem gefunden werden, handelt es sich dabei um die optimale Lösung des relaxierten Optimierungsproblems. Der Zielfunktionswert dieser Lösung wird vom Solver erfragt und wird im `MIPModel`-Objekt im Attribut `lastOptimalValue` gespeichert. Dieser Zielfunktionswert bildet eine obere Schranke für den Zielfunktionswert des unrelaxierten Optimierungsproblems und wird im Folgenden beim Übergang zwischen den einzelnen Heuristiken dazu verwendet, die Güte des bisher ermittelten Ausführungsplans abzuschätzen, um so festzustellen, ob die Ausführung eines Verbesserungsverfahrens überhaupt noch notwendig ist.

4.5.1.2 Realisierung des zweiten Schrittes der Heuristik

Der zweite Schritt der Heuristik wird ebenfalls innerhalb der Funktion `solve` des `H1RelaxIP`-Objekts ausgeführt. Mithilfe des Backtracking-Verfahrens wird hier ein erster zulässiger Ausführungsplan ermittelt. Um die Effektivität des Backtracking-Verfahrens zu steigern, werden vor der Anwendung dieses Verfahrens die Prozessschritte und die jeweils für einen Prozessschritt möglichen Web Services gemäß der Lösung des relaxierten Optimierungsproblems vorsortiert. Die Vorsortierung wird durch die Funktion `init` des `H1RelaxIP`-Objekts durchgeführt. Die Sortierung erfolgt, wie in Abschnitts 4.4.2.2 beschrieben, nach der geschätzten Wahrscheinlichkeit, mit der ein Web Service für einen bestimmten Prozessschritt die optimale Besetzung darstellt und wie sicher diese Entscheidung ist. Die zur Sortierung benötigten Werte der Entscheidungsvariablen $x_{i,j}$ aus der optimalen Lösung des relaxierten Optimierungsproblems werden über entsprechende API-Aufrufe vom Solver ausgelesen und in Arrays von `ExecutionPlanItemWSSortInfo`-Objekten, gruppiert nach Prozessschritten im jeweils zugehörigen `ExecutionPlanItem`-Objekt, abgelegt.

Die eigentliche Sortierung wird anschließend auf Basis der Methode `java.util.Arrays.sort` und einer angepassten Implementierung des `java.util.Comparator`-Interfaces realisiert. Nach der Sortierung ist für jeden Prozessschritt bekannt, in welcher Reihenfolge die Web Services an der entsprechenden Position im Ausführungsplan durch das Backtracking-Verfahren gesetzt werden müssen. Zusätzlich wird die Liste `positionVector` erstellt, welche die Indizes der Positionen des Ausführungsplans in der Reihenfolge enthält, in der das Backtracking-Verfahren diese Positionen mit Web Services zu besetzen hat. Als Rückgabewert der Funktion `init`, welche die Vorsortierung der Web Services und Prozessschritte vorgenommen hat, wird der `positionVector` geliefert. Dieser wird nun als Argument an die Funktion `solveWithPositionVector` des `H1RelaxIP`-Objekts übergeben, die das eigentliche Backtracking-Verfahren realisiert. Die Positionen des Ausführungsplans (`ExecutionPlanItem`-Objekte) werden in der durch den `positionVector` vorgegebenen Reihenfolge mit Web Services besetzt, wobei die Web Services in der Reihenfolge ihrer Vorsortierung gesetzt werden. Die Reihenfolge, mit der die Web Services an einer Position zu setzen sind, kann in jedem `ExecutionPlanItem`-Objekt dem sortierten Array von `ExecutionPlanItemWSSortInfo`-Objekten entnommen werden. Ist es dem Backtracking-Verfahren möglich, alle Positionen des Ausführungsplans so mit Web Services zu besetzen, dass dieser zulässig ist, so ist ein erster gültiger Ausführungsplan

gefunden und die Funktion `solveWithPositionVector` liefert den Wahrheitswert `true` zurück. Konnte hingegen kein zulässiger Ausführungsplan gefunden werden, wird `false` zurückgeliefert und das untersuchte unrelaxierte Optimierungsproblem ist als unlösbar erkannt.

4.5.2 Verbesserungsverfahren H2_SWAP

Konnte durch das `H1RelaxIP`-Objekt ein zulässiger Ausführungsplan erzeugt werden, wird der Zielfunktionswert dieses Ausführungsplans über die Funktion `quality` ermittelt. Weicht der so berechnete Zielfunktionswert weniger als ein zuvor definierter Wert von der oberen Schranke für den Zielfunktionswert ab, wird die Heuristik beendet. Anderenfalls wird angenommen, dass es sich um einen suboptimalen Ausführungsplan handelt und die Heuristik H2_SWAP wird ausgeführt (siehe Abschnitt 4.4.3).

Das in Heuristik H2_SWAP angewendete Optimierungsverfahren wird durch die Klasse `H2SWAP` bereitgestellt. Nach der Ausführung der Funktion `solve` des `H1RelaxIP`-Objekts liegt der mittels des Backtracking-Verfahrens erstellte Ausführungsplan in dem `ExecutionPlan`-Objekt vor, welches dem `H1RelaxIP`-Objekt bei seiner Erzeugung im Konstruktor übergeben wurde. Um diesen Ausführungsplan durch das Optimierungsverfahren H2_SWAP weiter optimieren zu lassen, wird eine Instanz der Klasse `H2SWAP` erzeugt und das `ExecutionPlan`-Objekt im Konstruktor übergeben. Durch den Aufruf der Funktion `optimize` des `H2SWAP`-Objekts wird das Optimierungsverfahren auf den Ausführungsplan angewendet. Hierzu werden zum aktuell besten, bekannten Ausführungsplan Nachbarlösungen durch den zufälligen Austausch je eines Web Service erzeugt. Wird hierbei eine Nachbarlösung gefunden, die einen höheren Zielfunktionswert besitzt, wird sie zur aktuell besten, bekannten Lösung und man fährt mit der Untersuchung ihrer Nachbarlösungen fort. Das Verfahren endet, sobald in einer bestimmten Anzahl aufeinanderfolgender Schritte nur solche Nachbarlösungen erzeugt wurden, die zu keiner Steigerung des Zielfunktionswertes geführt haben. Der optimierte Ausführungsplan wird als Rückgabewert in Form eines `ExecutionPlan`-Objekts von der Funktion zurückgeliefert.

4.5.3 Verbesserungsverfahren H3_SIMUL_ANNEAL

Nachdem der optimierte Ausführungsplan in Form eines `ExecutionPlan`-Objekts von der Funktion `optimize` des `H2SWAP`-Objekts zurückgeliefert wurde, wird wiederum geprüft, ob der Zielfunktionswert des Ausführungsplans stärker als zuvor definiert von der oberen Schranke des Zielfunktionswertes abweicht. Ist dies der Fall, wird die Heuristik H3_SIMUL_ANNEAL ausgeführt, um den Ausführungsplan weiter zu optimieren.

Das bei H3_SIMUL_ANNEAL zum Einsatz kommende Simulated-Annealing-Verfahren wird in der Klasse `H3SimulAnneal` implementiert. Nach der Ausführung der Funktion `optimize` des `H2SWAP`-Objekts wird mittels des Optimierungsverfahrens der optimierte Ausführungsplan als Rückgabewert der Funktion in einem `ExecutionPlan`-Objekt zurückgegeben. Um diesen Ausführungsplan durch das Simulated-Annealing-Verfahren weiter optimieren zu lassen, wird eine Instanz der Klasse `H3SimulAnneal` erzeugt und das `Execution-`

`Plan`-Objekt im Konstruktor übergeben. Durch den Aufruf der Funktion `optimize` des `H3SimulAnneal`-Objekts wird das Simulated-Annealing-Verfahren auf den Ausführungsplan angewendet und der optimierte Ausführungsplan als Rückgabewert in Form eines `ExecutionPlan`-Objekts von der Funktion zurückgeliefert. Die Implementierung der Funktion `optimize` setzt das Simulated-Annealing-Verfahren, wie in Abschnitt 4.4.4 beschrieben, um. Parameter, die das Verhalten des Simulated-Annealing-Verfahrens beeinflussen, können in der Ausführungsumgebung des WorkflowOptimizer (vgl. Anhang, Abschnitt B) konfiguriert werden. Hierzu zählen z. B. der Faktor v_1 zur Berechnung der Anzahl der durchzuführenden Iterationen in Phase 1 des Simulated-Annealing-Verfahrens, der Temperaturparameterwert λ_1 beim Eintritt in Phase 2, der Temperaturparameterwert λ_2 beim Beenden von Phase 1, der Faktor v_2 zur Berechnung der maximal erlaubten Anzahl von Iterationen ohne Verbesserung des Zielfunktionswertes in Phase 2 sowie der Faktor b_λ, durch den die Abnahme des Temperaturparameters λ in Phase 2 herbeigeführt wird.

4.6 Evaluation

In diesem Abschnitt werden die zuvor entwickelten Heuristiken H1_RELAX_IP, H2_SWAP und H3_SIMUL_ANNEAL evaluiert. Um die Ergebnisse und die Rechenzeit der Heuristiken einem exakten Lösungsverfahren gegenüberstellen zu können, wurden die Testreihen, sofern mit vertretbarem Rechenaufwand möglich, auch in Form eines unrelaxierten Optimierungsmodells durch den Solver lp_solve, der ein Branch&Bound-Verfahren verwendet, gelöst. Hierbei wird zum einen die Rechenzeit der Heuristik mit der Rechenzeit der exakten Lösung verglichen. Zudem wird untersucht, inwieweit der Zielfunktionswert der in dieser Arbeit entwickelten Heuristiken von dem Zielfunktionswert der exakten Lösung abweicht. In diesem Zusammenhang wird die Metrik *Lösungsgüte* eingeführt. Eine Lösungsgüte von 100% bedeutet, dass die Heuristik denselben Zielfunktionswert wie das exakte Lösungsverfahren errechnet. Bei einer Lösungsgüte von 90% liegt der durch die Heuristiken berechnete Zielfunktionswert 10% unter dem optimalen Zielfunktionswert. Die Testreihen werden mithilfe der im Anhang, Abschnitt B vorgestellten Simulationsumgebung evaluiert.

Um den Einfluss eines die Lösungsgüte oder Laufzeit der Heuristik betreffenden Parameters näher untersuchen zu können, werden Testreihen gebildet, die ceteris paribus jeweils einen Parameter variieren. Die Einflussgrößen *Workflowlänge* (siehe Abschnitt 4.6.1), *Anzahl der Prozessschrittkandidaten* (siehe Abschnitt 4.6.2) und *Restriktionsstärke* (siehe Abschnitt 4.6.3) werden untersucht. Die Evaluation der Verbesserungsverfahren ist Gegenstand von Abschnitt 4.6.4.

Die Restriktionsstärke beschreibt den Einfluss der Nebenbedingungen, durch welche die Gesamtparameterwerte einer Lösung beschränkt werden. Je stärker eine Restriktion ist, d. h. je mehr sie den Lösungsraum einschränkt, desto schwieriger wird es, eine zulässige Lösung zu finden. Zur Definition der Restriktionsstärke wird im Folgenden ein prozentualer Wert verwendet, der die Höhe der Annäherung eines Schrankenwertes einer Restriktion an den bestmöglichen Gesamtparameterwert beschreibt. Kann ein Gesamtparameterwert aufgrund der

Parameterwerte in der Datenbasis nur Werte zwischen $\underline{h}$ und $\overline{h}$ annehmen und stellt $\overline{h}$ den bestmöglichen Wert dar, so ist eine Restriktion des Gesamtparameters auf $\overline{h}$ die stärkste definierbare Restriktion. Ihr wird die Restriktionsstärke 100% zugewiesen. Eine Restriktion des Gesamtparameters auf $\underline{h}$ entspricht keiner Restriktion, da aufgrund der Definition von $\underline{h}$ als schlechtestmöglichen Gesamtparameterwert jeder Gesamtparameterwert die Restriktion erfüllt. Einer Restriktion auf $\underline{h}$ wird daher die Restriktionsstärke 0% zugewiesen. Einer Restriktionsstärke von 50% entspricht eine Restriktion auf den Mittelwert aller Gesamtparameterwerte zwischen $\underline{h}$ und $\overline{h}$, da im Mittel die Hälfte aller erzeugbaren Gesamtparameterwerte die Restriktion erfüllt.

Die Simulation wurde auf einem PC mit einem Intel Pentium 4 Prozessor (3 GHz) sowie 1 GB Arbeitsspeicher unter dem Betriebssystem Microsoft Windows XP Professional durchgeführt.

4.6.1 Einfluss der Workflowlänge

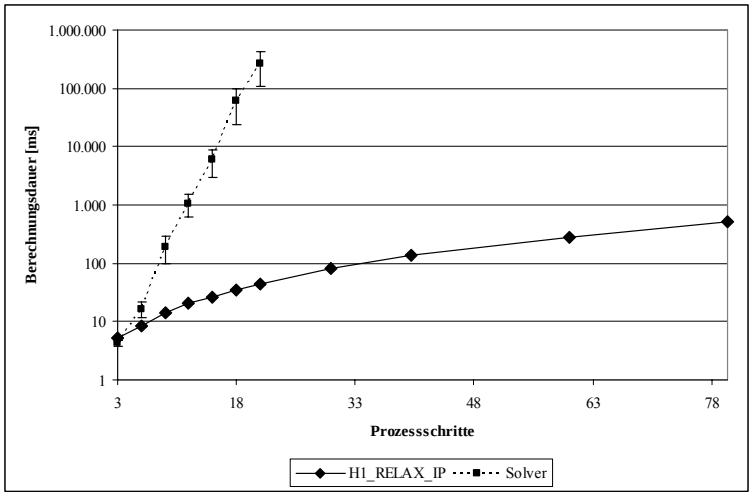
Um den Einfluss der Workflowlänge auszuwerten, wird diese von drei Prozessschritten bis hin zu einer Workflowlänge von 80 Prozessschritten erhöht. Für jede Workflowlänge werden 35 Testfälle generiert. Die Generierung der Testfälle wird in Abschnitt B.4 beschrieben. Pro Prozessschritt stehen 40 alternative Web Services gleicher Funktionalität zur Verfügung. Jeder Web Service wird durch vier nicht-funktionale Eigenschaften beschrieben, die alle durch eine Nebenbedingung (Restriktionsstärke 10%) eingeschränkt sind.

In Tabelle 7 wird die vom Solver benötigte Zeit zur Ermittlung des optimalen Ausführungsplans mit t_s bezeichnet und der ermittelte Zielfunktionswert (Gesamtnutzen des Ausführungsplans) mit $F(\vec{x}_s)$. Die von der Heuristik benötigte Zeit zur Ermittlung eines optimierten Ausführungsplans wird mit t_h bezeichnet und der zugehörige Zielfunktionswert mit $F(\vec{x}_h)$. In der Spalte $\frac{t_h}{t_s}$ wird die Lösungszeit der Heuristik in Prozent der benötigten Lösungszeit des Solvers ausgedrückt. Die Spalte $\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$ enthält den Zielfunktionswert der Lösung der Heuristik in Prozent des Zielfunktionswertes der optimalen Lösung des Solvers.

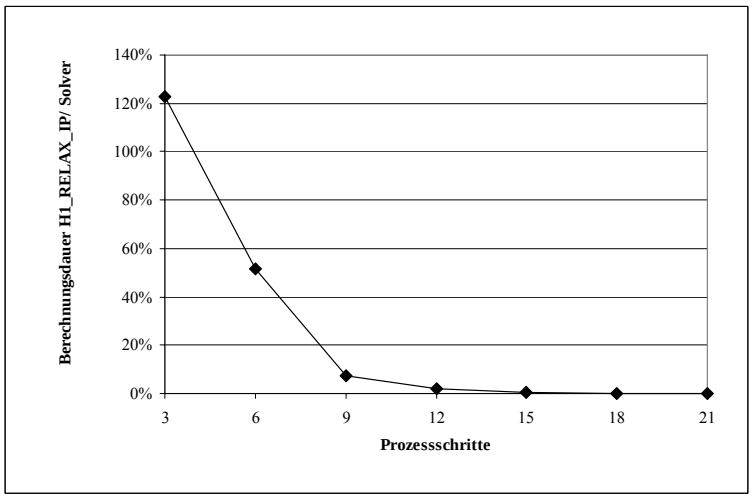
Workflow-länge	Solver t_s [ms]	H1_RELAX_IP t_h [ms]	$\frac{t_h}{t_s}$ [%]	$\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$ [%]
3	4,29	5,27	123,03	99,96
6	16,35	8,47	51,81	99,89
9	194,99	13,95	7,15	99,72
12	1.062,72	20,36	1,92	99,44
15	5.976,94	26,28	0,44	99,38
18	60.772,19	35,57	0,06	99,23
21	264.177,57	43,37	0,02	98,83
30	n/a	81,91	n/a	n/a
40	n/a	138,20	n/a	n/a
60	n/a	284,80	n/a	n/a
80	n/a	504,12	n/a	n/a

Tabelle 7: Evaluierung der Workflowlänge

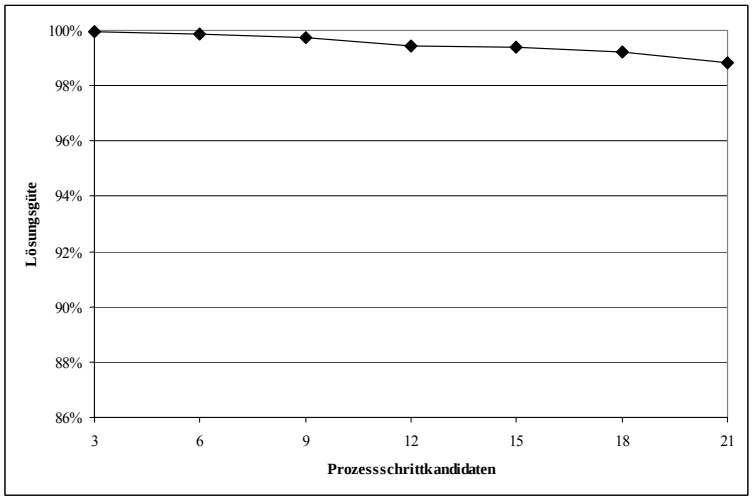
In Abbildung 31a wird die Rechendauer der Heuristik mit der des Solvers verglichen (95%-Konfidenzintervall). Das Laufzeitverhalten der Heuristik H1_RELAX_IP ist ab einer Workflowlänge von 6 Prozessschritten deutlich besser als das der exakten Lösung. Ab einer Workflowlänge von 15 Prozessschritten benötigt die Heuristik weniger als 1% der Rechenzeit des Solvers und erreicht eine Lösungsgüte von über 99%. Unabhängig von der Länge eines Workflows erreicht die Heuristik H1_RELAX_IP eine Lösungsgüte von mindestens 98,83%. Nur in Testreihen mit einer geringen Anzahl an Prozessschritten ist das Laufzeitverhalten des Solvers geringfügig besser als das der Heuristik H1_RELAX_IP. Das Laufzeitverhalten der Heuristik skaliert auch bei Workflows mit hoher Anzahl von Prozessschritten. Ab einer Workflowlänge von 30 Prozessschritten liegen keine exakten Lösungen mehr vor, da der Solver nicht in der Lage ist, Ergebnisse innerhalb einer akzeptablen Zeit zu liefern. Die Heuristik H1_RELAX_IP hingegen benötigt selbst bei einer Workflowlänge von 80 Prozessschritten nur ca. 500 ms. Bei einer Steigerung der Geschäftsprozesslänge von 3 auf 21 Prozessschritte (Steigerung um Faktor 7) steigt die benötigte Rechenzeit des Solvers von ca. 4 ms auf ca. 264.000 ms an (Steigerung um ca. Faktor 61.580). Die benötigte Rechenzeit der Heuristik hingegen steigt lediglich von ca. 5 ms auf ca. 43 ms an (Steigerung um ca. Faktor 9).



(31a) Rechendauer von H1_RELAX_IP und Solver



(31b) Rechendauer von H1_RELAX_IP relativ zum Solver



(31c) Lösungsgüte H1_RELAX_IP

Abbildung 31: Evaluierung der Workflowlänge

Bei einer Workflowlänge von 21 benötigt die Heuristik nur 0,19% der Laufzeit des Solvers (siehe Abbildung 31b), erzeugt jedoch eine Lösung, deren Zielfunktionswert 98,83% des Zielfunktionswertes der optimalen Lösung erreicht. Selbst mit zunehmender Geschäftsprozesslänge nimmt die Abweichung vom Optimum nur leicht zu (siehe Abbildung 31c).

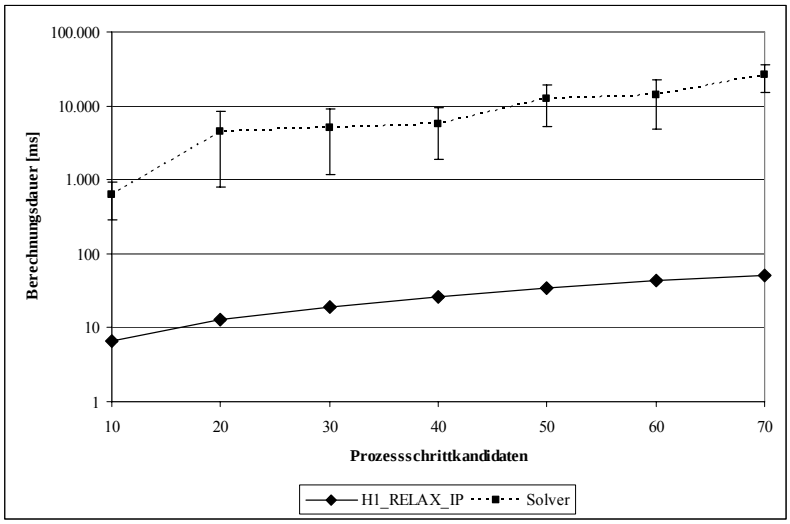
4.6.2 Einfluss der Prozessschrittkandidaten

Zur Untersuchung des Einflusses der Prozessschrittkandidatenanzahl auf das Laufzeitverhalten und die Lösungsgüte werden Testfälle untersucht, bei denen die Anzahl der Prozessschrittkandidaten in Zehnerschritten von 10 auf 70 erhöht werden. Die Länge des Workflows wird auf 15 Prozessschritte festgelegt. Jeder Web Service wird durch vier nicht-funktionale Eigenschaften beschrieben, die alle durch eine Nebenbedingung (Restriktionsstärke 10%) eingeschränkt sind. Die sich bei der Lösung der Testfälle durch den Solver und die Heuristik ergebenden Laufzeiten und Zielfunktionswerte können Tabelle 8 entnommen werden. Diese ist wie Tabelle 7 aufgebaut.

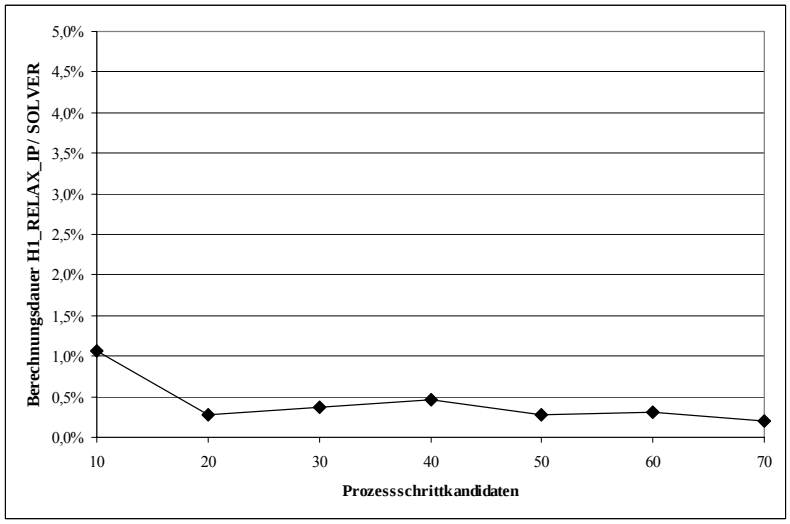
Prozessschritt-kandidaten	Solver t_s [ms]	H1_RELAX_IP t_h [ms]	$\frac{t_h}{t_s}$ [%]	$\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$ [%]
10	617,42	6,59	1,07	98,31
20	4.528,68	12,69	0,28	99,09
30	5.122,33	19,20	0,37	99,18
40	5.723,99	26,38	0,46	99,49
50	12.302,95	34,86	0,28	99,43
60	13.789,37	43,29	0,31	99,39
70	25.840,12	51,09	0,20	99,27

Tabelle 8: Evaluierung der Anzahl der Prozessschrittkandidaten

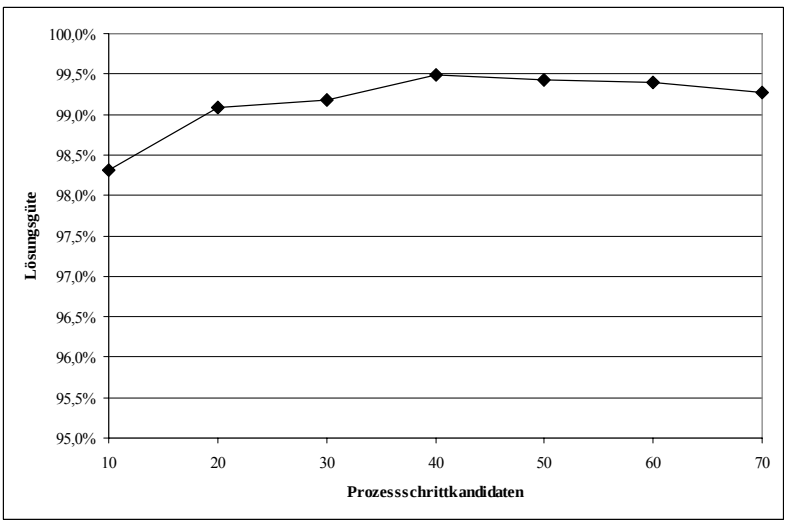
In Abbildung 32a wird die Rechendauer der Heuristik mit der des Solvers verglichen (95%-Konfidenzintervall). Wie bei der Erhöhung der Workflowlänge, so steigt auch bei der Erhöhung der Prozessschrittkandidatenanzahl die benötigte Rechenzeit des Solvers und der Heuristik an. Der Anstieg der benötigten Rechenzeit fällt jedoch im Gegensatz zur Erhöhung der Geschäftsprozesslänge wesentlich geringer aus. Der Unterschied in den benötigten Laufzeiten zwischen Solver und Heuristik ist deutlich. Bei einer Steigerung der Anzahl der Prozessschrittkandidaten von 10 auf 70 (Steigerung um Faktor 7) steigt die benötigte Rechenzeit des Solvers von ca. 617 ms auf ca. 26.000 ms an (Steigerung ca. um Faktor 40). Die benötigte Rechenzeit der Heuristik hingegen steigt lediglich von ca. 7 ms auf ca. 51 ms an (Steigerung ca. um Faktor 7). Zur Lösung der Testreihe mit 70 Prozessschrittkandidaten benötigt die Heuristik nur 0,2% der Rechenzeit des Solvers (siehe Abbildung 32b) und erreicht hierbei eine Lösungsgüte von 99,27%. Der Laufzeitvorteil der Heuristik steigt mit zunehmender Anzahl von Prozessschrittkandidaten an, zudem bleibt die Güte der ermittelten Lösung auf einem Niveau von ca. 99% konstant (siehe Abbildung 32c).



(32a) Rechendauer von H1_RELAX_IP und Solver



(32b) Rechendauer von H1_RELAX_IP relativ zum Solver



(32c) Lösungsgüte von H1_RELAX_IP

Abbildung 32: Evaluierung des Einflusses der Anzahl der Prozessschrittkandidaten

4.6.3 Einfluss der Restriktionsstärke

Um den Einfluss der Restriktionsstärke zu untersuchen, werden Testfälle generiert, bei denen die Länge des Workflows auf 15 Prozessschritte und die Anzahl der Prozessschrittkandidaten auf 40 Web Services festgelegt werden. Die Web Services werden, wie in Tabelle 9 dargestellt, durch vier nicht-funktionale Eigenschaften beschrieben. In der Zielfunktion wird mit der Gesamtantwortzeit nur ein Gesamtparameter gewichtet. Alle anderen in der Zielfunktion ungewichteten Gesamtparameter werden mit Restriktionen jeweils gleicher Restriktionsstärke belegt. Die Gesamtparameterwerte werden jeweils durch einen additiven, einen multiplikativen und einen Minimaloperator aggregiert, der nicht bereits durch die Gewichtung in der Zielfunktion optimiert wird. Die Testreihe wird gebildet, indem die Restriktionsstärke der restringierten Gesamtparameter von 20% auf 65% gesteigert wird.

Index	Parametername	Aggregationsoperator	korreliert mit
1	Antwortzeit	additiv	-
2	Preis	additiv	Antwortzeit
3	Verfügbarkeit	multiplikativ	-
4	Maximaler Durchsatz	Minimaloperator	-

Tabelle 9: Parameter zur Beschreibung eines Web Service

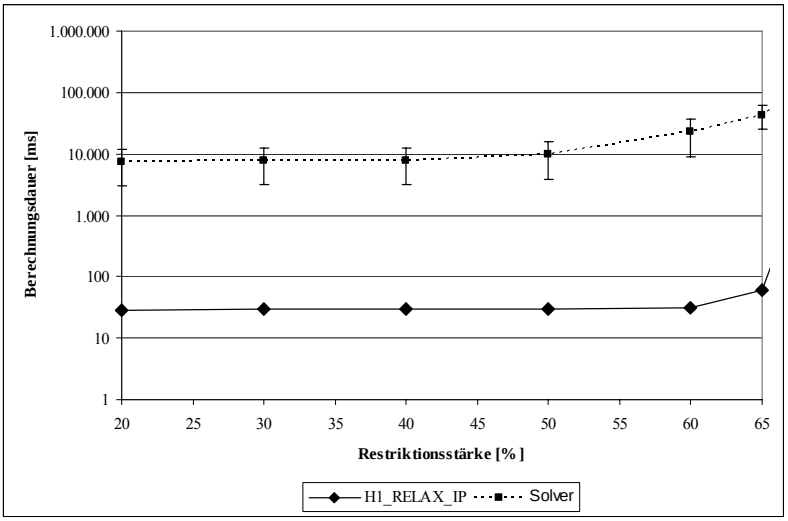
Wird ein Gesamtparameter in der Zielfunktion gewichtet und gleichzeitig einer Restriktion unterworfen, so führt ein höherer Zielerreichungsgrad der Optimierung des Gesamtparameters auch zu einem höheren Zielerreichungsgrad der Einhaltung der Restriktion bezüglich dieses Gesamtparameters. Wird beispielsweise die Gesamtantwortzeit optimiert und gleichzeitig gefordert, dass die Gesamtantwortzeit eine gewisse Schranke nicht überschreiten darf, so führt die Optimierung der Gesamtantwortzeit automatisch auch zur besseren Einhaltung der Schranke. Durch die Einführung von Restriktionen auf genau den Gesamtparametern, die nicht in der Zielfunktion gewichtet werden, wird die Problemkomplexität gesteigert, da eine Optimierung des Zielfunktionswertes nicht automatisch mit einer besseren Erfüllung der Restriktionen einhergeht. Im hier untersuchten Beispiel ist der Preis eines Web Service mit seiner Antwortzeit negativ korreliert, d. h., ein Web Service mit niedriger Antwortzeit besitzt tendenziell einen hohen Preis, weshalb durch die Optimierung der Gesamtantwortzeit und der Restriktion des Gesamtpreises ein Zielkonflikt entsteht. Durch die Optimierung der Antwortzeit wird versucht, Lösungen zu finden, die eine möglichst geringe Gesamtantwortzeit aufweisen. Jedoch sind diese Lösungen sehr teuer, was in vielen Fällen zu einer Verletzung der Restriktion des Gesamtpreises führt.

Die sich bei der Lösung der Testfälle durch den Solver und die Heuristik ergebenden Laufzeiten und Zielfunktionswerte können Tabelle 10 entnommen werden. Diese ist wie Tabelle 7 aufgebaut.

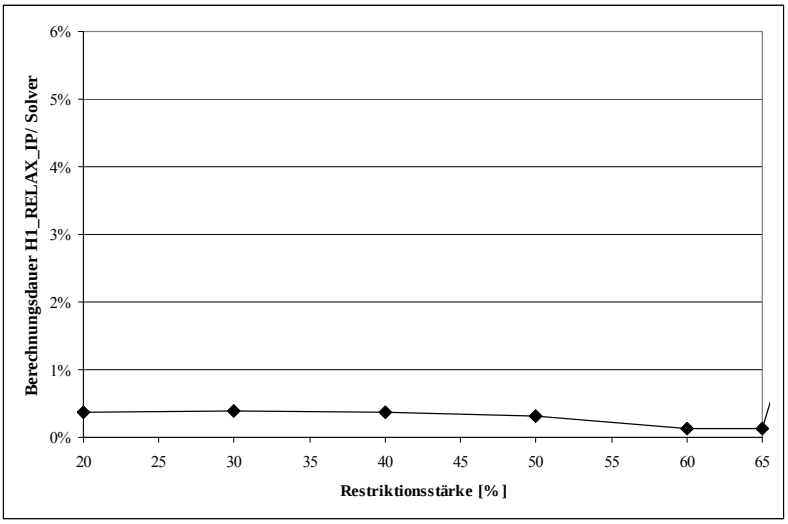
Restriktions-stärke [%]	Solver t_s [ms]	H1_RELAX_IP t_h [ms]	$\frac{t_h}{t_s}$ [%]	$\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$ [%]
20	7.585,16	28,48	0,38	99,29
30	7.674,93	29,23	0,38	99,29
40	7.888,81	29,93	0,38	99,29
50	9.632,88	30,17	0,31	99,16
60	22.996,15	30,54	0,13	98,96
65	43.145,07	59,15	0,14	98,92

Tabelle 10: Evaluation der Restriktionsstärke

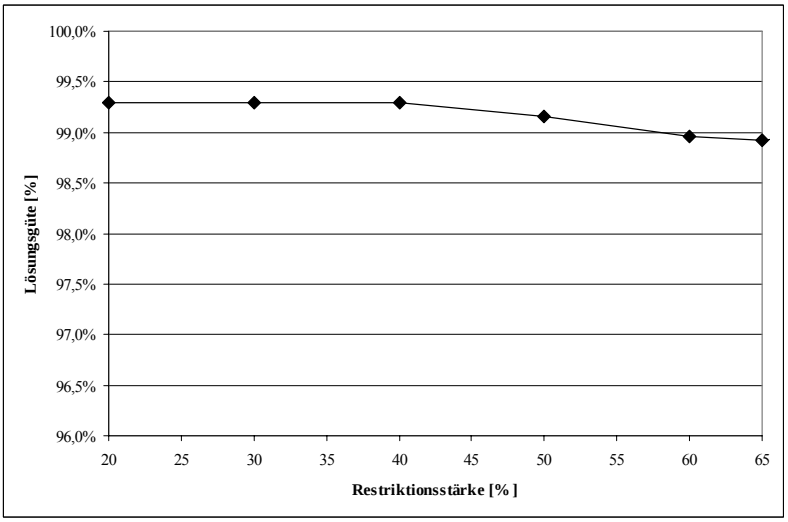
In Abbildung 33a wird die Rechendauer der Heuristik mit der des Solvers verglichen (95%-Konfidenzintervall). Durch die Erhöhung der Restriktionsstärke steigt sowohl die Laufzeit des Solvers als auch die der Heuristik an. Der Anstieg der Laufzeit des Solvers fällt jedoch wesentlich höher aus als die der Heuristik, wodurch sich der Laufzeitvorteil der Heuristik gegenüber dem Solver mit zunehmender Restriktionsstärke erhöht. Besonders deutlich wird dies im Fall der Erhöhung der Restriktionsstärke von 50% auf 65%. Die Laufzeit des Solvers steigt hier von ca. 9.600 ms auf ca. 43.000 ms an (ca. Faktor 4,5). Die Laufzeit der Heuristik steigt hingegen nur um Faktor 2 von ca. 30 ms auf ca. 60 ms. In diesem Fall benötigt die Heuristik nur 0,32% der Laufzeit des Solvers, liefert aber eine Lösung, deren Zielfunktionswert 98,92% des Zielfunktionswertes der optimalen Lösung beträgt. Das Laufzeitverhalten der Heuristik H1_RELAX_IP ist bei jeder Restriktionsstärke besser als das des Solvers (Abbildung 33a). So benötigt die Heuristik in allen Szenarien weniger als 1% der Laufzeit des Solvers (Abbildung 33b) und erreicht immer eine Lösungsgüte von über 98,9% (siehe Abbildung 33c).



(33a) Rechendauer von H1_RELAX_IP und Solver



(33b) Rechendauer von H1_RELAX_IP relativ zum Solver



(33c) Lösungsgüte von H1_RELAX_IP

Abbildung 33: Evaluierung der Restriktionsstärke

4.6.4 Evaluation der Verbesserungsverfahren

Um zu evaluieren, ob die durch H1_RELAX_IP errechneten Lösungen noch weiter verbessert werden können, wurden die zwei zuvor beschriebenen Verbesserungsverfahren H2_SWAP (vgl. Abschnitt 4.4.3) und H3_SIMUL_ANNEAL (vgl. Abschnitt 4.4.4) simuliert.

H1_RELAX_IP liefert Ergebnisse, die sehr nahe an das Optimum kommen. Die beiden Verbesserungsverfahren erreichen daher nur eine marginale Steigerung des von H1_RELAX_IP berechneten Zielfunktionswertes. Dies wird exemplarisch am Einfluss der Restriktionsstärke gezeigt (siehe Tabelle 11). So trägt eine Ausführung von H2_SWAP maximal zu 0,35% des endgültigen Zielfunktionswerts bei. Dies ist aber mit einem entsprechenden Laufzeitanstieg um Faktor 4 verbunden. Die Verbesserungsverfahren H2_SWAP und H3_SIMUL_ANNEAL haben ihre Existenzberechtigung in Szenarien, bei denen es von großer Bedeutung ist, noch näher an das Optimum zu kommen als H1_RELAX_IP. Hierbei ist die Rechenzeit, welche

durch die sukzessive Ausführung der Heuristiken H2_SWAP und H3_SIMUL_ANNEAL benötigt wird, immer noch um ein Vielfaches geringer als die Rechenzeit des exakten Verfahrens.

Restriktions-stärke [%]	Anstieg Rechendauer verglichen mit H1_RELAX_IP [%]	Beitrag am Zielfunktionswert [%]		
		H1_RELAX_IP	H2_SWAP	H3_SIMUL_ANNEAL
20	196,85	99,83	0,17	0,00
30	197,47	99,90	0,10	0,00
40	208,94	99,86	0,14	0,00
50	235,69	99,81	0,12	0,06
60	404,28	99,59	0,35	0,06
65	500,84	99,61	0,24	0,15

Tabelle 11: Auswertung der Verbesserungsverfahren

4.7 Zusammenfassung

In diesem Kapitel wird die Zuordnung konkreter Web Services zu abstrakten Prozessschritten als ein Optimierungsproblem modelliert. Zur Lösung dieses NP-schweren Optimierungsproblems wurden Heuristiken entwickelt, implementiert und simuliert. Die Simulation wurde mithilfe einer dafür entwickelten Simulationsumgebung durchgeführt. Bei der Evaluierung der Heuristiken wurde der Einfluss der Größen Workflowlänge, Anzahl der Prozessschrittkandidaten und Restriktionsstärke auf die Laufzeit und die Lösungsgüte der Heuristiken untersucht. Das Laufzeitverhalten und die Lösungsgüte der Heuristiken werden mit der exakten Lösung des Optimierungsproblems verglichen. Unabhängig von der untersuchten Einflussgröße weist die Heuristik einen erheblichen Laufzeitvorteil gegenüber der exakten Lösung des Optimierungsproblems durch den Solver auf. Der Laufzeitvorteil der Heuristik verstärkt sich zudem mit der Zunahme der Einflussgröße merklich. So konnte bei einer Workflowlänge von 21 Prozessschritten die Heuristik H1_RELAX_IP das Ergebnis bei einer Lösungsgüte von über 99% um Faktor 6.100 schneller liefern als der Solver. Dies entspricht im konkreten Fall einer Zeitersparnis von mehr als 4 Minuten. Hieran wird die Überlegenheit der Heuristik gegenüber der exakten Lösung durch den Solver in Bezug auf die Laufzeit deutlich. Darüber hinaus zeigen die hier erzielten Ergebnisse, dass die Lösung solcher Optimierungsprobleme in zeitkritischen Anwendungen (z. B. aufgrund von Nutzerinteraktion) nur durch heuristische Lösungsverfahren möglich ist. Die Lösungsgüte der Heuristik liegt in allen Testreihen bezüglich des Einflusses der Problemgröße bei über 98%, d. h. die Heuristik erreicht bei erheblich kürzerer Laufzeit einen Zielfunktionswert von über 98% des Zielfunktionswertes der exakten, optimalen Lösung. Das Verhältnis aus benötigter Rechenzeit und erreichter Lösungsgüte kann daher als sehr gut bewertet werden.

5 Ein Replanning-Mechanismus für Web Service Workflows

Das tatsächliche Laufzeitverhalten von Web Services weicht in vielen Fällen von dem ursprünglich geplanten ab. Gründe hierfür sind u. a. in der Unzuverlässigkeit externer Serviceanbieter und der Unzuverlässigkeit des Internets als Kommunikationskanal zu sehen. So kann z. B. die Antwortzeit von Web Services wegen Serverausfällen oder überlasteten Verbindungen länger sein als erwartet. In Extremfällen ist davon auszugehen, dass einzelne Web Services überhaupt nicht mehr verfügbar sind. Besteht zwischen Serviceanbieter und Servicenutzer ein Vertragsverhältnis über die Leistungserbringung der Services, z. B. in Form eines SLAs (vgl. Abschnitt 2.4.3), führen Ausfälle oder Verzögerungen möglicherweise zu Konventionalstrafen. Um dies zu vermeiden, legen Serviceanbieter den Werten in ihren SLAs in vielen Fällen konservative Schätzungen zugrunde, die mit einer hohen Wahrscheinlichkeit erfüllt werden können. Daher ist davon auszugehen, dass in solchen Fällen das tatsächliche Laufzeitverhalten der Web Services meist besser sein wird als in den entsprechenden SLAs geplant.

In dieser Arbeit wird ein Replanning-Mechanismus [15, 17] vorgestellt, der zur Ausführungszeit des Workflows das tatsächliche Laufzeitverhalten bereits ausgeführter Web Services bei der Auswahl noch nicht aufgerufener Web Services berücksichtigt. Dadurch wird sichergestellt, dass sowohl die Präferenzen des Nutzers als auch die von ihm definierten Restriktionen trotz abweichenden Laufzeitverhaltens berücksichtigt werden. Als Proof-of-Concept wird das Replanning-Verfahren auf Basis der in dieser Arbeit entwickelten Heuristik H1_RELAX_IP (vgl. Abschnitt 4.4.2) implementiert sowie dessen Laufzeitverhalten und die durch Replanning realisierten Kosteneinsparungen simuliert.

Dieses Kapitel ist wie folgt gegliedert: In Abschnitt 5.1 wird die Problemstellung beschrieben, die den Einsatz von Replanning motiviert. Der Replanning-Mechanismus als solcher wird in Abschnitt 5.2 erläutert und seine Funktionsweise anhand eines Beispielszenarios verdeutlicht. In Abschnitt 5.3 wird dargestellt, wie Replanning-Strategien definiert und in BPEL4WS integriert werden können. Als Proof-of-Concept wird eine prototypische Umsetzung dieses Konzept in eine Open-Source-BPEL-Engine vorgestellt. Die Auswertung des Replanning-Verfahrens hinsichtlich Laufzeitverhalten und Kosteneinsparungen wird in Abschnitt 5.4 diskutiert. Als Lösungsverfahren wird die in Abschnitt 4.4.2 beschriebene Heuristik H1_RELAX_IP verwendet. In Abschnitt 5.5 werden die wesentlichen Ergebnisse dieses Kapitels zusammengefasst.

5.1 Problemstellung

Das tatsächliche Laufzeitverhalten von Web Services weicht häufig von dem ursprünglich geplanten ab. So bietet das Internet als Kommunikationskanal für extern bezogene Web Services keine garantierte Dienstgüte. Eine ungleiche Serverauslastung, zwischenzeitliche Serverausfälle und ein stark schwankendes Verkehrsaufkommen lassen kaum zuverlässige Prognosen über das tatsächliche Laufzeitverhalten von Web Services zu. Zudem basiert die Zuordnung konkreter Web Services zu den entsprechenden Prozessschritten auf den in den SLAs vertraglich garantierten Dienstgüteeigenschaften, die vor der Ausführung des Web Service

Workflows abgeschlossen worden sind. Diese in den SLAs festgeschriebenen Dienstgüteeigenschaften werden jedoch häufig von Anbietern, basierend auf konservativen Worst-Case-Schätzungen, berechnet. Auf diese Weise möchte sich der Web Service Anbieter vor einem Reputationsverlust und Konventionalstrafen schützen, die als Folge von SLA-Verletzungen drohen. In einer Vielzahl von Fällen ist das Laufzeitverhalten der Web Services daher besser als in den SLAs garantiert. Dennoch ist es notwendig, die Auswahl der Web Services auf Grundlage der SLAs zu treffen, da der Web Service Nutzer seinerseits häufig den gesamten Workflow als Service nach außen hin anbietet und hierfür ebenfalls Garantien bezüglich des Laufzeitverhaltens abgeben muss.

Um auf die Abweichung zwischen Planwerten und dem tatsächlichen Laufzeitverhalten zu reagieren, wird in diesem Kapitel ein Replanning-Mechanismus beschrieben, der sicherstellt, dass der Web Service Workflow auch zur Laufzeit *ausführbar, gültig* hinsichtlich der Restriktionen seitens des Nutzers und *optimal* bezüglich der Präferenzstruktur des Nutzers bleibt. Im Einzelnen bedeutet dies:

- Replanning stellt sicher, dass der Web Service Workflow *ausführbar* bleibt. Wird ein Web Service aufgerufen, der zwischenzeitlich, z. B. aufgrund eines Serverausfalls, nicht mehr verfügbar ist, würde auch die Ausführung des gesamten Workflows scheitern. Ein Replanning-Mechanismus schafft hier Abhilfe. Nachdem durch einen Monitoring-Mechanismus, wie z. B. die Monitoring-Komponente von WSQoSX (vgl. Abschnitt 3.4.5), die Nichtverfügbarkeit des entsprechenden Web Service erkannt wurde, kann durch Replanning ein anderer Web Service aufgerufen werden, der die gleiche Funktionalität bereitstellt. Nahe liegend ist, dass im Falle einer lokalen Optimierungsstrategie derjenige Web Service aufgerufen wird, der die zweithöchste Bewertung in der entsprechenden Kategorie erhalten hat. Wird eine globale Optimierungsstrategie angewendet, wird das zugrunde liegende Optimierungsproblem durch Anpassung der Nebenbedingungen für den noch nicht ausgeführten Teil des Workflows erneut berechnet.
- Die Anwendung von Replanning gewährleistet, dass der Web Service Workflow *gültig* hinsichtlich der Restriktionen seitens des Nutzers ist. Im Falle einer hohen Netzauslastung ist es möglich, dass die Antwortzeit eines Web Service länger ist, als ursprünglich vor der Ausführung geplant. Eine Folge kann die Verletzung von Restriktionen (z. B. eine Überschreitung der Gesamtantwortzeit des Workflows) sein. Replanning kann in diesem Fall Abhilfe schaffen, indem es für den noch nicht ausgeführten Teil des Workflows solche Web Services auswählt, deren Antwortzeiten geringer sind als die der ursprünglich geplanten. Dafür wird das zugrunde liegende Optimierungsproblem mit veränderten Nebenbedingungen neu gelöst.
- Replanning ermöglicht, dass der Web Service Workflow *optimal* bezüglich der Präferenzen des Nutzers ist. Wenn das tatsächliche Laufzeitverhalten der Web Services besser ist, als ursprünglich auf Grundlage der SLAs geplant, dann sind zwar weiterhin alle Restriktionen erfüllt, jedoch ist der Ausführungsplan nicht mehr zwangsweise optimal hinsichtlich der Nutzerpräferenzen. Dies lässt sich an folgendem Beispiel ver-

deutlichen: Ein Nutzer präferiert kostengünstige Web Services mit der Restriktion, dass eine gewisse Gesamtantwortzeit des Workflows nicht überschritten werden darf. Es wird unterstellt, dass Web Services mit höherer Antwortzeit geringere Kosten verursachen als solche mit kurzer Antwortzeit. Vor der Ausführung des Workflows wird ein Ausführungsplan errechnet, der hinsichtlich der Nutzerpräferenzen optimal ist und alle Restriktionen berücksichtigt. Ist das Laufzeitverhalten der Web Services nun besser als erwartet, dann ist der Ausführungsplan nicht mehr zwangsläufig optimal hinsichtlich der Nutzerpräferenzen, da im noch nicht ausgeführten Teil kostengünstigere Web Services ausgewählt werden könnten, ohne Restriktionen zu verletzen. Durch Replanning können nun für den noch nicht ausgeführten Teil des Workflows Web Services ausgewählt werden, welche kostengünstiger sind als solche, die vor der Ausführung gewählt wurden. So kann durch Replanning sichergestellt werden, dass der Workflow wieder optimal hinsichtlich der Nutzerpräferenzen ist, ohne dass hierbei Restriktionen verletzt werden.

5.2 Funktionsweise

Die Zuordnung konkreter Web Services zu den abstrakten Prozessschritten eines Workflows wird, wie in Abschnitt 4.3 beschrieben, als ein Optimierungsproblem modelliert, das explizit die Präferenzstruktur des Nutzers und dessen Anforderungen an den Workflow berücksichtigt.

Nach Ausführung eines Web Service an der Position *i* des Ausführungsplans eines Web Service Workflows wird dieser in zwei Partitionen aufgeteilt (siehe Abbildung 34): Der erste Teil enthält alle Prozessschritte i' ($i' \leq i$), die bereits ausgeführt wurden, während im zweiten Teil die noch auszuführenden Prozessschritte i'' ($i'' > i$) zusammengefasst sind.

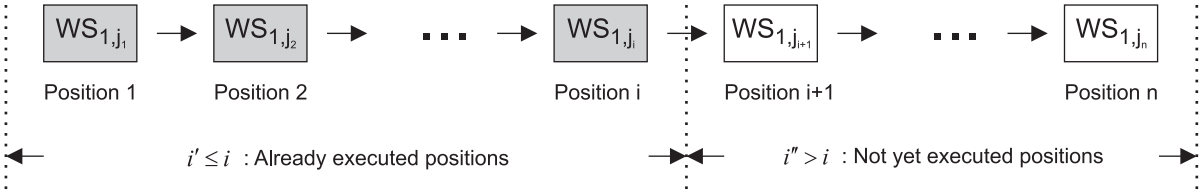


Abbildung 34: Partitionierung des Ausführungsplans

Nach Ausführung des *i*-ten Web Service werden mithilfe der entsprechenden Aggregationsfunktionen (vgl. Abschnitt 4.3) die aktuellen Gesamtparameterwerte berechnet. Diese Gesamtparameterwerte werden verwendet, um die Nebenbedingungen für den noch auszuführenden Teil anzupassen. Das so modifizierte Optimierungsproblem kann nun durch ein geeignetes Verfahren, z. B. durch die im Rahmen dieser Arbeit entwickelte Heuristik H1_RELAX_IP, gelöst werden. Dies führt zu einem neuen Ausführungsplan für den noch nicht ausgeführten Teil *i"* des Workflows. Durch Anwenden von Replanning entsteht so ein neuer Ausführungsplan, der sowohl den Präferenzen des Nutzers entspricht als auch dessen Restriktionen erfüllt. Dieser Ausführungsplan wird nun für die weitere Ausführung des Web Service Workflows zugrunde gelegt.

Die Funktionsweise des Replanning-Mechanismus wird im Folgenden anhand eines Beispiels (siehe Abbildung 35) verdeutlicht. Web Services werden durch zwei additive Parameter Antwortzeit $p_{i,j,1}$ und Kosten pro Aufruf $p_{i,j,2}$ beschrieben. Die Gesamtantwortzeit x_1^+ ist auf $c_1^+ = 2.000$ ms restringiert. Die Minimierung des Gesamtkosten x_2^+ ist das einzige Optimierungskriterium (ausgedrückt durch $w_1^+ = 0$ und $w_2^+ = -1$). Vor der Ausführung des Workflows errechnet die Workflow-Engine einen Ausführungsplan, der die Gesamtkosten minimiert ($x_2^+ = 0{,}13$\$) und die Nebenbedingung einhält ($x_1^+ = 1.950$ ms < $c_1^+ = 2.000$ ms). Der der ersten Aktivität im Ausführungsplan zugeordnete Web Service $WS_{1,3}$ wird ausgeführt. Seine tatsächliche Antwortzeit ist mit 475 ms geringer als auf Basis der SLA mit 600 ms geplant. Würde der Workflow basierend auf dem aktuellen Ausführungsplan weiter abgearbeitet und das Laufzeitverhältnis aller noch nicht ausgeführten Web Services würde nicht vom geplanten abweichen, dann wäre mit einer Gesamtantwortzeit von 1.825 ms zu rechnen.

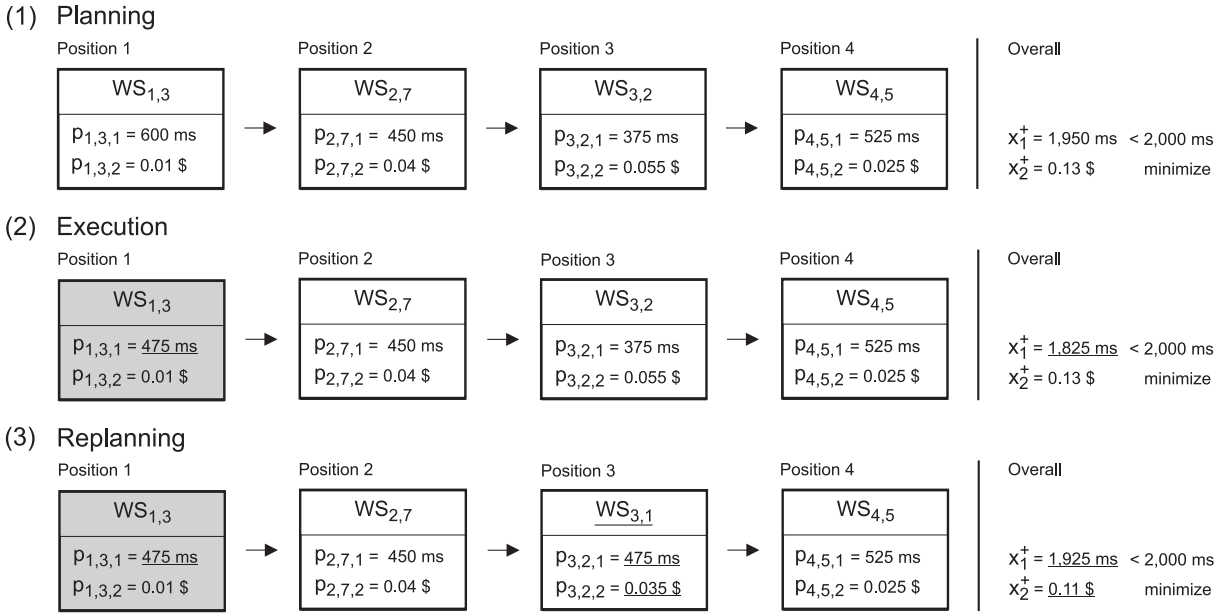


Abbildung 35: Beispielszenario für Replanning

Allerdings ist der aktuelle Ausführungsplan nicht mehr zwangsläufig optimal hinsichtlich des Optimierungskriteriums "Minimierung der Gesamtkosten". Um dies aber sicherzustellen, wird durch Replanning ein neues Optimierungsproblem für den noch nicht ausgeführten Teil des Workflows gelöst, welche das tatsächliche Laufzeitverhalten bereits ausgeführter Web Services mit einbezieht. Dies wird dadurch realisiert, dass die Nebenbedingung des Optimierungsproblems, in diesem Beispiel die Gesamtantwortzeit, angepasst wird auf $c_1^+ = 2.000$ ms – 475 ms = 1.525 ms. Das auf diese Weise modifizierte Optimierungsproblem wird nun mit der Heuristik H1_RELAX_IP gelöst. Da der bereits ausgeführte Web Service eine kürzere Antwortzeit als erwartet benötigt hat, wird durch Ausführen der Heuristik H1_RELAX_IP an Position 3 der Web Service $WS_{3,2}$ durch den langsameren, aber billigeren Web Service $WS_{3,1}$ ersetzt. Der so neu errechnete Ausführungsplan hat eine Gesamtantwort-

zeit von x_1^+ = 1.925 ms, seine Gesamtkosten x_2^+ = 0,11$ sind aber geringer als die des ursprünglich kalkulierten Ausführungsplans, ohne dass die Nebenbedingung verletzt wird.

5.3 Integration von Replanning in eine Workflow-Engine

In diesem Abschnitt wird beschrieben, wie Replanning in die Workflow-Sprache BPEL4WS integriert werden kann. Hierzu wird in Abschnitt 5.3.1 WSQoSX um den *Adaptability Manager (AM)* erweitert, der das Replanning von Web Service Workflows ermöglicht. Die XML-basierte Modellierung von Replanning-Strategien ist Gegenstand von Abschnitt 5.3.2. In Abschnitt 5.3.3 werden *Event-Condition-Action (ECA)*-Regeln zur Umsetzung des Replanning-Verfahrens vorgeschlagen. Des Weiteren wird als Proof-of-Concept eine prototypische Umsetzung basierend auf ECA-Regeln anhand der *ActiveBPEL-Workflow-Engine*[5] in Abschnitt 5.3.4 vorgestellt.

5.3.1 Architektur

Abbildung 36 zeigt die um den Adaptability Manager erweiterte Web Service Architektur WSQoSX (vgl. Abschnitt 3.4).

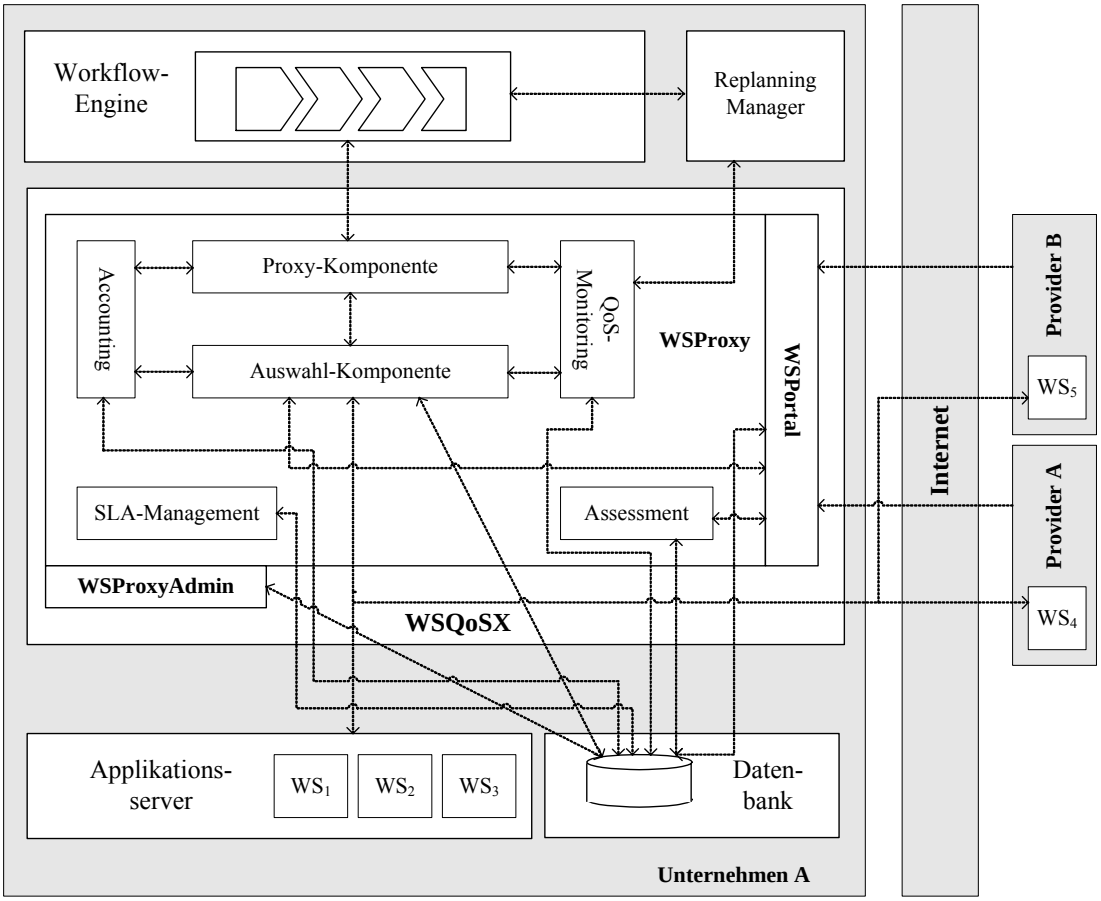


Abbildung 36: Der Adaptability Manager als Architekturkomponente für Replanning

[5] Bei ActiveBPEL handelt es sich um eine Open-Source-Workflow-Engine des Unternehmens Active Endpoints (http://www.activebpel.org/). ActiveBPEL ist konform zu den Standards BPEL4WS 1.1 und WS-BPEL 2.0.

Der Adaptability Manager stellt die in Abschnitt 5.2 beschriebene Replanning-Funktionalität bereit. Die hierfür vom Adaptability Manager benötigten Informationen werden von der QoS-Monitoring-Komponente gemessen und in der Datenbank abgelegt.

5.3.2 Modellierung von Replanning-Strategien

Der in diesem Abschnitt vorgestellt Ansatz nutzt die Erweiterbarkeit der BPEL4WS-Spezifikation, um Replanning in BPEL4WS zu integrieren. Durch das `qos_rt_adaptability`-Element wird die BPEL4WS-basierte Workflow-Beschreibung erweitert, sodass der Nutzer individuelle Replanning-Strategien konfigurieren kann. Eine Replanning-Strategie beschreibt die Art und Weise, wie Replanning angewendet werden soll und beinhaltet folgende Auswahlmöglichkeiten:

- Die Replanning-Strategie definiert, ob der Workflow *lokal* oder *global* optimiert werden soll. Eine lokale Optimierung bedeutet, wie auch im Rahmen von WSQoSX umgesetzt (vgl. Kapitel 3), dass innerhalb einer Gruppe funktional identischer Web Services derjenige ausgewählt wird, der den Präferenzen des Nutzers am besten entspricht. Dem hingegen ermöglicht eine globale Optimierung, dass die Auswahl der Web Services so erfolgt, dass Präferenzen und Restriktionen seitens des Nutzers an den gesamten Workflow, und nicht nur auf eine bestimmte Kategorie bezogen, berücksichtigt werden.
- Zudem legt die Replanning-Strategie fest, *wann* ein Replanning ausgeführt werden soll. So kann Replanning z. B. nach jeder Ausführung eines Web Service, nur bei Auftreten eines Fehlers oder aber bei sich abzeichnenden SLA-Verletzungen durchgeführt werden.
- Die Replanning-Strategie beschreibt auch den *Bereich* des Workflows, auf den Replanning angewendet werden soll. In diesem Zusammenhang kann der Nutzer entscheiden, dass Replanning beispielsweise aus Gründen der Rechenzeit nur bei kritischen Aktivitäten ausgeführt wird.
- Durch die Replanning-Strategie kann zudem bestimmt werden, welches *konkrete Verfahren* zur Umsetzung des Replanning zum Einsatz kommen soll. Diese Entscheidung hat Auswirkungen auf die Lösungsgüte und das Laufzeitverhalten des Replanning-Verfahrens. In diesem Zusammenhang kann festgelegt werden, ob exakte oder heuristische Lösungsverfahren eingesetzt werden. Entscheidet sich der Nutzer aufgrund des besseren Laufzeitverhaltens für heuristische Lösungsverfahren, kann er auswählen, ob diese z. B. auf genetischen Algorithmen oder auf einem Backtracking-Verfahren beruhen. Zudem können neben Eröffnungs- auch Verbesserungsverfahren ausgesucht werden.

Dieser Ansatz hat den Vorteil, dass ein auf diese Weise erweiterter BPEL4WS-Workflow auch von Workflow-Engines ausgeführt werden kann, welche die Erweiterung in Form des `qos_rt_adaptability`-Element nicht unterstützen, da es in diesem Falle ignoriert wird. Zudem wird eine Semantik zur Beschreibung einer Replanning-Strategie definiert, die aber

keine Aussagen über die tatsächlich verwendete Umsetzung und Implementierung trifft. Die Replanning-Strategie lässt dem Nutzer z. B. freie Wahl, welche Implementierung eines auf genetischen Algorithmen basierenden Verfahrens zum Einsatz kommt.

Abbildung 37 zeigt ein Beispiel für eine in BPEL4WS integrierbare Replanning-Strategie:

```
<?xml version="1.0" encoding="UTF-8"?>

<process name="BP_A" suppressJoinFailure="yes" targetNamespace="http://BP_A"
xmlns:adapt="http://www.physikus.org/def/Adaptability_Schema">

       <scope name="scope_A1">

              <!-- elements of scope A1 -- >

       </scope>

     <adapt:qos_rt_adaptability name="Gold_Service_Offering_Adapt"
     xmlns:tns="http://BP_A">

       <scope name="tns:BP_A">

              <binding selection_policy="PID004" optimization="global"/>

              <replanning enabled="true"/>

       </scope>

       <scope name="tns:scope_A1">

              <binding selection_policy="PID003" optimization="local"/>

              <rebinding enabled="true"/>

       </scope>

     </adapt:qos_rt_adaptability>

</process>
```

Abbildung 37: Beispiel für eine Replanning-Strategie

5.3.3 Ein ECA-Mechanismus zur Umsetzung der Replanning-Strategien

Im vorherigen Abschnitt wurde beschrieben, wie Replanning-Strategien als Erweiterung der BPEL4WS-Spezifikation modelliert werden. Um diese auszuführen, muss eine BPEL4WS-Workflow-Engine entsprechend angepasst und erweitert werden. In diesem Abschnitt wird daher konzeptionell gezeigt, wie das Event Condition Action (ECA)-Paradigma verwendet werden kann, um Replanning in eine BPEL4WS-Workflow-Engine zu integrieren. Als Proof-of-Concept wird eine prototypische Erweiterung der ActiveBPEL-Workflow-Engine um einen Replanning-Mechanismus diskutiert.

Der ECA-Ansatz wurde Ende der 80er Jahre erstmals vorgestellt und ausführlich im Zusammenhang mit aktiven Datenbanken untersucht und angewendet, z. B. [109]. ECA-Regeln kommen vorwiegend in Systemen zum Einsatz, die ein reaktives Verhalten spezifizieren. ECA-Regeln sind grundsätzlich wie folgt aufgebaut:

```
On <event> if <condition> then <action>
```

Tritt ein bestimmtes Ereignis (engl. *Event*) ein und ist eine zugehörige Bedingung (engl. *Condition*) erfüllt, dann wird eine entsprechende Aktion (engl. *Action*) durchgeführt.

Im Zusammenhang mit Replanning stellt jede Ausführung eines Prozessschrittes ein Ereignis dar. Der AM überprüft dann, ob eine Bedingung erfüllt ist. Mögliche Bedingungen sind z. B. eine Abweichung vom ursprünglich geplanten Laufzeitverhalten oder die Verletzung von Laufzeiteigenschaften, die in einem SLA zugesichert werden. Ist eine solche Bedingung erfüllt, dann wird die zugehörige Aktion ausgeführt. Ein Beispiel für eine mögliche Aktion kann das Ausgeben von Warnmeldungen und die Ausführung des entsprechenden Replanning-Verfahrens sein. Die ECA-Regeln als solche werden in einem Verzeichnis abgelegt und können von den Nutzern erstellt und bearbeitet werden.

Die Ausführung eines Prozessschrittes stellt, wie bereits erläutert, ein Ereignis dar. Die Laufzeiteigenschaften dieses Prozessschrittes werden durch eine Monitoring-Komponente ermittelt und dann in einer Datenbank abgelegt. Die dabei verwendete Datenstruktur enthält folgende Elemente:

- `TimeStamp`: Zeitpunkt, an dem das Event generiert wurde
- `ProcessName`: Name des Workflows
- `ScopeName`: Name des Bereiches
- `ActivityName`: Name des Prozessschrittes, der das Event ausgelöst hat
- `TryCount`: Anzahl der Versuche, den entsprechenden Web Service auszuführen
- `Status`: Gibt den Zustand des ausgeführten Web Service an. So bedeutet z. B. `finished`, dass der Web Service erfolgreich ausgeführt wurde, während `fault` auf eine fehlerhafte Ausführung hinweist.
- Aktuelles Laufzeitverhalten: Die unter diesem Punkt subsumierten Werte beschreiben das gemessene Laufzeitverhalten des entsprechenden Web Service.

Abbildung 38 zeigt die XML-basierte Modellierung einer ECA-Regel, die bei Einhaltung der Bedingung als Aktion die entsprechende Replanning-Strategie auslöst:

```
<rule parameter=”append" create=“single” name=”response_time_rule”>
       <description>verifies response time of an executed activity</description>
       <condition-ref enabled=”yes”>response_time_condition</condition-ref>
       <action-ref enabled=”yes”>response_time_action</action-ref>
       <instance-limit>None</instance-limit>
</rule>
```

Abbildung 38: Modellierung einer ECA-Regel

Die entsprechende Bedingung ist Abbildung 39 zu entnehmen. Diese besagt, dass ein Replanning ausgeführt wird, sobald die Antwortzeit 25 ms überschreitet und der Serviceaufruf viermal gescheitert ist.

```
<condition-def always-true=”no” composite=“no” name=”response_time_condition”>
	<parameters>
		<parameter name=”response_time”/>
		<parameter name=“try”/>
	</parameters>
	<expressions>
		<expression parameter=”response_time”>25</expression>
		<expression parameter=”try”>4</expression>
		<expression>and</expression>
	</expressions>
</condition-def>
```

Abbildung 39: Modellierung einer Bedingung

5.3.4 Implementierung

Im Folgenden wird der Adaptability Manager als prototypische Implementierung einer um Replanning erweiterten Workflow-Engine beschrieben. Als Workflow-Engine wird die ActiveBPEL-Engine verwendet, da dies eine Open-Source-Software ist und sich somit für eine Erweiterung des Funktionsumfangs im besonderen Maße eignet. Der Adaptability Manager ist in der Lage, eine nach Abschnitt 5.3.2 beschriebene Replanning-Strategie einzulesen und umzusetzen. Zur Modellierung der `Event`-Datenstruktur wird ein XML-basierter Syntax verwendet (siehe Abbildung 40).

```
<rulecore-event time="20060412 12:20:10" type="rebinding-event">
	<parameters>
		<parameter name="process" type="string">
			http://online_banking
		</parameter>
		<parameter name="scope" type="string">
			http://online_banking
		</parameter>
		<parameter name="activity" type="string">
			invoke_pdf_convert
		</parameter>
		<parameter name="try" type="number">1</parameter>
		<parameter name="status" type="string">FINISHED</parameter>
		<parameter name="response_time" type="number">29</parameter>
	</parameters>
</rulecore-event>
```

Abbildung 40: Parameter eines Replaning-Events

ActiveBPEL verwaltet die verschiedenen Funktionalitäten durch sog. Manager (z. B. `IAeQueueManager` und `IAeAlarmManager`). Der AM implementiert die `IAeManager`-Schnittstelle, die alle Methoden beschreibt, die ein Manager bei ActiveBPEL unterstützen muss, wie z. B. die `Create`-, `Start`- und `Stop`-Methode. Für jeden angemeldeten Prozess empfängt der AM ein `DeploymentEvent`. Durch die `handleDeploymentEvent`-Methode des `AdDeploymentListener` überprüft der AM, ob der angemeldete Prozess über eine Erweiterung durch das `qos_rt_adaptability`-Element verfügt. Ist dies der Fall, so überprüft der AM noch die Gültigkeit der zugewiesenen Werte. So kann z. B. das `replanning`-Element nur die Werte `true` oder `false` annehmen.

Nach erfolgreicher Prüfung der zugewiesenen Werte wird ein `AdAdaptabilityContext`-Objekt erstellt und in der `AdAdaptiveProcessesRegistry` abgelegt. Dieses Objekt enthält die im `qos_rt_adaptability`-Element definierten Werte. Zur Ausführung des Replanning-Mechanismus ruft der AM ein konkretes Verfahren auf, wie z. B. eine Implementierung von H1_RELAX_IP. Der AM erstellt des Weiteren einen Datensatz zu jeder Prozessinstanz, der Informationen über deren Laufzeitverhalten enthält. Dieser Datensatz wird über den gesamten Lebenszyklus der Prozessinstanz hinweg aktualisiert und enthält folgende Elemente:

- Auf Basis eines SLAs geplante Antwortzeit des Web Service
- Tatsächliche Antwortzeit des Web Service
- Berechnungszeit des Ausführungsplans
- Gesamtantwortzeit
- Aktueller Status des ausgeführten Web Service (`Finished` oder `Fault`)
- Einhaltung des SLAs (`true` oder `false`)

5.4 Evaluation

Im vorherigen Abschnitt wurde gezeigt, wie Replanning-Strategien in BPEL4WS integriert werden und wie eine Open-Source-BPEL4WS-Engine um einen Replanning-Mechanismus erweitert werden kann. In diesem Abschnitt wird der Replanning-Mechanismus bezüglich seines Laufzeitverhaltens und potenzieller Kosteneinsparungen evaluiert. Da das Replanning zur Laufzeit u. a. von Echtzeitanwendungen durchgeführt wird, muss das zugrunde liegende Lösungsverfahren hochperformant sein. Aus diesem Grunde wird das im Rahmen dieser Arbeit entwickelte heuristische Lösungsverfahren H1_RELAX_IP (vgl. Abschnitt 4.4.2) ausgewählt.

Dieser Abschnitt ist wie folgt gegliedert. Nach einer Beschreibung des Versuchsaufbaus in Abschnitt 5.4.1, werden die Ergebnisse der Evaluation des Replanning-Verfahrens basierend auf H1_RELAX_IP hinsichtlich Workflowlänge (siehe Abschnitt 5.4.2), Anzahl der Prozessschrittkandidaten (siehe Abschnitt 5.4.3) und Restriktionsstärke (siehe Abschnitt 5.4.4) diskutiert.

5.4.1 Versuchsaufbau

Um das Laufzeitverhalten und Vorteile von Replanning, wie z. B. Kosteneinsparungen, zu evaluieren, wurde ein Datengenerator entwickelt, der für jeden Parameter $p_{i,j,k}$ eines Web Service zwei Werte generiert. Bei dem ersten Wert handelt es sich um den Planwert p, der mit dem Web Service Anbieter im Rahmen eines SLAs ausgehandelt wurde. Der zweite Wert $p´$ gibt das tatsächliche Laufzeitverhalten des Web Service nach dessen Ausführung an. Der Wert p' wird berechnet als normalverteilte Zufallsvariable $X_{p,}$, die den Planwert p als Begrenzung zu einem bestimmten Konfidenzniveau besitzt. Der Wert $p´$ wird in vier Schritten berechnet. Zuerst wird ein gleichverteilter Abweichungsfaktor e aus einem Intervall $[\underline{e};\bar{e}]$ gezo-

gen. Anschließend wird der Abweichungsfaktor *e* vom Planwert *p* abgezogen und das Ergebnis dieser Subtraktion als Mittelwert $\mu_{p'}$ der Normalverteilung $X_{p'}$ festgelegt. Im nächsten Schritt wird die Standardabweichung $\sigma^2_{p'}$ der Normalverteilung $X_{p'}$ berechnet, sodass *p* die Grenze eines Konfidenzintervalls zum Konfidenzniveau $1-\alpha$ darstellt. Das tatsächliche Laufzeitverhalten *p´* wird als Realisierung einer Zufallsvariablen $X_{p'}$ ($N(\mu_{p'},\sigma^2_{p'})$-verteilt) ermittelt. Optional kann festgelegt werden, dass *p´* in einem bestimmten Intervall ($[\underline{l},\bar{l}]$) liegen soll. Ist *p´* nicht in diesem Intervall, wird der letzte Schritt so oft wiederholt, bis dies der Fall ist.

Die zur Auswertung des Replanning generierten Testfälle basieren auf Web Services, die durch die beiden nicht-funktionalen Eigenschaften Antwortzeit und Kosten (pro Aufruf) beschrieben werden. Der gleichverteilte Planwert wird aus einem Intervall mit den Grenzen 400 ms bis 2.000 ms bestimmt. Das tatsächliche Laufzeitverhalten weicht vom Planwert ab und wird auf Grundlage folgender Parameter berechnet: $\underline{e}$ = 5%, $\bar{e}$ = 25%, α = 5%, $\underline{l} = 40ms$, $\bar{l} = \infty$. Es wird unterstellt, dass die Kosten (pro Aufruf) $p_{i,j,2}$ mit der Antwortzeit $p_{i,j,1}$ des Web Service korreliert sind. Je länger die Antwortzeit $p_{i,j,1}$, desto niedriger sind die Kosten $p_{i,j,2}$. Die tatsächlichen Kosten pro Aufruf sind identisch zu den ursprünglich geplanten.

Als Optimierungskriterium wird die Minimierung der Gesamtkosten festgelegt, die Gesamtantwortzeit wird mit einer Nebenbedingung belegt. Um dem Optimierungskriterium zu genügen, sind bevorzugt Web Services mit geringen Kosten auszuwählen. Da diese jedoch relativ hohe Antwortzeiten haben, ist davon auszugehen, dass die ausschließliche Verwendung kostengünstiger Web Services zu einer Verletzung der Nebenbedingung führen würde. Um diesem Zielkonflikt gerecht zu werden, ist bei der Auswahl der Web Services eine Abwägung insofern zu treffen, als dass Web Services zwar möglichst preiswert sein sollten, aber zugleich auch eine hinreichend kurze Antwortzeit aufweisen müssen.

Um die Wirksamkeit des Replanning zu evaluieren, werden drei Testszenarien untersucht, in denen die Einflussgrößen Workflowlänge, Anzahl der Prozessschrittkandidaten und Restriktionsstärke variiert werden. Zur Untersuchung einer Einflussgröße werden jeweils 35 Simulationsdurchläufe ausgeführt, aus deren Durchschnitt dann das Ergebnis berechnet wird.

Die Simulation wurde auf einem PC mit einem Intel Pentium 4 Prozessor (3 GHz) sowie 1 GB Arbeitsspeicher unter dem Betriebssystem Microsoft Windows XP Professional durchgeführt. Die Implementierung der Replanning-Algorithmen wurde mit Java 2 Platform Standard Edition (J2SE) in der Version 5.0 erstellt und in die im Rahmen dieser Arbeit entwickelte Simulationsumgebung WorkflowOptimizer (vgl. Anhang, Abschnitt B) integriert.

5.4.2 Einfluss der Workflowlänge

Um den Einfluss der Workflowlänge auf den Replanning-Mechanismus zu untersuchen, wird diese in Fünferschritten von 5 bis 35 Prozessschritte erhöht. Die Anzahl der Prozessschritt-

kandidaten wird konstant auf 40 Web Services festgesetzt. Die Restriktionsstärke der Nebenbedingung auf der Gesamtantwortzeit beträgt 40%. Die Restriktionsstärke beschreibt hierbei den Einfluss der Nebenbedingungen, durch welche die Gesamtparameterwerte einer Lösung beschränkt werden. Für weitere Angaben zur Restriktionsstärke wird auf Abschnitt 4.6, S. 100 verwiesen. Der Replanning-Mechanismus wird nach jeder simulierten Abarbeitung eines Prozessschrittes ausgeführt.

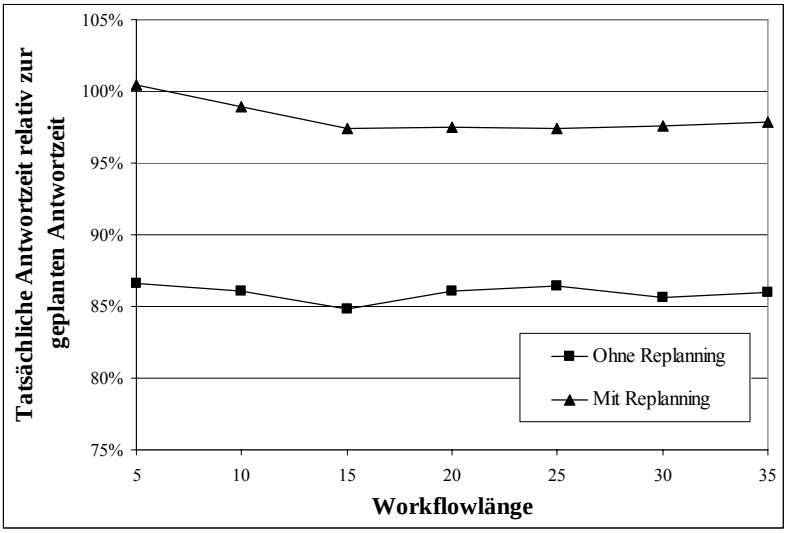
Zu Beginn wird mithilfe der Heuristik H1_RELAX_IP ein Ausführungsplan basierend auf den Planwerten (z. B. aus einem SLA) berechnet. Die Simulation dieses Ausführungsplanes ohne Replanning ergibt, dass die tatsächliche Gesamtantwortzeit des Workflows bei ungefähr 85% der zuvor geplanten Antwortzeit liegt (siehe Abbildung 41a). Dies ist darauf zurückzuführen, dass bei der Generierung der Daten eine konservative Schätzung der Planwerte unterstellt wurde. Daher ist das tatsächliche Laufzeitverhalten besser als vor der Ausführung geplant.

Allerdings ist dieser Ausführungsplan nicht mehr optimal bezüglich der Nutzerpräferenz „Minimiere die Gesamtkosten, ohne dabei eine bestimmte Gesamtantwortzeit überschreiten." Durch den Einsatz von Replanning hingegen können Web Services mit geringeren Kosten verwendet werden, ohne dass die Nebenbedingung auf der Antwortzeit verletzt wird. Durch den Einsatz kostengünstiger und langsamer Web Services steigt die Antwortzeit auf ca. 97% des ursprünglich geplanten Wertes. Da durch das Anwenden von Replanning langsamere, aber dafür kostengünstigere Web Services eingesetzt werden, können Kosteneinsparungen zwischen 23% und 26% erreicht werden (siehe Abbildung 41b). Für die Ausführung des Replanning-Mechanismus durch die Heuristik H1_RELAX_IP wird zusätzliche Rechenzeit benötigt. Wie in Abbildung 41c zu sehen, steigt diese mit zunehmender Workflowlänge an. Allerdings liegt die zusätzliche, durch Replanning bedingte Rechendauer auch bei einer Workflowlänge von 35 Prozessschritten unter 0,5 Sekunden. Abbildung 41d zeigt die erreichte Kostenreduktion pro zusätzlich benötigter Rechenzeit für Replanning. Da die Kostenreduktion mit zunehmender Workflowlänge geringer ansteigt als die hierfür benötigte Rechenzeit, sinkt die Effizienz des Replanning-Verfahrens mit zunehmender Workflowlänge leicht.

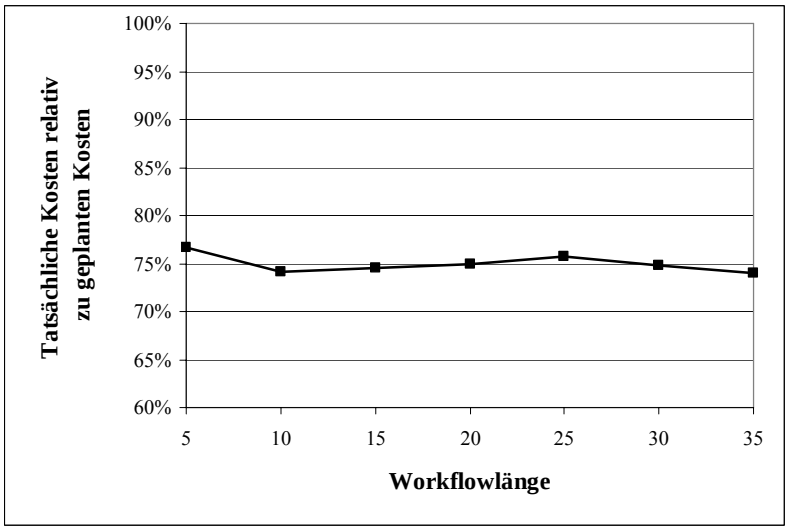
Die Ergebnisse der Simulation sind Tabelle 12 zu entnehmen. Diese zeigt die prozentuale Kosteneinsparung aufgrund des Replanning. Zudem wird der zusätzliche durch Replanning bedingte Rechenaufwand verglichen mit der zur Berechnung des ursprünglichen Ausführungsplans benötigten Rechenzeit. So wurde bei einer Workflowlänge von 5 Prozessschritten durch den im Rahmen dieser Arbeit vorgestellten Replanning-Mechanismus eine Kosteneinsparung von 23,4% erreicht. Die Berechnungszeit des ursprünglichen Ausführungsplans beträgt 4,83 ms, durch Replanning wurde die Berechnungsdauer um Faktor 1,62 erhöht. Das Verhältnis von Kosteneinsparung zu zusätzlichem Rechenaufwand beträgt 0,1442.

Workflowlänge	Kostenersparnis [%]	Rechendauer ursprünglicher Ausführungsplan [ms]	Zusätzliche Rechendauer Replanning [%]	Kostenersparnis [%] / Zusätzliche Rechendauer [%]
5	23,40	4,83	162,30	0,1442
10	25,90	9,49	274,98	0,0942
15	25,53	16,37	388,15	0,0658
20	25,08	23,20	478,99	0,0524
25	24,27	33,19	552,07	0,0440
30	25,18	43,74	640,05	0,0393
35	25,95	57,30	718,05	0,0361

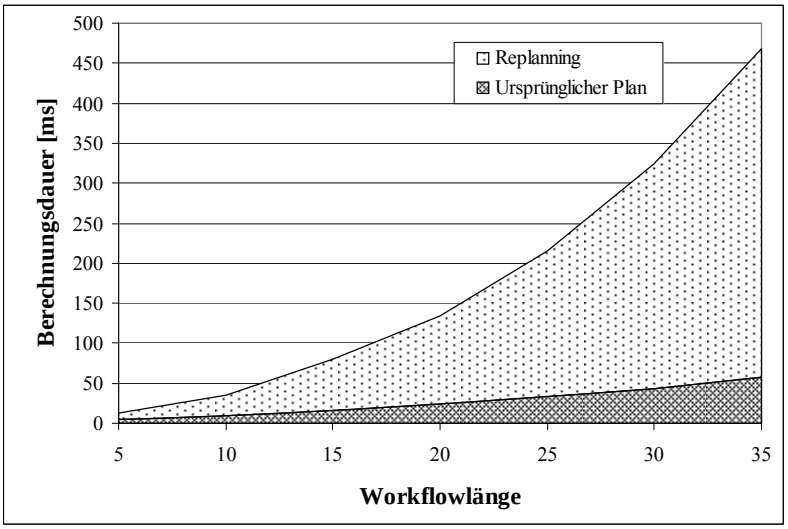
Tabelle 12: Evaluierung der Workflowlänge bei Replanning



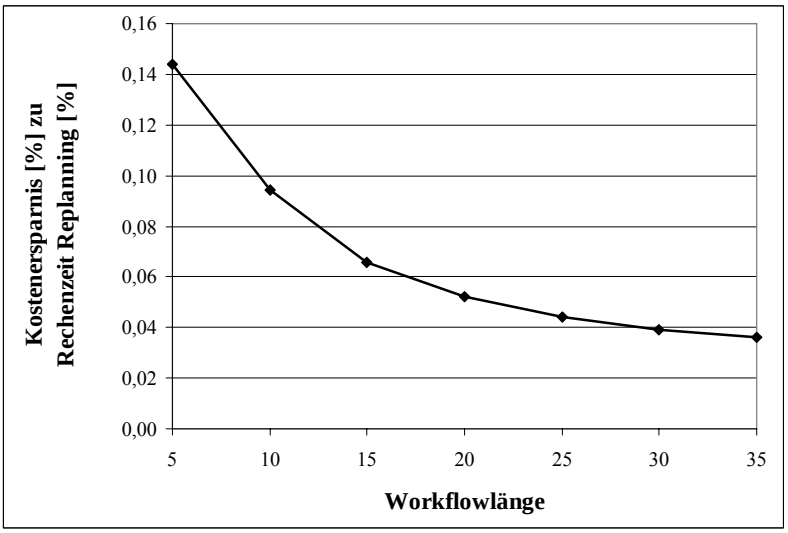
(41a) Antwortzeit des gesamten Workflow



(41b) Kosten



(41c) Rechendauer



(41d) Verhältnis Kostenersparnis zu zusätzlicher Rechendauer

Abbildung 41: Evaluierung der Workflowlänge bei Replanning

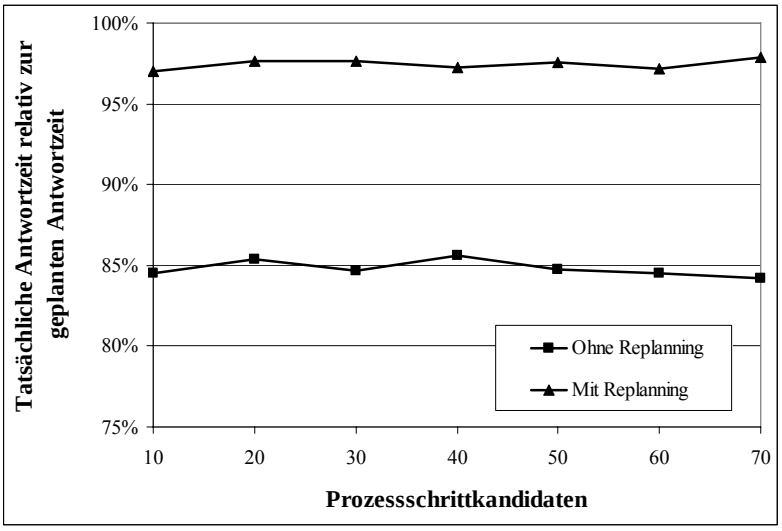
5.4.3 Einfluss der Prozessschrittkandidaten

In diesem Szenario wird der Einfluss der Anzahl der Prozessschrittkandidaten auf das Laufzeitverhalten und potenzielle Kosteneinsparungen im Rahmen von Replanning untersucht. Hierzu wird die Anzahl der Prozessschrittkandidaten von 10 auf 70 erhöht. Die Workflowlänge wird konstant auf 20 Prozessschritte festgelegt und die Antwortzeit ist durch eine Nebenbedingung der Restriktionsstärke 40% eingeschränkt. Nach jedem simulierten Prozessschritt wird ein Replanning durchgeführt. Die Ergebnisse können Tabelle 13 entnommen werden. Diese ist wie Tabelle 12 aufgebaut.

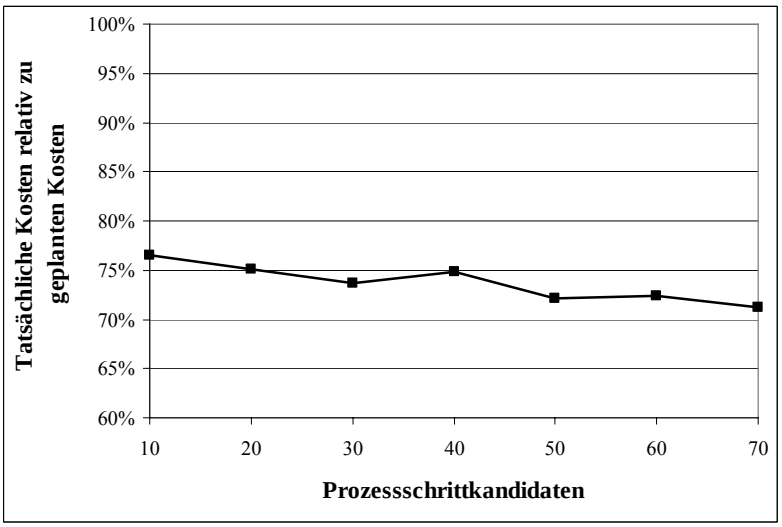
Prozess-schrittkan-didaten	Kosten-ersparnis [%]	Rechendauer ursprünglicher Aus-führungsplan [ms]	Zusätzliche Rechendauer Replanning [%]	Kostenersparnis [%] / Zusätzliche Rechen-dauer [%]
10	23,43	6,29	399,15	0,0587
20	24,88	11,69	391,51	0,0636
30	26,27	17,24	436,88	0,0601
40	25.19	23,73	464,02	0,0543
50	27,83	30,37	509,19	0,0547
60	27,60	38,49	534,39	0,0517
70	28,76	46,04	577,49	0,0498

Tabelle 13: Evaluierung der Anzahl der Prozessschrittkandidaten bei Replanning

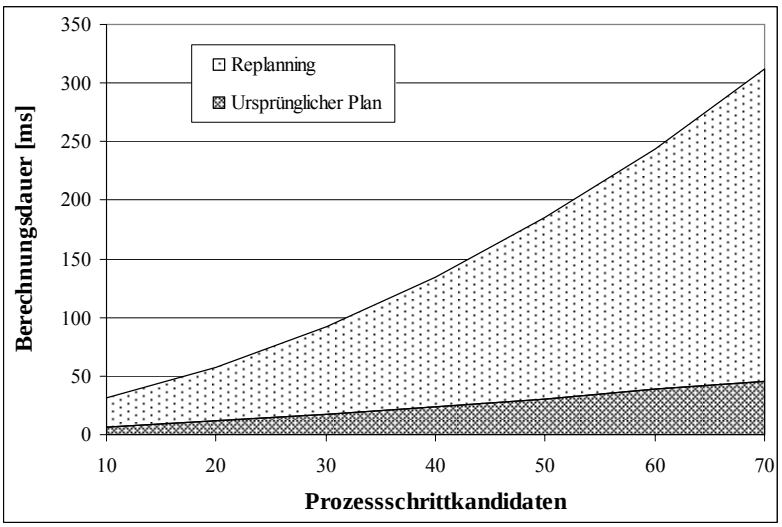
Ohne den Einsatz von Replanning liegt die tatsächliche Gesamtantwortzeit unabhängig von der Workflowlänge durchschnittlich 15% unter der ursprünglich geplanten. Dies ist darauf zurückzuführen, dass bei der Generierung des tatsächlichen Laufzeitverhaltens durch die Parameterauswahl festgelegt wurde, dass die Planwerte konservativ gewählt werden. Kommt Replanning zum Einsatz, dann können Kosteneinsparungen zwischen 23% und 29% realisiert werden (siehe Abbildung 42b). Dies wird dadurch erreicht, dass kostengünstige Web Services ausgewählt werden, ohne dass die Nebenbedingung bezüglich der Gesamtantwortzeit verletzt wird. Da kostengünstigere Web Services eine längere Ausführungsdauer benötigen, steigt die Gesamtantwortzeit auf 97% der ursprünglich geplanten, verglichen mit einer Antwortzeit von 85% ohne Replanning (siehe Abbildung 42a). Aus Tabelle 13 kann entnommen werden, dass mit einer steigenden Zahl von Prozessschrittkandidaten größere Kosteneinsparungen erreicht werden können. Dies ist darauf zurückzuführen, dass mit einer steigenden Anzahl potenzieller Web Services mehr Auswahlmöglichkeiten zur Verfügung stehen, auf die der Replanning-Mechanismus zurückgreifen kann, wenn das Optimierungsproblem für den noch nicht ausgeführten Teil des Workflows neu gelöst wird. So steigt mit zunehmender Anzahl an Prozessschrittkandidaten die Wahrscheinlichkeit, dass kostengünstigere Web Services gefunden werden können. Die steigende Anzahl an möglichen Prozessschrittkandidaten bedingt aber auch gleichzeitig einen höheren Rechenaufwand für das Replanning, da eine größere Anzahl von Web Services berücksichtigt werden muss (siehe Abbildung 42c). Da aber die Kosteneinsparungen mit zunehmender Workflowlänge ebenfalls steigen, sinkt das Verhältnis Kostenaufwand zu eingesetzter Rechenzeit nur marginal (siehe Abbildung 42d).



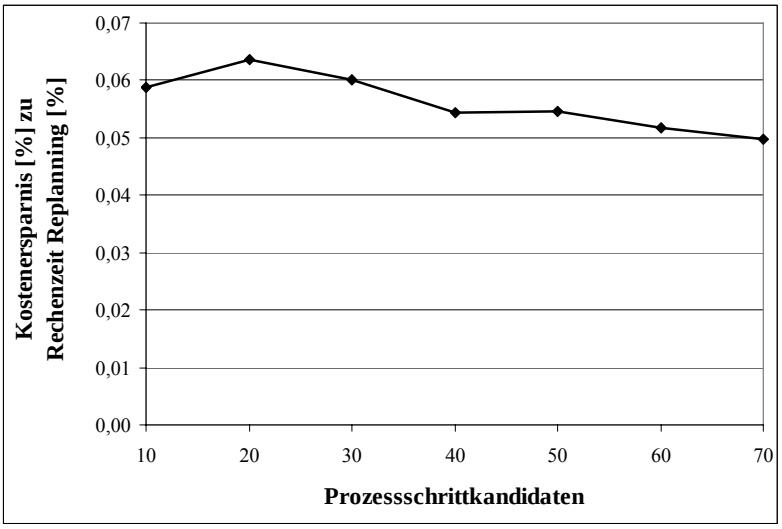
(42a) Antwortzeit des gesamten Workflow



(42b) Kosten



(42c) Rechendauer



(42d) Verhältnis Kostenersparnis zu zusätzlicher Rechendauer

Abbildung 42: Evaluierung der Anzahl der Prozessschrittkandidaten bei Replanning

5.4.4 Einfluss der Restriktionsstärke

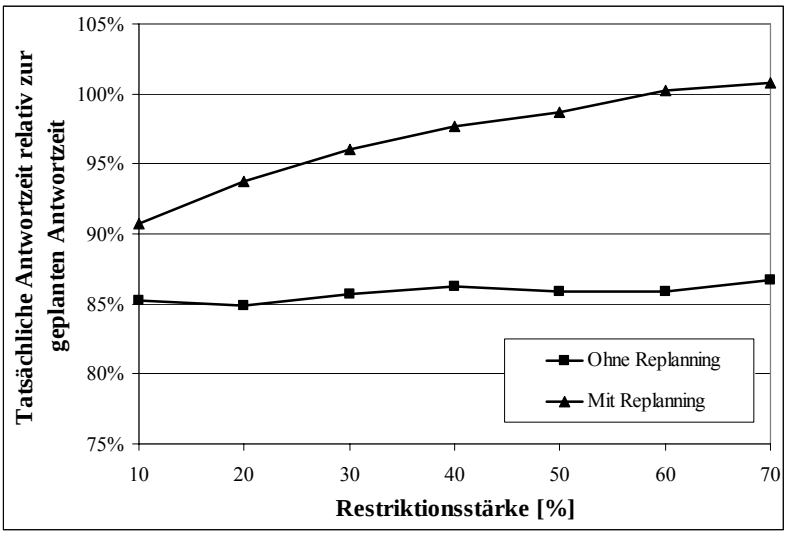
Um den Einfluss der Restriktionsstärke auf die Leistungsfähigkeit des Replanning-Verfahrens zu untersuchen, wird die Restriktionsstärke der Nebenbedingung, welche die Gesamtantwortzeit beschränkt, von 10% auf 70% erhöht. Die Workflowlänge wird auf 20 Prozessschritte und die Anzahl der Prozessschrittkandidaten auf 40 Web Services festgelegt. Nach jedem simulierten Prozessschritt wird ein Replanning durchgeführt. Die Ergebnisse können Tabelle 14 entnommen werden. Diese ist wie Tabelle 12 aufgebaut.

Restriktions-stärke [%]	Kosten-ersparnis [%]	Rechendauer ursprünglicher Aus-führungsplan [ms]	Zusätzliche Rechendauer durch Replan-ning [%]	Kosten-ersparnis [%] / Zusätzliche Rechendauer [%]
10	38,36	20,87	532,04	0,0721
20	40,60	21,44	518,37	0,0783
30	29,71	22,75	488,50	0,0608
40	24,91	23,12	478,80	0,0520
50	19,38	24,46	453,11	0,0428
60	16,50	25,25	437,94	0,0377
70	11,58	25,36	435,16	0,0266

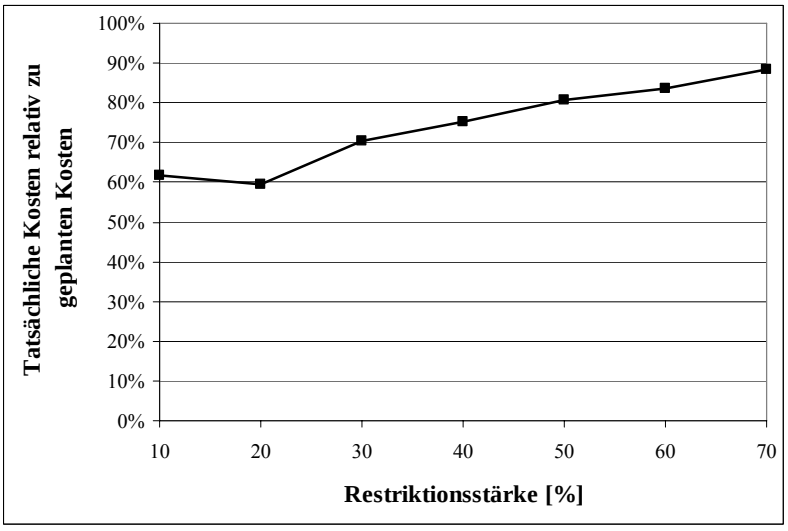
Tabelle 14: Evaluierung der Restriktionsstärke bei Replanning

Unabhängig von der Restriktionsstärke liegt die Gesamtantwortzeit ohne Replanning bei ca. 86% der ursprünglich geplanten Werte (siehe Abbildung 43a). Durch Replanning gelingt es, kostengünstige Web Services mit höheren Antwortzeiten zu verwenden, ohne dass die Nebenbedingung bezüglich der Gesamtantwortzeit verletzt wird. Mit zunehmender Restriktionsstärke werden die durch Replanning erreichten Kosteneinsparungen jedoch geringer (siehe

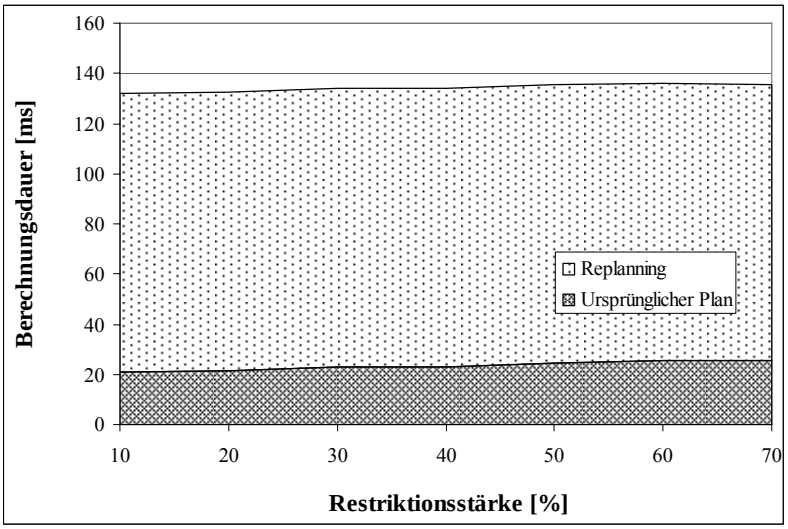
Abbildung 43b). Dies ist darauf zurückzuführen, dass es aufgrund der immer stärker werdenden Beschränkung des Lösungsraums nur noch wenige Web Services gibt, die überhaupt die Konstruktion einer gültigen Lösung erlauben, ohne dass die Nebenbedingung verletzt wird. Eine steigende Restriktionsstärke führt jedoch nicht zu einer höheren Rechenzeit (siehe Abbildung 43c), da die Anzahl der Prozessschritte und der Prozessschrittkandidaten gleich bleibt. Als eine Folge hiervon sinkt das Verhältnis Kosteneinsparung zu Rechenaufwand mit steigender Restriktionsstärke (siehe Abbildung 43d).



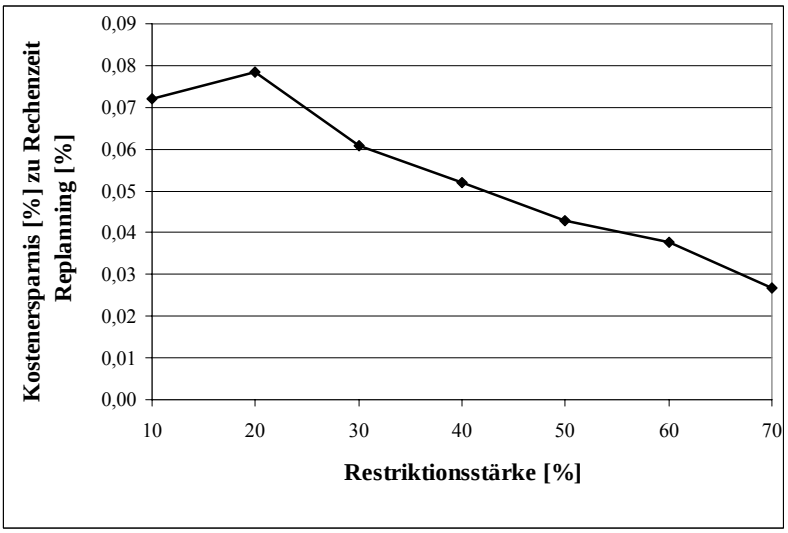
(43a) Antwortzeit des gesamten Workflow



(43b) Kosten



(43c) Rechendauer



(43d) Verhältnis Kostenersparnis zu zusätzlicher Rechendauer

Abbildung 43: Evaluierung der Restriktionsstärke bei Replanning

5.5 Zusammenfassung

In diesem Kapitel wurde ein Replanning-Verfahren für Service-orientierte Workflows vorgestellt, das bei der Auswahl und Zuordnung von Web Services zu abstrakten Prozessschritten das tatsächliche Laufzeitverhalten bereits ausgeführter Web Services berücksichtigt. Die Auswahl der Web Services wird hierbei als ein Optimierungsproblem modelliert, dass die Präferenzen und Anforderungen seitens des Nutzers bezüglich der Dienstgüte explizit berücksichtigt. Es wird gezeigt, wie dieses Replanning-Verfahren in BPEL4WS integriert werden kann. Eine prototypische Implementierung, welche die ActiveBPEL-Engine um einen Replanning-Mechanismus erweitert, wird ebenfalls vorgestellt. Da das Replanning zur Laufzeit durchgeführt wird, muss das zugrunde liegende Lösungsverfahren hochperformant sein. Aus diesem Grunde wurde das im Rahmen dieser Arbeit entwickelte heuristische Lösungsverfahren H1_RELAX_IP (vgl. Abschnitt 4.4.2) ausgewählt.

Die Evaluation des Replanning-Verfahrens basierend auf dieser Heuristik hat gezeigt, dass signifikante Kosteneinsparungen bei einem akzeptablen Rechenaufwand erreicht werden. Bezüglich der Kosteneinsparung ist die Restriktionsstärke von Bedeutung. Je geringer die Restriktionsstärke, d. h. je weniger der mögliche Lösungsraum durch die Nebenbedingungen beschränkt wird, desto höher sind die durch Replanning zu erzielenden Kosteneinsparungen. Die Anzahl der Prozessschritte und die Anzahl der Prozessschrittkandidaten haben ebenfalls eine, wenn auch nur geringe Auswirkung auf Kosteneinsparungen. So steigen die Kosteneinsparungen mit steigender Anzahl an Prozessschritten bzw. Prozessschrittkandidaten. Allerdings führen eine hohe Anzahl an Prozessschritten bzw. eine hohe Anzahl an Prozessschrittkandidaten zu einem deutlichen Anstieg der durch Replanning benötigten Berechnungszeit. Bezüglich des Verhältnisses aus erzielten Kosteneinsparungen zu dem hierfür benötigten Rechenaufwand liefert der Replanning-Mechanismus bei einer niedrigen Restriktionsstärke und einer geringen Anzahl an Prozessschritten bzw. Prozessschrittkandidaten die besten Ergebnisse.

6 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten und Themenbereiche erläutert. So werden in Abschnitt 6.1 Arbeiten diskutiert, die sich mit dem Dienstgütemanagement von Web Services und Service-orientierten Workflows auseinandersetzen. In Abschnitt 6.2 werden Arbeiten vorgestellt, die den Zusammenhang des SOA-Paradigmas bzw. der Web Service Technologie mit den Themenbereichen Grid Computing und Peer-to-Peer Computing untersuchen.

6.1 Dienstgüteunterstützung für Web Services

Zeng et al. stellen in [171] und [172] die Middleware-Plattform *AgFlow* vor. Diese Plattform ermöglicht es, abstrakt beschriebene zusammengesetzte Web Services auszuführen, wobei zum Aufrufzeitpunkt ein optimaler Ausführungsplan mit konkret zu verwendenden Web Services erstellt wird. Dieser berücksichtigt die Dienstgütekriterien der verfügbaren Web Services sowie Optimierungskriterien und Restriktionen. Über einen erweiterten UDDI-Verzeichnisdienst können anhand einer frei definierbaren Ontologie Web Services zur Ausführung einer Aktivität gefunden werden. Im Zustandsdiagramm des zusammengesetzten Web Service wird ein Weg vom Startzustand zum Endzustand als Ausführungspfad bezeichnet. Zu jedem Ausführungspfad wird ein optimaler Ausführungsplan bestimmt, der zu jeder Aktivität einen aufzurufenden Web Service definiert. Die Ausführungspläne werden anschließend zu einem Gesamtausführungsplan aggregiert. Werden einer Aktivität in verschiedenen Ausführungsplänen unterschiedliche Web Services zugeordnet, so wird derjenige Web Service in den Gesamtausführungsplan übernommen, der in dem Ausführungspfad festgelegt ist, der bisher am häufigsten ausgeführt wurde. Wird jedoch zur Laufzeit ein anderer Ausführungspfad ausgewählt, so ist es möglich, dass nicht alle Restriktionen eingehalten werden. Web Services werden durch die Dienstgüteparameter Ausführungszeit, Verfügbarkeit, Preis und Reputation beschrieben. Zur Erstellung des optimierten Ausführungsplans wird ein gemischt ganzzahliges Optimierungsproblem durch einen Solver exakt gelöst. Während der Ausführung des Gesamtausführungsplans wird ein Monitoring der Dienstgüteparameter durchgeführt. Wird hierbei festgestellt, dass die erbrachte Dienstgüte ausgeführter Web Services signifikant von der geplanten Dienstgüte abweicht, so wird ein Replanning ausgeführt. Die von Zeng et al. vorgelegte Evaluation zeigt jedoch, dass dieser auf einem exakten Lösungsverfahren basierende Ansatz für zeitkritische Anwendungen ungeeignet ist.

Yu und Lin befinden den Lösungsansatz von Zeng et al. [171, 172] als nicht geeignet, um Entscheidungen bzgl. der Zusammensetzung von Web Service Workflows zur Laufzeit treffen zu können, da die exakte Lösung des komplexen ganzzahligen linearen Optimierungsmodells zu viel Rechenzeit beansprucht [166]. Der von ihnen im Rahmen der *QoS-capable Web Service Architecture (QCWS)* [34, 169] entwickelte Lösungsansatz vereinfacht das Problem der Serviceauswahl auf zusammengesetzte Web Services mit sequenzieller Struktur. Die Erstellung eines Ausführungsplans für den zusammengesetzten Web Service Workflow lässt sich nur einer einzigen Restriktion, der Beschränkung der Gesamtantwortzeit, unterwerfen [166, 167]. Bei der Auswahl der einzelnen Web Services wird eine Nutzenmaximierung verfolgt. Den Nutzen, den ein Web Service bei seiner Verwendung stiftet, ist abhängig von den entste-

henden Kosten und der Auslastung des Servers, der ihn bereitstellt. Eine niedrige Serverlast stellt für Yu und Lin einen wichtigen Aspekt dar, um Clients mit einer hohen Dienstgüte versorgen zu können. In [165] stellen sie daher die Algorithmen HQ und RQ zur Ressourcenallokation vor, die es Servern in der QCWS-Architektur ermöglicht, Ressourcen so auf die anfragenden Clients zu verteilen, dass sowohl die Serverlast als auch die Anzahl der nötigen Rekonfigurationen minimiert wird. Da Yu und Lin im Rahmen der Serviceauswahl in [166, 167] nur Optimierungsmodelle mit einer Restriktion betrachten, ist es möglich, dieses Optimierungsproblem als *Multiple-Choice Knapsack Problem (MCKP)* zu beschreiben. Einem Web Service werden sein Nutzen sowie seine Antwortzeit zugeordnet. Der Rucksack (engl. Knapsack) ist nutzenmaximal zu füllen, wobei die Kapazität des Rucksackes durch eine maximale Antwortzeit beschränkt ist. Zur Lösung des NP-schweren MCKP wird der Algorithmus von Pisinger [110] verwendet. Dieser vereinfacht das MCKP zunächst durch Anwenden einer LP-Relaxation zum *Linearen Multiple-Choice Knapsack Problem (LMCKP)* und löst dieses exakt in linearer Laufzeit O(n). Anschließend werden Lösungsverfahren der dynamischen Programmierung verwendet, um aus der optimalen Lösung des LMCKP die optimale Lösung des MCKP zu konstruieren. Der Algorithmus von Pisinger löst das MCKP sehr effizient und die von Yu und Lin in Experimenten gemessenen sehr geringen Laufzeiten zeigen dessen Eignung für die Lösung von Optimierungsmodellen in zeitkritischen Anwendungen. Hierbei ist jedoch zu beachten, dass das Modell sehr einfach gehalten ist und nur eine einzige Restriktion sowie wenige Dienstgüteeigenschaften berücksichtigt. Für den Fall von zusammengesetzten Web Services mit nicht-sequenzieller Struktur schlagen Yu und Lin vor, für jeden möglichen Ausführungspfad den Pisinger-Algorithmus zu verwenden und anschließend den Ausführungsplan des Ausführungspfades zu wählen, der den höchsten Nutzenwert erzielt. Wie jedoch verfahren werden soll, wenn dieser Ausführungspfad nicht auch tatsächlich ausgeführt wird, bleibt unbeantwortet. In [168] erweitern Yu und Lin ihre Arbeiten und betrachten mehrere Restriktionen. So modellieren sie die Serviceauswahl als *Multi-Dimension Multi-Choice 0-1 Knapsack Problem (MMKP)* und als *Multi-Constraint Optimal Path (MCOP)*. Zur Lösung beider Ansätze wird jeweils ein heuristisches Lösungsverfahren vorgestellt. Die Evaluation zeigt, dass diese Verfahren ein sehr gutes Laufzeitverhalten aufweisen, das vergleichbar ist mit dem Laufzeitverhalten der Heuristik H1_RELAX_IP. Allerdings wird nicht evaluiert, wie sich das Laufzeitverhalten und die Lösungsgüte dieser Heuristiken bei einer hohen Restriktionsstärke verhalten. Zudem geht nicht klar hervor, wie Dienstgüteeigenschaften, die anhand des Minimum-Operators aggregiert werden, z. B. der Durchsatz, im Modell berücksichtigt werden. In [170] stellen Yu und Lin einen Replacing-Mechanismus vor, der nach einer gescheiterten Web Service Ausführung einen alternativen Ausführungsplan errechnet. Dieser Ansatz ist grundsätzlich vergleichbar mit dem in dieser Arbeit vorgestellten Replanning-Mechanismus. Allerdings ist der Replacing-Mechanismus vorwiegend auf den Eintritt eines Web Service Ausfalls fixiert und berücksichtigt nicht, dass durch die Abweichung des tatsächlichen vom geplanten Laufzeitverhalten eventuell Nutzerpräferenzen und Restriktionen nicht mehr gewahrt sind. Des Weiteren wird bei der Evaluation des Replacing-Mechanismus nur das Laufzeitverhalten untersucht.

Canfora et al. [23] beschäftigen sich ebenfalls mit der dienstgüteunterstützten Komposition zusammengesetzter Web Services, jedoch verwenden sie zur Lösung des zugrunde liegenden Optimierungsproblems eine Heuristik auf Basis Genetischer Algorithmen (GA). Berücksichtigte Dienstgütekriterien sind Ausführungszeit, Kosten, Verfügbarkeit und Zuverlässigkeit. Da das Optimierungsproblem nicht als (G)LOP formuliert wird, müssen weder die Aggregationsfunktionen noch die Zielfunktion bzw. die Restriktionen linear sein, was eine flexible Modellierung der Dienstgütekriterien zulässt. Die zu optimierende Zielfunktion kombiniert die normierten und gewichteten Gesamtparameter eines Ausführungsplans zu einem Nutzenwert. Da genetische Algorithmen bei der Optimierung nicht direkt Restriktionen berücksichtigen, geht die Summe der Abweichung von der Erfüllung definierter Restriktionen als hoch gewichtetes Strafmaß in die Zielfunktion ein. Die Optimierung des Zielfunktionswertes führt daher implizit auch zu einer bestmöglichen Erfüllung der Restriktionen. Als Gen bzw. Individuum wurde im GA eine Array gewählt, welches zu jedem abstrakt definierten Web Service in der Struktur des zusammengesetzten Web Service Workflows den Index des konkret verwendeten Web Service enthält. Zur Optimierung wird eine gewisse Population an Individuen erzeugt und durch Kreuzungs- und Mutationsoperationen verändert bzw. vermehrt. Die Individuen mit den besten Zielfunktionswerten überleben und bilden die nächste Generation. Dieses Vorgehen wird solange wiederholt, bis ein Abbruchkriterium erfüllt wird. Canfora et al. schlagen mehrere alternative Abbruchkriterien vor, wie z. B. das Erreichen einer maximalen Anzahl von Iterationen, das Erfüllen aller Restriktionen oder die Unveränderlichkeit des Zielfunktionswertes des besten Individuums über eine gewisse Anzahl von Iterationen hinweg. In der Analyse des GA haben Canfora et al. die Laufzeit und Lösungsgüte mit der Lösung eines entsprechenden GLOPs verglichen. Um die gleiche Lösungsgüte wie das GLOP zu erreichen, benötigt der GA bei wenigen Kandidaten pro Web Service eine höhere Laufzeit. Mit zunehmender Anzahl an Kandidaten pro Web Service steigt die Laufzeit zur Lösung des GLOPs exponentiell, wohingegen sich die Laufzeit des GA als nahezu konstant darstellt. Die Rechenzeit des GA wird hauptsächlich durch die Anzahl der abstrakten Web Services in der Struktur des zusammengesetzten Web Services determiniert, nicht jedoch durch die Anzahl der Kandidaten für diese abstrakten Web Service. Die Problematik der Überwachung der Dienstgüte zur Ausführungszeit und des Einleitens und Durchführens eines ggf. nötigen Replannings wird von Canfora et al. berücksichtigt und in weiteren Arbeiten [22, 24] ausführlich diskutiert.

Aggarwal et al. stellen in [2] das Framework *METEOR-S* (METEOR for Semantic Web Services) vor. Dieses Framework ermöglicht die semantische Beschreibung von Web Services und daraus komponierter Workflows durch die Erweiterung mit semantischen Informationen. Zudem wird eine entsprechende Laufzeitumgebung zur Ausführung solcher Web Service Workflows bereitgestellt. Mittels eines *Abstract Process Designer* ist es möglich, BPEL4WS-Prozesse zu definieren, die keine konkreten Web Services, sondern semantische Beschreibungen benötigter Web Services in Form von *Service Templates* enthalten. Ein Service Template beschreibt den benötigten Web Service mittels Ontologien in Bezug auf die zu verarbeitenden Daten, die durchzuführende Funktionalität und gewünschte Dienstgüteeigenschaften. Informationen über verfügbare Web Services werden mit entsprechenden semantischen Zusatzinformationen in einem erweiterten UDDI-Verzeichnisdienst abgelegt. Die *Discovery Engine* ist

in der Lage, zu den Service Templates der abstrakten Prozessbeschreibung die Menge der geeigneten Web Services aus dem UDDI-Verzeichnisdienst zu ermitteln. Der *Constraint Analyser* kann aus der Menge der geeigneten Web Services einen konkreten Web Service derart auszuwählen, dass zuvor definierte Nebenbedingungen berücksichtigt werden. In diesem Zusammenhang können Anforderungen an die nicht-funktionalen Eigenschaften des Prozesses (wie z. B. Ausführungszeit, Kosten, Verlässlichkeit und Verfügbarkeit) definiert werden. Zur Berechnung der nicht-funktionalen Eigenschaften stehen Standard-Aggregationsfunktionen (z. B. Addition und Multiplikation) zur Verfügung. Die Anforderungen an die nicht-funktionalen Eigenschaften des Prozesses werden als GLOP formuliert, dessen Zielfunktion aus einer Linearkombination der numerisch ausgedrückten, nicht-funktionalen Eigenschaften des Prozesses besteht. Das GLOP wird mittels eines Solvers gelöst. Wie es hierbei jedoch gelingt, individuell definierbare Aggregationsfunktionen für alle in einem BPEL4WS-Geschäftsprozess möglichen Kontrollkonstrukte anzugeben und automatisiert in ein lineares Optimierungsproblem zu übertragen, wird nicht näher erläutert. Angaben zur Laufzeit, die für die Lösung eines derartigen Optimierungsproblems nötig ist, können der Arbeit ebenfalls nicht entnommen werden. Durch den *Binder* werden die in Form von Service Templates abstrakt definierten Web Services im BPEL4WS-Prozess automatisch durch die konkret gewählten Web Services ersetzt und an die BPEL4WS-Workflow-Engine übergeben werden. Aussagen darüber, ob die nicht-funktionalen Eigenschaften der verwendeten Web Services während der Prozessausführung durch eine Monitoring-Komponente überwacht und protokolliert werden und ob ein Replanning durchgeführt werden kann, werden nicht getroffen.

Tian stellt in [139] eine erweiterte Web Service Architektur und deren prototypische Implementierung vor, die sich dadurch auszeichnet, dass sie ein Schichten übergreifendes Dienstgütemanagement ermöglicht. Die Beschreibung von Dienstgüteanforderungen seitens des Nutzers bzw. der zugesicherten Dienstgüteeigenschaften des Serviceanbieters wird durch ein hierfür entwickeltes XML-Schema ermöglicht. Die Auswahl konkreter Services und deren Aufruf werden, ähnlich wie in der vorliegenden Arbeit, über eine Proxy-Architektur realisiert. Ein Schwerpunkt der Arbeit von Tian liegt auf der Unterstützung von Web Services, die auf mobilen Geräten ausgeführt werden. Es werden jedoch keine Algorithmen und Replanning-Mechanismen im Rahmen der Serviceauswahl vorgestellt, die sicherstellen, dass Präferenzen und Restriktionen, die sich auf einen Web Service Workflow beziehen, berücksichtigt werden.

Cardoso et al. beschreiben in [26] einen Ansatz, die nicht-funktionalen Eigenschaften eines Web Service Workflows zu bestimmen. Hierzu definieren sie ein Dienstgütemodell aus den drei nicht-funktionalen Eigenschaften Zeit, Kosten und Zuverlässigkeit. Zur Definition eines Web Service Workflows können verschiedene Kontrollkonstrukte verwendet werden. Die nicht-funktionalen Eigenschaften der Aktivitäten sowie die Wahrscheinlichkeiten, mit denen im Workflow verzweigt wird, sind mit Annahmen über das zu erwartende Minimum, Maximum, den Durchschnitt und einer Verteilungsannahme zu initialisieren. Anschließend können die nicht-funktionalen Eigenschaften durch einen als *Stochastic Workflow Reduction (SWR)* bezeichneten Algorithmus berechnet werden [25]. Der Algorithmus reduziert den Workflow

schrittweise durch Anwendung von sechs Reduktionsoperationen, die das Inverse der zur Konstruktion erlaubten Kontrollkonstrukte darstellen, auf einen einzigen atomaren Prozessschritt. Bei jeder Reduktion erfolgt die Aggregation der nicht-funktionalen Eigenschaften, bis schließlich dem verbleibenden atomaren Prozessschritt die aggregierten, nicht-funktionalen Eigenschaften des gesamten Workflows zugewiesen werden. Während der Ausführung eines Workflows werden die tatsächlichen nicht-funktionalen Eigenschaften der Aktivitäten sowie die tatsächlichen Durchlaufpfade durch den Workflow von einer Monitoring-Komponente protokolliert. Mittels dieser Informationen werden die Annahmen über die nicht-funktionalen Eigenschaften der Aktivitäten sowie die Wahrscheinlichkeiten, mit denen im Workflow verzweigt wird, aktualisiert. Die Thematik der Auswahl von Services entsprechend den Anforderungen an die nicht-funktionalen Eigenschaften eines Workflows wird nicht angesprochen. Cardoso et al. beschränken sich auf einen Ansatz, der es ermöglicht, die nicht-funktionalen Eigenschaften eines Workflows bestehend aus fest definierten Services möglichst präzise vorherzusagen.

Tosic et al. stellen in [142, 145] die Web Service Architektur *Web Service Offerings Infrastructure (WSOI)* vor. Diese ermöglicht das Management von Web Services basierend auf Serviceklassen (sog. *Service Offerings*). Die Serviceklassen eines Web Service unterscheiden sich bezüglich der Dienstgüteeigenschaften, Zugriffsrechte und Kosten, nicht aber in der erbrachten Funktionalität. Serviceklassen können mit der *Web Service Offerings Language (WSOL)* [146] modelliert werden. WSOI bietet Monitoring- und Accounting-Mechanismen für die mit WSOL modellierten Serviceklassen. In [89] wird gezeigt, wie Appache Axis erweitert wird, um die entsprechenden Monitoring-Mechanismen im Rahmen von WSOI umzusetzen. WSOI ist in der Lage, dynamisch zur Laufzeit Serviceklassen zu ändern [143]. So kann ein Nutzer eine andere Serviceklasse wählen, die eine schnellere Ausführungszeit verspricht, sollte er mit dem Laufzeitverhalten der aktuellen Serviceklasse unzufrieden sein. Eine Erweiterung von WSOI, welche die Anforderungen von mobilen Web Services berücksichtigt, stellen Tosic et al. in [144] vor. In [43] wird vorgeschlagen, die Suche nach geeigneten Web Service dezentral umzusetzen, um Engpässe bei einem zentralen Verzeichnisdienst zu verhindern. Auf eine prototypische Umsetzung dieses Ansatzes wird jedoch nicht näher eingegangen. Auch werden in den Arbeiten von Tosic et al. keine Algorithmen zur Serviceauswahl beschrieben.

Maximilien und Singh stellen in [97] das *Web Service Agent Framework (WSAF)* vor, welches die Vertrauenswürdigkeit der Aussagen von Anbietern über die Dienstgüte ihrer angebotenen Web Services durch einen Austausch von Informationen über die tatsächlich von den Web Services erbrachte Dienstgüte durch Agenten berücksichtigt. Web Services werden im WSAF von den Nutzern nicht direkt aufgerufen. Dies übernehmen Agenten, die als Proxy dem Nutzer dieselbe Schnittstelle zur Verfügung stellen, jedoch eine dynamische Auswahl des auszuführenden Web Service vornehmen. Die Auswahl eines konkreten Web Service erfolgt anhand von Dienstgüteeigenschaften, die mittels einer Ontologie festgelegt werden können. Ein Anbieter definiert sein Dienstgüteangebot und ein Nutzer seine Dienstgüteanforderungen in Form einer *Policy* auf Basis einer definierten Dienstgüteontologie. Damit der Agent

einen konkreten Web Service aufrufen kann, wird ein Matching der Policies von Anbieter und Nutzer durchgeführt, um für den Nutzer einen möglichst geeigneten Web Service ermitteln zu können. Bei dieser Auswahl wird neben den Informationen der Policy des Anbieters zusätzlich berücksichtigt, welche Dienstgüte der Web Service in der Vergangenheit tatsächlich erbracht hat. Je stärker die Abweichung von der durch eine Policy versprochenen Dienstgüte in der Vergangenheit war, desto weniger Vertrauen wird dieser Policy geschenkt. Bei einem Aufruf des Web Service protokolliert der Agent die tatsächlich erbrachte Dienstgüte des Web Service und berücksichtigt sie bei zukünftigen Entscheidungen. Um die Verbreitungsgeschwindigkeit der Vertrauensinformationen zu steigern, können Agenten über sog. *Agencies* Informationen über die erbrachte Dienstgüte von Web Services mit anderen Agenten austauschen. Eine Agency verwaltet diese Informationen und ermöglicht es so, Agenten bei der Auswahl von Web Services an den Erfahrungen anderer Agenten partizipieren zu lassen. In einer Simulation verfälschen Maximilien und Singh die von Anbietern angebotene Dienstgüte und lassen lediglich einen Anbieter die versprochene Dienstgüte erbringen. Die Auswertungen zeigen, dass der Aufruf eines Web Service über einen Agenten mit steigender Anzahl von Aufrufen zunehmend präziser den Web Service wählt, dessen Angebot unverfälscht, d. h. vertrauenswürdig, ist. Bei einem Austausch von Informationen von mehreren Agenten über eine Agency konvergiert die Auswahl schneller zu dem vertrauenswürdigen Web Service. Durch den Einsatz des WSAF kann eine optimierte Auswahl von Web Services auf der Basis einzelner Prozessschritte in einem Geschäftsprozess erreicht werden. Es ist jedoch nicht möglich, einen Ausführungsplan für den gesamten Geschäftsprozess unter Einhaltung globaler Restriktionen und Optimierungskriterien zu erstellen. Der Ansatz liefert Informationen darüber, wie Informationen über die Vertrauenswürdigkeit eines Anbieters in ein Dienstgütemodell integriert und entsprechende Informationen unter den Nutzern propagiert werden können.

Liu et al. diskutieren in [87] Empfehlungen, aus welchen Quellen Dienstgüteinformationen bezogen werden können und wie die Berechnung eines Wertes erfolgen kann, der den Grad der Übereinstimmung der Dienstgüte eines Web Service mit den Dienstgüteanforderungen eines Nutzers ausdrückt. Auf Basis des Dienstgütemodells ist es für den Nutzer möglich, Anforderungen an die Dienstgüte eines Web Service präzise zu formulieren, damit der Grad der Eignung eines Web Service bestimmt werden kann. Als wichtigste Informationsquellen für Dienstgüteinformationen werden gesehen: Übermittlung von Dienstgüteeigenschaften durch den Web Service Anbieter, Ermittlung von Dienstgüteeigenschaften durch ein Monitoring während der Nutzung des Web Service sowie Feedback der Nutzer über die erbrachte Dienstgüte nach Ausführung des Web Service. Ein Verzeichnis von Web Services, welches Dienstgüteinformationen beinhaltet, sollte alle drei Informationsquellen unterstützen und sowohl Web Service Anbietern, Monitoring-Komponenten als auch Web Service Nutzern entsprechende Schnittstellen zur Aktualisierung der Dienstgüteinformationen bereitstellen. Unter den Dienstgütekriterien unterscheiden Liu et al. zwischen generischen Kriterien, die in Bezug auf alle Web Services verwendet werden können, und geschäfts- bzw. domänenspezifische Kriterien mit besonderer Relevanz im spezifischen Kontext der Nutzung. Als Beispiele für generische Kriterien werden die Kosten der Ausführung (sollten durch den Anbieter übermittelt werden), die benötigte Ausführungszeit (sollte durch Monitoring gemessen werden) und die

Reputation des Anbieters (sollte durch Feedback der Nutzer ermittelt werden) genannt. Zur Bestimmung des Grades der Übereinstimmung der angebotenen Dienstgüte mit den Dienstgüteanforderungen eines Nutzers wird die Verwendung einer Metrik vorgeschlagen. Zur Berechnung dieser Metrik werden zunächst eine Normalisierung der Dienstgütewerte eines Web Service anhand des Durchschnitts aller Dienstgütewerte und einem definiertem Maximum durchgeführt. Die normalisierten Dienstgütewerte werden gemäß den definierten Gruppen aggregiert und erneut anhand der Gruppendurchschnitte und eines weiteren definierbaren Maximums normalisiert. Die Metrik ergibt sich als gewichtete Summe der normalisierten Dienstgütegruppenwerte. Die Metrik ist je höher, desto besser die mittels der definierten Gewichte ausgedrückten Anforderungen des Nutzers erfüllt werden. Hierdurch lässt sich eine Bewertung der Web Services nach deren Eignung zur Erfüllung der Nutzeranforderungen durchführen. Durch das von Liu et al. vorgeschlagene Ranking von Web Services anhand einer Metrik ist es möglich, eine optimierte Auswahl von Web Services auf der Basis einzelner Prozessschritte in einem Geschäftsprozess zu erreichen. Es ist jedoch nicht möglich, einen Ausführungsplan für einen gesamten Geschäftsprozess unter Berücksichtigung globaler Restriktionen und Optimierungskriterien zu erstellen.

Jäger [63] bzw. Jäger et al. [64, 65, 67] untersuchen ebenfalls die Komposition von Web Services zu Workflows unter Berücksichtung der Web Service Dienstgüte. Die Autoren verfolgen, ähnlich wie in der vorliegenden Arbeit, den Ansatz, die Serviceauswahl als Optimierungsproblem zu modellieren. Hierbei vergleichen sie die dienstgütebasierte Serviceauswahl mit ganzzahligen bzw. kombinatorischen Optimierungsproblemen, wie z. B. dem MCKP. Jäger bzw. Jäger et al. kommen auch zu dem Ergebnis, dass aufgrund der Tatsache, dass es sich bei der dienstgütebasierten Serviceauswahl um ein NP-schweres Optimierungsproblem handelt, heuristische Lösungsverfahren einzusetzen sind. Diese werden implementiert und hinsichtlich verschiedener Einsatzszenarien evaluiert. Ein weiterer Schwerpunkt der Arbeiten liegt auf der Berechnung von aggregierten Dienstgüteeigenschaften bei komplexen Workflowstrukturen [66].

Chang et al. bemängeln in [29], dass bei der Vorhersage der Dienstgüte von Web Services bzw. Web Service Workflows oft von zu statischen Annahmen und Szenarien ausgegangen wird. Sie schlagen daher vor, durch Simulationen der Verhaltensweisen von Web Services und Prozessen realistischere Aussagen über die erreichbare Dienstgüte zu treffen. Anbieter von Web Services können hierdurch das Verhalten ihrer Web Services unter bestimmten Bedingungen (z. B. Entwicklung der Antwortzeit beim Ansteigen der Anzahl gleichzeitiger Nutzer) besser abschätzen und durch die Analyse relevanter Szenarien präzisere Aussagen bzgl. der Dienstgüte ihrer Web Services geben. Nutzer können mithilfe dieser Informationen einerseits das Verhalten von Web Service Workflows analysieren und andererseits durch Variation der verwendeten Web Services Entscheidungen über die Auswahl konkreter Web Services treffen. Zur Lösung komplexer Problemstellungen, wie z. B. der Auswahl konkreter Web Services zur Einhaltung gewisser Restriktionen bei gleichzeitiger Optimierung von Dienstgütekriterien, schlagen sie die Formulierung und Lösung ganzzahliger Optimierungsprobleme vor. Die Autoren stellen in diesem Zusammenhang fest, dass zum Lösen derartiger Optimie-

rungsprobleme durch exakte Verfahren ein hoher Rechenzeitbedarf entsteht. Sie schlagen daher vor, statt exakter Lösungsmethoden entsprechende Heuristiken zur näherungsweisen Lösung des Problems zu verwenden, ohne jedoch hierauf näher einzugehen. Zur Unterstützung der Simulation von Prozessen skizzieren sie ein Framework, welches es ermöglicht, einen Prozess grafisch zu entwerfen und hieraus zugleich eine Prozessdefinition in einer Beschreibungssprache (z. B. BPEL4WS) zu erstellen und ein Simulationsmodell zu generieren. Wie dies technisch umgesetzt werden soll, wie Szenarien zur Analyse definiert werden können, aus welchen Quellen Dienstgüteinformationen stammen, wie diese zur Dienstgüte von Prozessen aggregiert werden und wie Ergebnisse verarbeitet und genutzt werden können, wird jedoch nicht näher erläutert.

Die Arbeiten von Charfi und Mezini, z. B. [30, 33], zeichnen sich dadurch aus, dass das Paradigma der Aspektorientierten Anwendungsentwicklung u. a. auch auf mit BPEL4WS modellierte Workflows übertragen wird. Vorteile dieses Ansatzes bestehen in einer höheren Flexibilität und der Modularisierung sog. *Crosscutting Concerns*, wie z. B. Monitoring, Billing und Charging. Als Proof-of-Concept wird mit AO4BPEL ein Konzept und dessen prototypische Umsetzung einer aspektorientierten Sprache zur Komposition von Web Services vorgestellt. Es wird zudem ein *Process Container Framework* beschrieben, das eine Middleware-Unterstützung (z. B. Transaktionsmechanismen und Sicherheit) für mit AO4BPEL modellierte Web Service Workflows anbietet [32]. Des Weiteren stellen die Autoren im Zusammenhang mit AO4BPEL einen regelbasierten Ansatz zur Komposition von Web Services vor [31].

Karastoyanova et al. stellen in [71] eine Erweiterung von BPEL4WS vor, welche die Flexibilität der mit BPEL4WS modellierten Workflows erhöht. Der in diesem Zusammenhang von den Autoren eingeführte `find-and-bind`-Mechanismus erlaubt die Auswahl von Web Services zur Laufzeit des BPEL4WS-Workflows, z. B. auf Basis von Policies. Als Proof-of-Concept wird eine BPEL4WS-Engine vorgestellt, die diese Erweiterungen unterstützt. In [72, 73] zeigen Karastoyanova et al., wie `portTypes` und `operations` bei BPEL4WS-Workflows parametrisiert werden können, um deren Flexibilität und Wiederverwendung zu erhöhen.

6.2 SOA und Web Services in anderen Forschungsgebieten

An verteilte Systeme und Anwendungen, deren Funktionalität den Nutzern über das Internet zugänglich gemacht wird, werden zukünftig gestiegene Anforderungen vor allem bezüglich Skalierbarkeit und Verfügbarkeit gestellt. Es zeigt sich zunehmend, dass die vorwiegend eingesetzten Client-Server-basierten Anwendungen diesen neuen Anforderungen des Internets nicht mehr umfassend gerecht werden [135]. So sind insbesondere zentrale Server, deren Ressourcen zum Flaschenhals werden können, gezielte Angriffspunkte für Hackerattacken. In diesem Zusammenhang stellt das Peer-to-Peer-Paradigma [94, 136] eine Alternative dar. Unter einem Peer-to-Peer-System wird ein sich selbst organisierendes, dezentrales System gleichberechtigter, autonomer Entitäten (sog. Peers) verstanden, das vorzugsweise ohne Nutzung zentraler Dienste auf der Basis eines Rechnernetzes mit dem Ziel der gegenseitigen Nutzung von Ressourcen operiert [103, 135]. Das Peer-to-Peer-Paradigma wird auch im Zusam-

menhang von Service-orientierten Architekturen und Web Services untersucht, z. B. [47, 58, 147, 159]. Götze et al. [56] stellen einen Ansatz vor, der den Verzeichnisdienst für Services dezentral basierend auf JXTA realisiert. Als Proof-of-Concept wird das Konzept im Rahmen der Venice-Service-Grid-Architektur [55, 57] umgesetzt. Diese Architektur bietet verschiedene Voice-Dienste basierend u. a. auf SIP sowie Mehrwertdienste als Web Services im Rahmen eines Service Grids an. Weitere Arbeiten thematisieren den Zusammenhang zwischen Grid Computing, dem SOA-Paradigma und Web Services. Unter Grid Computing wird eine Infrastruktur zur gemeinschaftlichen Nutzung von Ressourcen (wie z. B. Speicherplatz, Rechenleistung und Bandbreite) verstanden [45]. Die Idee hierbei ist, dass diese Ressourcen bedarfsgerecht wie Strom aus der Steckdose verwendet und abgerechnet werden. Nutzer können über eine definierte Schnittstelle Ressourcen anfordern, die ihnen dann von einer koordinierenden Instanz automatisch zugewiesen wird. Hierbei beschreibt der *OGSA (Open Grid Service Architecture)*-Ansatz [102] eine Möglichkeit, wie Grid Services in Form von Web Services angeboten werden können. Forschungsarbeiten, welche die Konvergenz von SOA, Web Services und Grid Computing untersuchen, beschäftigen sich hierbei schwerpunktmäßig mit Fragestellungen, wie Dienstgüte [162] und Sicherheit [127, 128] im Rahmen von Grid Computing sichergestellt werden können. Des Weiteren werden Themenstellungen über betriebswirtschaftliche Gesichtspunkte des Grid Computing untersucht, z. B. [124].

7 Zusammenfassung und Ausblick

Dieses Kapitel fasst die wesentlichen Ergebnisse der Arbeit zusammen und diskutiert im Ausblick Anknüpfungspunkte für zukünftige forschungsrelevante Fragestellungen im Umfeld von Service-orientierten Workflows.

7.1 Zusammenfassung

Das Paradigma der Service-orientierten Architektur gewinnt im Zusammenhang mit der Überwindung von Komplexität in heterogenen Anwendungslandschaften und der architekturellen Unterstützung unternehmensübergreifender Workflows zunehmend an Bedeutung. Dies gilt im besonderen Maße für Web Services als Technologie basierend auf offenen XML-Standards zur Umsetzung des SOA-Paradigmas.

Entscheidend für den weiteren Erfolg des SOA-Paradigmas und der Web Service Technologie ist eine umfassende Unterstützung des Dienstgütemanagements der entsprechenden Services und der aus Services zusammengesetzten Workflows. Während Dienstgüte im Zusammenhang von Kommunikationsnetzwerken und Multimedia-Anwendungen seit vielen Jahren Gegenstand der Forschung ist, ist Dienstgüte im Umfeld von SOA und Service-orientierten Workflows noch ein wenig erforschtes Themengebiet. Die vorliegende Arbeit liefert einen umfassenden Beitrag bei der Erforschung von Dienstgüte von Service-orientierten Workflows.

Im Rahmen dieser Arbeit wird die Dienstgütearchitektur WSQoSX vorgestellt, die das Dienstgütemanagement für Service-orientierte Workflows unterstützt. WSQoSX übernimmt das Dienstgütemanagement basierend auf einem neun Phasen umfassenden Vorgehensmodell. So wird durch WSQoSX sichergestellt, dass nur Web Services ausgewählt werden, deren Dienstgüteeigenschaften den Anforderungen des Nutzers entsprechen. Die Dienstgüteeigenschaften der entsprechenden Web Services müssen durch Service Level Agreements zugesichert werden. Die Einhaltung der SLAs wird ebenfalls durch WSQoSX überwacht.

Zudem wird als Weiterentwicklung von WSQoSX in dieser Arbeit gezeigt, wie die dienstgütebasierte Auswahl von Web Services basierend auf den Nutzerpräferenzen und -anforderungen an den gesamten Workflow in Form eines Optimierungsproblems modelliert werden können. Zur Lösung dieses NP-schweren Optimierungsproblems werden heuristische Lösungsverfahren entwickelt und implementiert. Die Evaluation dieser Heuristiken wird mittels einer Simulationsumgebung durchgeführt. Die Ergebnisse zeigen u. a., dass ein heuristisches Verfahren eine Lösungsgüte von durchschnittlich ca. 99% erreicht und hierbei ein wesentlich besseres Laufzeitverhalten hat als das exakte Lösungsverfahren. Somit ist das in der vorliegenden Arbeit entwickelte heuristische Lösungsverfahren sehr gut auch für den Einsatz in zeitkritischen Anwendungen mit Nutzerinteraktion geeignet.

Das tatsächliche Laufzeitverhalten der Web Services weicht u. a. aufgrund der Unzuverlässigkeit externer Serviceanbieter und der Unzuverlässigkeit des Internets als Kommunikationskanal zumeist von dem ursprünglich in SLAs geplanten ab. Daher wird in dieser Arbeit ein

Replanning-Mechanismus vorgestellt, der zur Ausführungszeit das tatsächliche Laufzeitverhalten bereits ausgeführter Web Services bei der Auswahl noch nicht aufgerufener Web Services berücksichtigt. Auf diese Weise wird sichergestellt, dass sowohl die Präferenzen des Nutzers als auch die von ihm definierten Restriktionen berücksichtigt werden. Eine prototypische Implementierung, welche die ActiveBPEL-Engine um einen Replanning-Mechanismus erweitert, wird ebenfalls vorgestellt. Als Proof-of-Concept wird das entwickelte Replanning-Verfahren auf Basis der in dieser Arbeit entwickelten Heuristiken implementiert sowie dessen Laufzeitverhalten und die durch Replanning realisierten Kosteneinsparungen simuliert. Die Evaluation des auf einem heuristischen Lösungsverfahren basierenden Replanning-Ansatzes zeigt, dass die Anwendung von Replanning zu signifikanten Kosteneinsparungen bei einem vertretbaren Rechenaufwand führt.

7.2 Ausblick

Die SOA-Forschungsagenda (siehe Abbildung 44) zeigt weitere, im SOA-Umfeld relevante Themengebiete, die Gegenstand zukünftiger Forschungsvorhaben sein können. Einige dieser Themengebiete werden im Folgenden kurz skizziert.

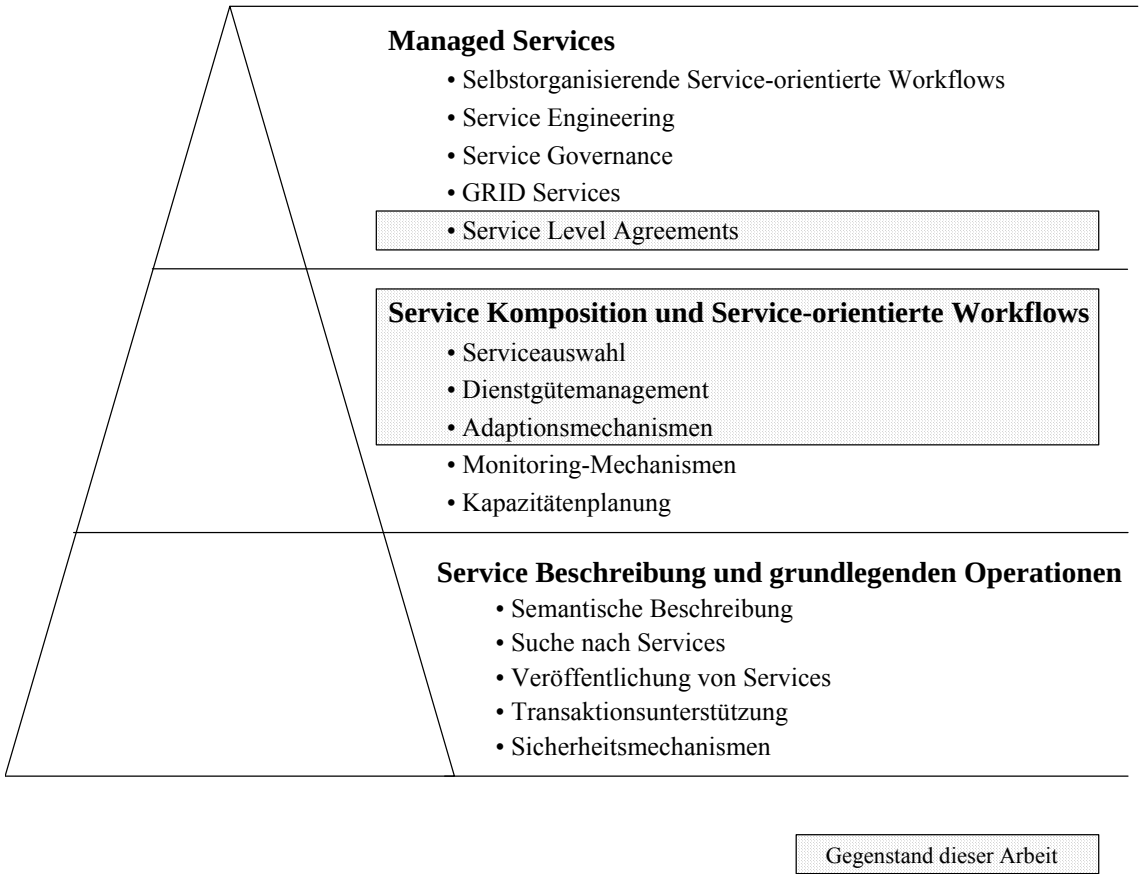


Abbildung 44: SOA-Forschungsagenda (in Anlehnung an [106])

Im Umfeld des *Autonomic Computing (bzw. Organic Computing)* [21] wird untersucht, wie sich Grundprinzipien, Konzepte und Mechanismen aus der (Molekular-)Biologie bzw. verwandten Bereichen auf IT-Systeme übertragen lassen. In diesem Zusammenhang steht die Erforschung *selbstorganisierender Systeme* im Mittelpunkt. Unter selbstorganisierenden Sys-

temen versteht man Systeme, die sich eigenständig an die Bedürfnisse ihrer Umgebung anpassen können, z. B. [126]. Charakteristisch für solche Systeme sind die sog. *Self-X-Eigenschaften*, wie z. B. selbstheilend, selbstschützend, selbstoptimierend und selbstkonfigurierend. Ein selbstorganisierendes System zeichnet sich dadurch aus, dass es Änderungen und Anpassungen seines Verhaltens eigenständig, ohne zentrale Kontrolle von einer Instanz außerhalb des Systems, vornehmen kann. Für zukünftige Forschungsarbeiten in diesem Zusammenhang ist die Konzeption und Umsetzung selbstorganisierender Service-orientierter Workflows relevant. Hierbei gilt es, Anforderungen an Monitoring- und Adaptions-Mechanismen neu zu definieren, sodass Workflows eigenständig in der Lage sind, sich selbst anzupassen und bei Bedarf neue Ressourcen zuzuschalten, um z. B. SLA-Verletzungen zu verhindern.

Der in dieser Arbeit vorgestellte Replanning-Mechanismus bezieht seine Informationen durch Monitoring-Mechanismen auf Service-Ebene. Für zukünftige Arbeiten von Bedeutung ist die Fragestellung, wie man Abweichungen vom geplanten Laufzeitverhalten der Web Services schon proaktiv erkennen kann. Repp et al. stellen in [113, 114] hierzu erste Ansätze vor, die Schichten übergreifend auf Basis von HTTP, TCP und IP frühzeitig Performanzanomalien erkennen.

Wenn zukünftig eine Vielzahl funktional ähnlicher Web Services verschiedener Anbieter zur Verfügung steht, so ermöglicht dies auch neue Geschäftsmodelle [132]. Unternehmen sind dann in der Lage, die benötige Funktionalität ihrer Unternehmenssoftware bedarfsgerecht in Form von Enterprise Resource Planning (ERP)-Services zu beziehen, anstatt monolithische ERP-Anwendungen eines bestimmten Anbieters nutzen zu müssen. Lizenzbasierte Preismodelle können somit durch Pay-per-use-Modelle ergänzt, wenn nicht sogar abgelöst werden. Service-Orientierung organisatorisch konsequent in Richtung service-orientierter Unternehmen weitergedacht, erfordert aber auch neue Governance-Ansätze [68], die Richtlinien zum Management des Service-Portfolios und der Serviceanbieter auf technischer, betriebswirtschaftlicher, regulatorischer und organisatorischer Ebene betrachten.

Literaturverzeichnis

[1] Aalst, W. M. P. v. d.; Hee, K. M. v.: *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2002.

[2] Aggarwal, R.; Verma, K.; Miller, J.; Milnor, W.: *Constraint Driven Web Service Composition in METEOR-S*. In: IEEE International Conference on Services Computing (SCC 2004), Shanghai, China, 2004, S. 23-30.

[3] Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V.: *Web Services. Concepts, Architectures and Applications.* Springer, Berlin, Heidelberg, 2004.

[4] Andrews, T.; Curbera, F.; Dholakia, H., et al.: *Business Process Execution Language for Web Services Version 1.1*, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, 2003, Abruf am 12.10.2006.

[5] ATIS Committee: *Accountability*, http://www.atis.org/tg2k/_accountability.html, 2001, Abruf am 09.05.2006.

[6] Balke, W.-T.; Badii, A.: *Assessing Web Services Quality for Call-by-Call Outsourcing*. In: 4th International Conference on Web Information Systems Engineering Workshops (WISEW 2003), 2003, S. 173-181.

[7] Becker, J.; Kahn, D.: *The Process in Focus*. In: Becker, J.; Kugeler, M.; Rosemann, M. (Hrsg.): Process Management. A Guide for the Design of Business Processes, Springer, Berlin, 2003, S. 1-12.

[8] Beimborn, D.; Franke, J.; Weitzel, T.: *Drivers and Inhibitors for Outsourcing Financial Processes – A Comparative Survey of Economies of Scale, Scope, and Skill*. In: 11th Americas Conference on Information Systems (AMCIS 2005), Omaha, NE, USA, 2005.

[9] Bellwood, T.; Capell, S.; Clement, L., et al.: *UDDI Version 3.0.2*. UDDI Spec Technical Committee Draft, http://uddi.org/pubs/uddi_v3.htm, OASIS, 2004, Abruf am 08.05.2006.

[10] Berbner, R.; Grollius, T.; Repp, N., et al.: *Management of Service-oriented Architecture (SoA) based Application Systems*. In: Enterprise Modelling and Information Systems Architectures - An International Journal, 2 (2007) 1, S. 14-25.

[11] Berbner, R.; Grollius, T.; Repp, N., et al.: *An approach for the Management of Service-oriented Architecture (SoA) based Application Systems*. In: Enterprise Modelling and Information Systems Architectures (EMISA 2005), Klagenfurt, Österreich, 2005, S. 208-221.

[12] Berbner, R.; Heckmann, O.; Mauthe, A.; Steinmetz, R.: *Eine Dienstgüte unterstützende Web Service-Architektur für flexible Geschäftsprozesse*. In: Wirtschaftsinformatik, 47 (2005) 4, S. 268-277.

[13] Berbner, R.; Heckmann, O.; Steinmetz, R.: *An Architecture for a QoS driven composition of Web Service based Workflows*. In: Networking and Electronic Commerce Research Conference (NAEC 2005), Riva Del Garda, Italien, 2005.

[14] Berbner, R.; Mauthe, A.; Steinmetz, R.: *Unterstützung dynamischer E-Finance-Geschäftsprozesse*. In: Konferenz Elektronische Geschäftsprozesse (EGP 2004), Klagenfurt, Österreich, 2004, S. 44-54.

[15] Berbner, R.; Spahn, M.; Repp, N.; Heckmann, O.; Steinmetz, R.: *An Approach for Replanning of Web Service Workflows*. In: 12th Americas Conference on Information Systems (AMCIS 2006), Acapulco, Mexiko, 2006.

[16] Berbner, R.; Spahn, M.; Repp, N.; Heckmann, O.; Steinmetz, R.: *Heuristics for QoS-aware Web Service Composition*. In: 4th IEEE International Conference on Web Services (ICWS 2006), Chicago, IL, USA, 2006, S. 72-82.

[17] Berbner, R.; Spahn, M.; Repp, N.; Heckmann, O.; Steinmetz, R.: *Dynamic Replanning of Web Service Workflows*. In: IEEE International Conference on Digital Ecosystems and Technologies 2007 (DEST 2007), Cairns, Australien, 2007.

[18] Berners-Lee, T.; Fielding, R.; Frystyk, H.: *Hypertext Transfer Protocol - HTTP 1.0*. Request for Comments 1945, http://www.ietf.org/rfc/rfc1945.txt, The Internet Society, Mai 1996, 1996, Abruf am 13.12.2006.

[19] Bichler, M.; Lin, K. J.: *Service-Oriented Computing*. In: IEEE Computer, 39 (2006) 3, S. 99-101.

[20] Buxmann, P.; Dirks, C.; Heintz, S.: *Zwischenbetriebliche Prozesse in der Automobilindustrie*. In: HMD - Praxis Wirtschaftsinformatik (1998) 200, S. 93.

[21] C-Lab: *Organic Computing*, http://www.c-lab.de/de/arbeitsgebiete/computing/organic-computing/index.html, 2007, Abruf am 10.02.2007.

[22] Canfora, G.; Penta, M. D.; Esposito, R.; Perfetto, F.; Villani, M. L.: *Service composition (re)binding driven by application–specific QoS*. In: 4th International Conference on Service-Oriented Computing (ICSOC 2006), Chicago, IL, USA, 2006, S. 141-152.

[23] Canfora, G.; Penta, M. D.; Esposito, R.; Villani, M. L.: *An approach for QoS-aware service composition based on genetic algorithms*. In: Genetic and Evolutionary Computation Conference (GECCO 2005), Washington DC, USA, 2005, S. 1069-1075.

[24] Canfora, G.; Penta, M. D.; Esposito, R.; Villani, M. L.: *QoS-Aware Replanning of Composite Web Services*. In: 3rd IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, USA, 2005, S. 121-129.

[25] Cardoso, J.: *Quality of Service and Semantic Composition of Workflows*. Ph.D. Thesis, University of Georgia, Department of Computer Science, Athens, GA, USA, 2002.

[26] Cardoso, J.; Sheth, A.; Miller, J.; Arnold, J.; Kochut, K.: *Quality of Service for Workflows and Web Service Processes*. In: Web Semantics: Science, Services and Agents on the World Wide Web, 1 (2004) 3, S. 281-308.

[27] Castellanos, M.; Casati, F.; Dayal, U.; Shan, M.-C.: *Intelligent Management of SLAs for Composite Web Services*. In: 3rd International Workshop on Databases in Networked Information Systems (DNIS 2003), Aizu, Japan, 2003, S. 158-171.

[28] CCITT: *CCITT Blue Book, VIII.2, Data Communication Networks: Services and Facilities, Interfaces.* Band 2, ITU, Genf, 1989.

[29] Chang, H.; Song, H.; Kim, W., et al.: *Simulation-Based Web Service Composition: Framework and Performance Analysis*. In: 3rd Asian Simulation Conference (AsiaSim 2004), Jeju Island, Korea, 2004, S. 352-360.

[30] Charfi, A.: *Aspect-Oriented Workflow Languages: AO4BPEL and Applications*. Dissertationsschrift, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Deutschland, 2007.

[31] Charfi, A.; Mezini, M.: *Hybrid Web Service Composition: Business Processes Meet Business Rules*. In: 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, NY, USA, 2004, S. 30-38.

[32] Charfi, A.; Mezini, M.: *Using Aspects for Security Engineering of Web Service Compositions*. In: 3rd IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, USA, 2005, S. 59-66.

[33] Charfi, A.; Mezini, M.: *Aspect-Oriented Workflow Languages*. In: International Conference on Cooperative Information Systems (CoopIS 2006), Montpellier, Frankreich, 2006, S. 183-200.

[34] Chen, H.; Yu, T.; Lin, K.-J.: *QCWS: An Implementation of QoS-Capable Multimedia Web Services*. In: 5th IEEE International Symposium on Multimedia Software Engineering (ISMSE 2003), Taichung, Taiwan, 2003, S. 38.

[35] Clark, M.; Fletcher, P.; Hanson, J. J., et al.: *Web Services Business Strategies and Architectures.* Wrox Press, 2002.

[36] Domschke, W.; Drexl, A.: *Einführung in Operations Research.* 4. Aufl., Springer, Berlin, Heidelberg, 1998.

[37] Dostal, W.; Jeckle, M.; Melzer, I.: *Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis.* Spektrum Akademischer Verlag, 2005.

[38] Eberhart, A.; Fischer, S.: *Web Services. Grundlagen und praktische Umsetzung mit J2EE und .NET.* Carl-Hanser-Verlag, München, Wien, 2003.

[39] Eckert, C.: *IT-Sicherheit.* 4. Aufl., R. Oldenbourg Verlag, Wien, München, 2006.

[40] Elgass, P.; Krcmar, H.: *Computergestützte Geschäftsprozessplanung*. In: Information Management (1993) 1, S. 42-49.

[41] Endrei, M.; Ang, J.; Arsanjani, A., et al.: *Patterns: Service-Oriented Architecture and Web Services.* IBM Redbook, 2004.

[42] Erl, T.: *Service-Oriented Architecture. A Field Guide to Integrating XML and Web Services.* Prentice Hall, Upper Saddle River, 2004.

[43] Esfandiari, B.; Tosic, V.: *Requirements for Web Service Composition Management*. In: 11th HP Open View University Association (HP-OVUA 2004) Workshop, Paris, Frankreich, 2004.

[44] Fielding, R.; Gettys, J.; Mogul, J., et al.: *Hypertext Transfer Protocol - HTTP/1.1*. Request for Comments 2616, http://www.ietf.org/rfc/rfc2616.txt, The Internet Society, Juni 1999, 1999, Abruf am 28.12.2006.

[45] Foster, I.; Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure.* Elsevier, 2004.

[46] Fremantle, P.; Weerawarana, S.; Khalaf, R.: *Enterprise Services*. In: Communications of the ACM, 45 (2002) 10, S. 77-82.

[47] Friese, T.; Müller, J. P.; Freisleben, B.: *Integrating Peer-to-Peer Technology into a Web Service Environment*. In: Multikonferenz Wirtschaftsinformatik (MKWI 2006), Workshop on P2P and Grid Computing, Passau, Deutschland, 2006.

[48] Fröschl, F.: *Vom IuK-Outsourcing zum Business Process Outsourcing*. In: Wirtschaftsinformatik, 41 (1999) 5, S. 458-460.

[49] Gouscos, D.; Kalikakis, M.; Georgiadis, P.: *An Approach to Modeling Web Service QoS and Provision Price*. In: 4th International Conference on Web Information Systems Engineering Workshops (WISEW 2003), Rom, Italien, 2003, S. 121-130.

[50] Heckmann, O.: *A System-oriented Approach to Efficiency and Quality of Service for Internet Service Providers*. Dissertationsschrift, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Deutschland, 2004.

[51] Heckmann, O.: *The Competitive Internet Service Provider.* Wiley & Sons, 2006.

[52] Heckmann, O.: *Efficiency and Quality of Service of IP Networks*. In: it-Information Technology Journal, 3 (2006) 6, S. 177-180.

[53] Henrici, D.; Müller, J.: *Security in Service-Oriented Architectures*. In: 19. DFN-Arbeitstagung, Düsseldorf, Deutschland, http://dspace.icsy.de/handle/123456789/155, 2005, Abruf am 12.12.2006.

[54] Hess, A.; Humm, B.; Voß, M.: *Regeln für serviceorientierte Architekturen hoher Qualität*. In: Informatik Spektrum, 29 (2006) 6, S. 395-411.

[55] Hillenbrand, M.; Götze, J.; Müller, P.: *Venice – A Lightweight Service Grid*. In: 32nd EUROMICRO Conference, Cavtat, Kroatien, http://dspace.icsy.de/handle/123456789/166, 2006, Abruf am 09.02.2007.

[56] Hillenbrand, M.; Götze, J.; Müller, P.: *Web Services Directory based on Peer-to-Peer Technology*. In: 32nd EUROMICRO Conference, Cavtat, Kroatien, http://dspace.icsy.de/handle/123456789/165, 2006, Abruf am 10.02.2007.

[57] Hillenbrand, M.; Götze, J.; Zhang, G.; Müller, P.: *A Lightweight Service Grid based on Web Services and Peer-to-Peer*. In: 15. ITG/ GI-Fachtagung Kommunikation in Verteilten Systemen (KIVS 2007), Bern, Schweiz, 2007.

[58] Hillenbrand, M.; Müller, P.: *Web Services and Peer-to-Peer*. In: Steinmetz, R.; Wehrle, K. (Hrsg.): Peer-to-Peer Systems and Applications, Springer, 2005.

[59] Huhns, M.; Singh, M. P.: *Service-Oriented Computing: Key Concepts and Principles*. In: IEEE Internet Computing, 9 (2005) 1, S. 75- 81.

[60] Humm, B.; Juwig, O.: *Eine Normalform für Services*. In: Software Engineering 2006, Fachtagung des GI-Fachbereichs Softwaretechnik, Leipzig, Deutschland, 2006, S. 99-110.

[61] Huth, C.: *Groupware-basiertes Ad-hoc-Workflow-Management: Das GroupProcess-System*. Dissertationsschrift, Universität Paderborn, Fakultät für Wirtschaftswissenschaften, Paderborn, Deutschland, 2004.

[62] Jablonski, S.; Bussler, C.: *Workflow Management Modeling: Concepts, Architecture and Implementation.* International Thomson Computer Press, London, 1996.

[63] Jäger, M. C.: *Optimising Quality-of-Service for the Composition of Electronic Services*. Dissertationsschrift, Technische Universität Berlin, Fakultät Elektrotechnik und Informatik, Berlin, Deutschland, 2007.

[64] Jäger, M. C.; Ladner, H.: *A Model for the Aggregation of QoS in WS Compositions Involving Redundant Services*. In: Journal of Digital Information Management, 4 (2005) 1, S. 44-49.

[65] Jäger, M. C.; Mühl, G.; Golze, S.: *QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms*. In: Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Zypern, 2005, S. 646-661.

[66] Jäger, M. C.; Rojec-Goldmann, G.; Mühl, G.: *QoS Aggregation for Service Composition using Workflow Patterns*. In: 8th International Enterprise Distributed Object Computing Conference (EDOC 2004), Monterey, CA, USA, 2004, S. 149-159.

[67] Jäger, M. C.; Rojec-Goldmann, G.; Mühl, G.: *QoS-Aggregation in Web Service Compositions*. In: IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE 2005) Hong Kong, China, 2005, S. 181-185.

[68] Johannsen, W.; Goeken, M.: *IT-Governance - neue Aufgaben des IT-Managements*. In: HMD-Praxis der Wirtschaftsinformatik, 43 (2006) 250, S. 7-20.

[69] Juric, M. B.; Mathew, B.; Sarang, P.: *Business Process Execution Language for Web Services: BPEL and BPEL4WS.* Packt Publishing Ltd., Birmingham, 2004.

[70] Kalepu, S.; Krishnaswamy, S.; Loke, S. W.: *Verity: A QoS Metric for Selecting Web Services and Providers*. In: 4th International Conference on Web Information Systems Engineering Workshops (WISEW 2003), Rom, Italien, 2003, S. 131-139.

[71] Karastoyanova, D.; Houspanossian, A.; Cilia, M.; Leymann, F.; Buchmann:, A. P.: *Extending BPEL for Run Time Adaptability*. In: 9th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2005), Enschede, Niederlande, 2005, S. 15-26.

[72] Karastoyanova, D.; Leymann, F.; Buchmann:, A. P.: *An Approach to Parameterizing Web Service Flows*. In: 3rd International Conference on Service-Oriented Computing (ICSOC 2005), Amsterdam, Niederlande, 2005, S. 533-538.

[73] Karastoyanova, D.; Leymann, F.; Nitzsche, J.; Wetzstein, B.; Wutke, D.: *Parameterized BPEL Processes: Concepts and Implementation*. In: 4th International Conference on Business Process Management (BPM 2006), Wien, Österreich, 2006, S. 471-476.

[74] Keller, W.: *Enterprise Application Integration. Erfahrungen aus der Praxis.* dpunkt Verlag, Heidelberg, 2002.

[75] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P.: *Optimization by Simulated Annealing*. In: Science, 220 (1983) 4598, S. 671-680.

[76] Klein, R.; Scholl, A.: *Planung und Entscheidung.* Verlag Vahlen, München, 2004.

[77] Koetzle, L.; Rutstein, C.; Liddell, H.; Buss, C.; Nakashima, T.: *Reducing Integration's Cost*, http://www.forrester.com/go?docid=11981, Forrester Research Report, 2001, Abruf am 26.10.2006.

[78] König, W.; Beimborn, D.; Franke, J.; Weitzel, T.: *Sourcing von Finanzprozessen – Ein Modell zur simultanen Bewertung von Economies of Scale und Scope*. In: Internationale Tagung Wirtschaftsinformatik (WI 2005), Bamberg, Deutschland, 2005, S. 1691-1714.

[79] Kossmann, D.; Leymann, F.: *Web Services*. In: Informatik-Spektrum, 27 (2004) 2, S. 117-128.

[80] Krafzig, D.; Banke, K.; Slama, D.: *Enterprise SOA. Service-Oriented Architecture. Best Practices.* Prentice Hall, Upper Saddle River, 2005.

[81] Kreger, H.: *Web Services Conceptual Architecture (WSCA 1.0)*, http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, 2001, Abruf am 12.12.2005.

[82] Laarhoven, P. J. M.; Aarts, E. H. L.: *Simulated annealing: theory and applications.* Kluwer Academic, Norwell, 1987.

[83] Leymann, F.: *Web Services Flow Language (WSFL 1.0)*, http://xml.coverpages.org/WSFL-Guide-200110.pdf, 2001, Abruf am 30.01.2007.

[84] Leymann, F.; Roller, D.: *Production Workflow, Concepts and Techniques.* Prentice Hall, Upper Saddle River, 2000.

[85] Leymann, F.; Roller, D.: *Web Services: Business Processes in a Web Services World. A quick overview of BPEL4WS*, http://www-106.ibm.com/developerworks/webservices/library/ws-bpelwp, IBM DeveloperWorks., 2002, Abruf am 26.12.2006.

[86] Lindert, F.; Wiedeler, M.: *Organisationsübergreifendes Geschäftsprozessmanagement*. In: IT Information Technology, 46 (2004) 4, S. 175-183.

[87] Liu, Y.; Ngu, A. H.; Zeng, L. Z.: *QoS computation and Policing in dynamic Web Service Selection*. In: 13th International World Wide Web Conference (WWW 2004), Alternate Track Papers and Posters, New York, NY, USA, 2004, S. 66-73.

[88] Ludwig, H.; Keller, A.; Dan, A.; King, R. P.; Franck, R.: *Web Service Level Agreement (WSLA) Language Specification*, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, IBM Corporation, 28.01.2003, 2003, Abruf am 10.05.2006.

[89] Ma, W.; Tosic, V.; Esfandiari, B.; Pagurek, B.: *Extending Apache Axis for Monitoring of Web Service Offerings*. In: IEEE Workshop on Business Services Networks (BSN 2005), Hong Kong, China, 2005, S. 44-51.

[90] Machiraju, V.; Sahai, A.; Moorsel, A. v.: *Web Services Management Network: An Overlay Network for Federated Service Management*, http://www.hpl.hp.com/techreports/2002/HPL-2002-234.pdf, HP Laboratories, 21.08.2002, 2002, Abruf am 14.12.2006.

[91] Malone, T. W.; Crowston, K.; Lee, J., et al.: *Tools for Inventing Organizations: Toward a Handbook of Organizational Processes*. In: Management Science, 45 (1999) 3, S. 425-443.

[92] Mani, A.; Nagarajan, A.: *Understanding quality of service for Web Services*, http://www-106.ibm.com/developerworks/webservices/library/ws-quality.html, IBM developerWorks, 01.01.2002, 2002, Abruf am 07.05.2006.

[93] Mathew, G. E.; Shields, J.; Verma, V.: *QoS Based Pricing for Web Services*. In: 5th International Conference on Web Information Systems Engineering Workshops (WISEW 2004), Brisbane, Australien, 2004, S. 264–275.

[94] Mauthe, A.; Hutchison, D.: *Peer-to-Peer Computing: Systems, Concepts and Characteristics*. In: Praxis in der Informationsverarbeitung & Kommunikation (PIK), 26 (2003) 2, S. 60-64.

[95] Mauthe, A.; Thomas, P.: *Professional Content Management Systems.* John Wiley & Sons 2004.

[96] Maximilien, E. M.; Singh, M. P.: *Conceptual Model of Web Service Reputation*. In: ACM SIGMOD Record, 31 (2002) 4, S. 36-41.

[97] Maximilien, E. M.; Singh, M. P.: *Toward autonomic Web Services Trust and Selection*. In: 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, NY, USA, 2004, S. 212-221.

[98] Menascé, D. A.: *QoS Issues in Web Services*. In: IEEE Internet Computing, 6 (2002) 6, S. 72-75.

[99] Menascé, D. A.: *Composing Web Services: A QoS View*. In: IEEE Internet Computing, 8 (2004) 6, S. 88-90.

[100] Michalewicz, Z.: *Genetic algorithms + data structures = evolution programs.* 2. Aufl., Springer, Berlin, New York, 1994.

[101] On, G.; Schmitt, J.; Steinmetz, R.: *On Availability QoS for Replicated Multimedia Service and Content*. In: Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems: Protocols and Systems for Interactive Distributed Multimedia (IDMS/ PROMS 2002), Coimbra, Portugal, 2002, S. 313-326.

[102] Open Grid Services Architecture Working Group: *Defining the Grid: A Roadmap for OGSA Standards*, http://www.gridforum.org/documents/GFD.53.pdf, 16.09.2005, 2005, Abruf am 16.02.2007.

[103] Oram, A.: *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly Media, 2001.

[104] Ortner, E.: *Repository Systems*. In: Informatik Spektrum, 22 (1999) 4, S. 235-251.

[105] Ortner, E.: *Terminologiebasierte, komponentenorientierte Entwicklung von Anwendungssystemen*. In: 2. Workshop komponentenorientierte betriebliche Anwendungssysteme (WKBA 2), Wien, Österreich, 2002, S. 1-20.

[106] Papazoglou, M. P.: *Service-Oriented Computing: Concepts, Characteristics and Directions*. In: 4th International Conference on Web Information Systems Engineering (WISE 2003), Rom, Italien, 2003, S. 3-12.

[107] Parnas, D. L.: *On the Criteria To Be Used in Decomposing Systems into Modules*. In: Communications of the ACM, 15 (1972) 12, S. 1053-1058.

[108] Patel, C.; Supekar, K.; Lee, Y.: *A QoS Oriented Framework for Adaptive Management of Web Service Based Workflows*. In: 14th International Conference on Database and Expert Systems Applications (DEXA 2003), Prag, Tschechische Republik, 2003, S. 826-835.

[109] Paton, N. W.: *Active Rules in Database Systems.* Springer, 1998.

[110] Pisinger, D.: *A minimal algorithm for the Multiple-choice Knapsack Problem*. In: European Journal of Operational Research, 83 (1995) 2, S. 394-410.

[111] Preuner, G.; Schrefl, M.: *Integration of Web Services into Workflows through a Multi-Level Schema Architecture*. In: 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002), Newport Beach, CA, USA, 2002, S. 51-60.

[112] Ran, S.: *A Model for Web Services Discovery with QoS*. In: ACM SIGecom Exchanges, 4 (2003) 1, S. 1-10.

[113] Repp, N.; Berbner, R.; Heckmann, O.; Steinmetz, R.: *A Cross-Layer Approach to Performance Monitoring of Web Services*. In: ECOWS Workshop on Emerging Web Services Technology (WEWST 2006), Zürich, Schweiz, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-234/paper2.pdf, 2006, Abruf am 09.02.2007.

[114] Repp, N.; Heckmann, O.; Berbner, R., et al.: *Der Einfluß von Transportschicht-Anomalien auf die Performanz von Web Services*. In: 15. ITG/ GI-Fachtagung Kommunikation in Verteilten Systemen (KIVS 2007), Bern, Schweiz, 2007, S. 117-122.

[115] Richter, J.-P.; Haller, H.; Schrey, P.: *Serviceorientierte Architektur*. In: Informatik Spektrum, 28 (2005) 5, S. 413-416.

[116] Riedl, R.: *Begriffliche Grundlagen des Business Process Outsourcing*. In: Information Management & Consulting, 18 (2003) 3, S. 6-10.

[117] Sahai, A.; Durante, A.; Machiraju, V.: *Towards Automated SLA Management for Web Services*, http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf, HP Laboratories, 11.07.2002, 2002, Abruf am 03.05.2006.

[118] Sahai, A.; Machiraju, V.; Sayal, M.; Moorsel, A. P. A. v.; Casati, F.: *Automated SLA Monitoring for Web Services*. In: 13th IFIP/ IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications (DSOM 2002), 2002, S. 28-41.

[119] Satzinger, J. W.; Jackson, R.; Burd, S. D.: *Systems Analysis and Design in a Changing World.* MIS Series, 2004.

[120] Scheer, A.-W.: *Wirtschaftsinformatik - Referenzmodelle für industrielle Geschäftsprozesse.* 7. Aufl., Springer, Berlin, 1997.

[121] Schmidt, R.: *Web Services Based Architectures to Support Dynamic Inter-organizational Business Processes*. In: 1st IEEE International Conference on Web Services (ICWS 2003), Erfurt, Deutschland, 2003, S. 123-136.

[122] Schmietendorf, A.; Dumke, R.; Reitz, D.: *SLA Management - Challenges in the Context of Web-Service-Based Infrastructures*. In: 2nd IEEE International Conference on Web Services (ICWS 2004), San Diego, CA, USA, 2004, S. 606-613.

[123] Schmitt, J. B.: *Heterogeneous Network Quality of Service Systems.* Kluwer Academic, Boston, 2001.

[124] Schwind, M.; Gujo, O.; Stockheim, T.: *Dynamic Resource Prices in a Combinatorial Grid System*. In: Joint 8th IEEE Conference on E-Commerce Technology (CEC 2006) and 3rd IEEE Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2006), San Francisco, CA, USA, 2006, S. 49.

[125] Seibold, H.: *IT-Risikomanagement.* 1. Aufl., R. Oldenbourg Verlag, Wien, München, 2006.

[126] Serugendo, G. D. M.; Foukia, N.; Hassas, S., et al.: *Self-Organisation: Paradigms and Applications*. In: 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), Melbourne, Australien, 2003, S. 1-19.

[127] Smith, M.; Friese, T.; Engel, M.; Freisleben, B.: *Countering Security Threats in Service-oriented on-demand Grid Computing using sandboxing and trusted Computing Techniques*. In: Journal of Parallel and Distributed Computing, 66 (2006) 9, S. 1189-1204.

[128] Smith, M.; Friese, T.; Engel, M., et al.: *Security Issues in On-Demand Grid and Cluster Computing*. In: 6th IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW 2006), Singapure, 2006, S. 24-38.

[129] Spahn, M.; Berbner, R.; Heckmann, O.; Mauthe, A.; Steinmetz, R.: *WSQoSX – Eine dienstgütebasierte Web-Service-Architektur*. Technische Universität Darmstadt, Darmstadt, Deutschland, Technical Report KOM 07-2004, 2004.

[130] Spahn, M.; Berbner, R.; Heckmann, O.; Steinmetz, R.: *Ein heuristisches Optimierungsverfahren zur dienstgütebasierten Komposition von Web-Service-Workflows*. Technische Universität Darmstadt, Darmstadt, Deutschland, Technical Report, KOM 02-2006, 2006.

[131] Steinmetz, R.: *Multimedia-Technologie: Grundlagen, Komponenten und Systeme.* 3. Aufl., Springer, Berlin, Heidelberg, 2000.

[132] Steinmetz, R.; Berbner, R.; Martinovic, I.: *Web Services zur Unterstützung flexibler Geschäftsprozesse in der Finanzwirtschaft*. In: Sokolovsky, Z.; Loeschenkohl, S. (Hrsg.): Handbuch Industrialisierung der Finanzwirtschaft, Gabler, Wiesbaden, 2004, S. 641-654.

[133] Steinmetz, R.; Nahrstedt, K.: *Multimedia Applications.* Springer, 2004.

[134] Steinmetz, R.; Nahrstedt, K.: *Multimedia Systems.* Springer, 2005.

[135] Steinmetz, R.; Wehrle, K.: *Peer-to-Peer-Networking & -Computing*. In: Informatik-Spektrum, 27 (2004) 1, S. 51-54.

[136] Steinmetz, R.; Wehrle, K.: *Peer-to-Peer Systems and Applications.* Springer, 2005.

[137] Tanenbaum, A. S.: *Computernetzwerke.* 3 Aufl., Prentice Hall, München, 1998.

[138] Thatte, S.: *XLANG. Web Services for Business Process Design*, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001, Abruf am 30.01.2007.

[139] Tian, M.: *QoS Integration in Web Services with the WS-QoS Framework*. Dissertationsschrift, Freie Universität Berlin, Fachbereich Mathematik und Informatik, Berlin, Deutschland, 2005.

[140] Tian, M.; Gramm, A.; Naumowicz, T.; Ritter, H.; Freie, J. S.: *A Concept for QoS Integration in Web Services*. In: 4th International Conference on Web Information Systems Engineering Workshops (WISEW 2003), Rom, Italien, 2003, S. 149-155.

[141] Tian, M.; Gramm, A.; Ritter, H.; Schiller, J.; Winter, R.: *A Survey of current Approaches towards Specification and Management of Quality of Service for Web Services*. In: PIK - Praxis der Informationsverarbeitung und Kommunikation, 27 (2004) 3, S. 132-139.

[142] Tosic, V.: *Service Offerings for XML Web Services and Their Management Applications*. Ph.D. Thesis, Carleton University, Department of Systems and Computer Engineering, Ottawa, Kanada, 2004.

[143] Tosic, V.; Esfandiari, B.; Pagurek, B.; Patel, K.: *On the Dynamic Manipulation of Classes of Service for XML Web Services*. In: Workshop on Web Services, e-Business, and the Semantic Web (WES 2002), Toronto, Kanada, 2002.

[144] Tosic, V.; Lutfiyya, H.; Tang, Y.: *Extending Web Service Offerings Infrastructure (WSOI) for Management of Mobile/Embedded XML Web Services*. In: Joint 8th IEEE Conference on E-Commerce Technology (CEC 2006) und 3rd IEEE Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2006), San Francisco, CA, USA, 2006, S. 87.

[145] Tosic, V.; Ma, W.; Pagurek, B.; Esfandiari, B.: *Web Services Offerings Infrastructure (WSOI) - A Management Infrastructure for XML Web Services*. In: IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Südkorea, 2004, S. 817-830.

[146] Tosic, V.; Pagurek, B.; Patel, K.; Esfandiari, B.; Ma, W.: *Management Applications of the Web Service Offerings Language (WSOL)*. In: 15th International Conference on Advanced Information Systems Engineering (CAISE 2003), Velden, Österreich, 2003, S. 468-484.

[147] Verma, K.; Sivashanmugam, K.; Sheth, A., et al.: *METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services*. In: Journal of Information Technology and Management, 6 (2005) 1, S. 17-39.

[148] W3C Working Group: *Web Services Description Language (WSDL) 1.1*. W3C Note, http://www.w3.org/TR/wsdl, W3C, 15.03.2001, 2001, Abruf am 13.05.2006.

[149] W3C Working Group: *QoS for Web Services: Requirements and Possible Approaches*. W3C Working Group Note, http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/, W3C, 25.11.2003, 2003, Abruf am 10.10.2006.

[150] W3C Working Group: *SOAP Version 1.2 Part 0: Primer*. W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, W3C, 24.06.2003, 2003, Abruf am 13.12.2006.

[151] W3C Working Group: *SOAP Version 1.2 Part 1: Messaging Framework*. W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/, W3C, 24.06.2003, 2003, Abruf am 13.05.2006.

[152] W3C Working Group: *SOAP Version 1.2 Part 2: Adjuncts*. W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part2-20030624/, W3C, 24.06.2003, 2003, Abruf am 13.05.2006.

[153] W3C Working Group: *Web Services Architecture*, http://www.w3.org/TR/ws-arch/, W3C, 11.02.2004, 2004, Abruf am 04.02.2007.

[154] W3C Working Group: *Web Services Architecture Requirements*. W3C Working Group Note, http://www.w3.org/TR/wsa-reqs/, W3C, 14.02.2004, 2004, Abruf am 07.04.2006.

[155] Wang, D.; Bayer, T.; Frotscher, T.; Teufel, M.: *Java Web Services mit Apache Axis.* Software & Support Verlag GmbH, Frankfurt, 2004.

[156] Weerawarana, S.; Curbera, F.; Leymann, F.; Storey, T.; Ferguson, D. F.: *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall, Upper Saddle River, 2005.

[157] Westerinen, A.; Schnizlein, J.; Strassner, J., et al.: *Terminology for Policy-Based Management. RFC 3198*, http://www.rfc-archive.org/getrfc.php?rfc=3198, 2001, Abruf am 13.05.2006.

[158] Wirth, N.: *Algorithms + Data Structures = Programs.* Prentice Hall, Englewood Cliffs, 1976.

[159] Wombacher, A.: *Decentralized establishment of consistent, multi-lateral collaborations*. Dissertationsschrift, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Deutschland, 2005.

[160] Workflow Management Coalition (WFMC): *The Workflow Reference Model*, http://www.wfmc.org/standards/docs/tc003v11.pdf, 19.01.1995, 1995, Abruf am 04.02.2007.

[161] Workflow Management Coalition (WFMC): *Workflow standard - Terminology & glossary*, http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf, Februar 1999, 1999, Abruf am 04.02.2007.

[162] Wu, Z.; Luo, J.: *The Measurement Model of Grid QoS*. In: 10th International Conference on Computer Supported Cooperative Work in Design (CSCW 2006), Banff, Alberta, Kanada, 2006, S. 1-6.

[163] Wyssusek, B.: *Geschäftsprozessmodell, Geschäftsprozessmodellierung*. In: Mertens, P.; Back, A.; Thome, R. et al. (Hrsg.): Lexikon der Wirtschaftsinformatik, Springer, Berlin, 2001, S. 210-211.

[164] Yang, J.; Papazouglou, M. P.; van den Heuvel, W. J.: *Tackling the Challenges of Service Composition in E-Marketplaces*. In: 12th International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/ E-Business Systems (RIDE 2002), San Jose, CA, USA, 2002, S. 125-133.

[165] Yu, T.; Lin, K.-J.: *The Design of QoS Broker Algorithms for QoS-Capable Web Services*. In: 1st International Conference on e-Technology, e-Commerce and e-Service (EEE 2004), Taipei, Taiwan, 2004, S. 17-24.

[166] Yu, T.; Lin, K.-J.: *Service Selection Algorithms for Web Services with End-to-End QoS Constraints*. In: IEEE International Conference on E-Commerce Technology (CEC 2004), San Diego, CA, USA, 2004, S. 129-136.

[167] Yu, T.; Lin, K.-J.: *A Broker-Based Framework for QoS-Aware Web Service Composition*. In: 2nd International Conference on e-Technology, e-Commerce, and e-Services (EEE 2005), Hong Kong, China, 2005, S. 22-29.

[168] Yu, T.; Lin, K.-J.: *Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints*. In: 3rd International Conference on Service-Oriented Computing (ICSOC 2005), Amsterdam, Niederlande, 2005, S. 130-143.

[169] Yu, T.; Lin, K.-J.: *QCWS: An Implementation of QoS-capable Multimedia Web Services*. In: Multimedia Tools and Applications 30 (2006) 2, S. 165-187.

[170] Yu, T.; Lin, K. J.: *Adaptive algorithms for Finding Replacement Services in Autonomic Distributed Business Processes*. In: 7th International Symposium on Autonomous Decentralized Systems (ISADS 2005), Chengdu, China, 2005, S. 427-433.

[171] Zeng, L.; Benatallah, B.; Dumas, M.; Kalagnanam, J.; Sheng, Q. Z.: *Quality Driven Web Services Composition*. In: 12th International Conference on World Wide Web (WWW 2003), Budapest, Ungarn, 2003, S. 411-421.

[172] Zeng, L.; Benatallah, B.; Ngu, A., et al.: *QoS-aware Middleware for Web Service composition*. In: IEEE Transactions on Software Engineering, 30 (2004) 5, S. 311-328.

Abkürzungsverzeichnis

3DES	Triple Data Encryption Standard
AM	Adaptability Manager
API	Application Programming Interface
B&B	Branch and Bound
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPO	Business Process Outsourcing
CCITT	Commité Consultatif Internationale Télégrafique et Téléfonique
CSV	Comma Separated Value
DES	Data Encryption Standard
DOM	Document Object Model
EAI	Enterprise Application Integration
ECA	Event Condition Action
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
GA	Genetische Algorithmen
GLOP	(Gemischt) Ganzzahliges lineares Optimierungsproblem
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDEA	International Data Encryption Algorithm
IDL	Interface Definition Languag
IP	Internet Protocol
ISDN	Integrated Services Digital Network
J2SE	Java 2 Platform Standard Edition
JAXR	Java API for XML Registries
LMCKP	Linear Multiple-Choice Knapsack Problem

LOP	Lineares Optimierungsproblem
LP	Lineare Programmierung
MCKP	Multiple-Choice Knapsack Problem
MCOP	Multi-Constraint Optimal Path
METEOR	Managing End-to-End Operations
METEOR-S	METEOR for Semantic Web Services
MIME	Multipurpose Internet Mail Extensions
MIP	Mixed Integer Programming
MMKP	Multi-Dimension Multi-Choice 0-1 Knapsack Problem
OASIS	Organization for the Advancement of Structured Information Standards
OGSA	Open Grid Service Architecture
PHP	PHP Hypertext Preprocessor
QCWS	QoS-capable Web Service Architecture
QoS	Quality of Service
RPC	Remote Procedure Call
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SOA	Service-orientierte Architektur
SQL	Structured Query Language
SWR	Stochastic Workflow Reduction
TCP	Transmission Control Protocol
TTR	Time-to-Repair
UDDI	Universal Discovery, Description and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USID	Unique Service Identifier
W3C	World Wide Web Consortium

WFMC	Workflow Management Coalition
WSAF	Web Service Agent Framework
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
WSFL	Web Services Flow Language
WSID	Web Service Identifier
WSLA	Web Service Level Agreement
WSML	Web Services Management Language
WSOI	Web Service Offerings Infrastructure
WSOL	Web Service Offerings Language
WSQoSX	Web Services Quality of Service Architectural Extension
XML	Extensible Markup Language

Verwendete Bezeichner

Δ	Prozentuale Verschlechterung des Zielfunktionswertes bei H3_SIMUL_ANNEAL
α	Signifikanzniveau
λ	Temperaturparameter bei H3_SIMUL_ANNEAL
AP	Ausführungsplan eines Workflows
b_λ	Prozentsatz, um den der Temperaturparameter λ bei H3_SIMUL_ANNEAL gesenkt wird
c^+, $c^\bullet$, $c^{\min}$	Nebenbedingung für einen (additiv, multiplikativ bzw. durch den Minimaloperator aggregierten) Gesamtparameter
e	Abweichungsfaktor, der im Intervall $[\underline{e};\overline{e}]$ liegt
$F(\vec{x})$	Zielfunktionswert des Optimierungsproblems
$[\underline{h};\overline{h}]$	Intervall, in dem die Parameterwerte $p_{i,j,k}$ liegen
i	Kategorie funktional identischer Web Services
j	Position eines Web Service $WS_{i,j}$ innerhalb der Kategorie i
k	Index eines Parameters des Web Service $WS_{i,j}$
K	Anzahl der Parameter, die einen Web Service $WS_{i,j}$ beschreiben
$[\underline{l},\overline{l}]$	Intervall, in dem der tatsächliche Parameterwert p' liegt
m	Anzahl der Web Services pro Kategorie
n	Anzahl der Prozessschritte eines Workflows
p, p'	Geplanter bzw. tatsächlicher Parameterwert
$p_{i,j,k}$	Parameterwert des Parameters k eines Web Service $WS_{i,j}$
$p^+_{i,j}$, $p^\bullet_{i,j}$, $p^{\min}_{i,j}$	Parameterwert (additiv, multiplikativ bzw. mit dem Minimaloperator aggregiert) eines Web Service $WS_{i,j}$
PS_i	i. Prozessschritt eines Workflows
$q_{i,j,k}$	normalisierter Parameterwert des Parameters k eines Web Service $WS_{i,j}$
r	Wert, um den die (vom WorkflowOptimizer) erzeugten Parameterwerte gleichverteilt variieren
$s_{i,j}$	Score eines Web Service $WS_{i,j}$
t_s, t_h	Rechendauer Solver bzw. Rechendauer Heuristik
U	Nutzen eines Web Service Workflows
v	Faktor zur Berechnung der Anzahl der Iterationen bei H2_SWAP bzw. H3_SIMUL_ANNEAL
$w_{i,k}$	Gewichtung eines Parameters k innerhalb einer Kategorie i
w^+, $w^\bullet$, $w^{\min}$	Gewichtung eines (additiv, multiplikativ, mit dem Minimaloperator aggregierten) Gesamtparameters
$WS_{i,j}$	Web Service j ist der Kategorie i zugeordnet

$x_{i,j}$	Binäre Variable, die aussagt, ob der Web Service $WS_{i,j}$ an Position i im Ausführungsplan enthalten ist ($x_{i,j} = 1$**)** oder nicht ($x_{i,j} = 0$). Nach der LP-Relaxation gibt $x_{i,j}$ an, mit welchem Anteil $WS_{i,j}$ an Position i im Ausführungsplan enthalten ist.
x^+, $x^\bullet$, $x^{\min}$	Gesamtparameterwert eines (additiv, multiplikativ bzw. mit dem Minimaloperator aggregierten) Gesamtparameters
z	Wahrscheinlichkeit, mit der bei H3_SIMUL_ANNEAL eine Verschlechterung akzeptiert wird.

Anhang

A WSQoSX

Im Folgenden werden als Ergänzung zu Abschnitt 3.5 weitere Teile der Implementierung von WSProxy (siehe Abschnitt A.1), WSPortal (siehe Abschnitt A.2) und WSProxyAdmin (siehe Abschnitt A.3) vorgestellt. Diese drei Komponenten sind die Hauptbestandteile von WSQoSX.

A.1 WSProxy

In Abschnitt A.1.1 werden die Pakete und Klassen des WSProxy aufgeführt. Die Modellierung der SLAs im Rahmen von WSQoSX ist Gegenstand von Abschnitt A.1.2. Die beim Logging verwendeten Status-Codes können Abschnitt A.1.3 entnommen werden.

A.1.1 Klassenübersicht

Tabelle 15 enthält eine Übersicht der Pakete und der jeweiligen Klassen des WSProxy.

Paket	Klassen

`de.tud.kom.dynws.proxy`	`WSProxy`
	`WSProxyClientThread`
	`WSProxyDaemonThread`
	`WSProxyHTTPRequest`

`de.tud.kom.dynws.proxy.history`	`WSHistory`
	`WSWarnings`
	`WSWarningsFactory`
	`WSWarningsPortal`
	`WSWarningsStatic`

`de.tud.kom.dynws.proxy.router`	`WSRouter`
	`WSRouterRemoteTimeoutException`
	`WSRouterUnroutableException`

`de.tud.kom.dynws.proxy.search`	`WSSearch`
	`WSSearchFactory`
	`WSSearchPortal`
	`WSSearchStatic`

`de.tud.kom.dynws.proxy.util`	`ChunkedInputStream`
	`CopyWrapInputStream`
	`CopyWrapOutputStream`
	`DBHelper`

Tabelle 15: Pakete und Klassen des WSProxy

A.1.2 Verarbeitung der SLA-Dokumente

Das Format eines SLA-Dokumentes im Rahmen von WSQoSX wird durch eine Untermenge der XML-basierten Sprache *Web Service Level Agreement (WSLA)* [88], die von IBM zur

Spezifikation von SLA-Dokumenten entwickelt wurde, definiert. Der Aufbau eines SLA-Dokumentes kann der Abbildung 45 entnommen werden.

Jede anzugebende Eigenschaft wird im SLA-Dokument durch ein `ServiceLevelObjective`-Element gekapselt. Die Gültigkeitsdauer der Garantie wird innerhalb des `Validity`-Elements durch ein `End`-Element definiert, welches ein Datum im Format <Jahr>-<Monat>-<Tag> umschließt. Die eigentliche garantierte Eigenschaft und der Wert dieser Eigenschaft werden mit einem Vergleichsoperator verknüpft.

```
<SLA>
      <ServiceLevelObjective>
            ...
            <Validity>
                  ...
                  <End>2005-12-31</End>
            </Validity>
            <Expression>
                  <Predicate type="Greater">
                        <SLAParameter>Availability</SLAParameter>
                        <Value>98.5</Value>
                  </Predicate>
            </Expression>
            ...
      </ServiceLevelObjective>
      <ServiceLevelObjective>
            ...
      </ServiceLevelObjective>
      ...
</SLA>
```

Abbildung 45: Struktur eines SLA-Dokumentes

Ein mit WSLA modelliertes SLA-Dokument wird durch die SLA-Management-Komponente des WSProxy verwaltet. Nach der Registrierung eines Web Service erzeugt die SLA-Management-Komponente eine Instanz der Klasse `SLAParser` und liest das SLA-Dokument über die Funktion `readSlaDocument` ein. Hierzu wird über die Funktion `parse` ein Parsing des eingelesenen Dokumentes durchgeführt. Dafür wird zunächst eine DOM-Repräsentation des Dokumentes erstellt. Werden im DOM-Baum die Blätter erreicht, so wird der Inhalt des Elements sowie dessen Gültigkeit ermittelt. Das Parsing ist abgeschlossen, wenn auf diesem Wege die Verarbeitung wieder das SLA-Wurzelelement erreicht. Zu Beginn des Parsing wird der Klasse `SLA` das SLA-Wurzelelement übergeben. Sie wählt alle `ServiceLevelObjective`-Kindelemente aus und instanziiert für jedes dieser Kindelemente ein Objekt der Klasse `ServiceLevelObjective`. Dieses selektiert aus seinem `ServiceLevelObjective`-Element das `End`-Kindelement des `Validity`-Kindelements und übergibt es an eine neue Instanz der `End`-Klasse. Das `Predicate`-Kindelement des `Expression`-Kindelements übergibt es an eine neue Instanz der `Predicate`-Klasse. Die `End`-Klasse extrahiert das Datum, welches das `End`-Element umschließt und prüft dessen Gültigkeit. Die `Predicate`-Klasse instanziiert für ihre Kindelemente `SLAParameter` und `Value` Objekte

der Klassen `SLAParameter` und `Value`. Die Klasse `SLAParameter` ermittelt den Eigenschaftsbezeichner, der durch das `SLAParameter`-Element umschlossen wird. Die Klasse `Value` berechnet den Wert, den das `Value`-Element umschließt. Die Klasse `Predicate` speichert die Inhalte der Kindelemente `SLAParameter` und `Value` sowie den Wert des Attributs `type` des `Predicate`-Elements. Sind alle Kindelemente und deren Attributwert vorhanden und mit gültigen Werten belegt, wird auch das `Predicate`-Element für gültig erklärt. Die Klasse `ServiceLevelObjective` sammelt auf diese Weise ebenfalls die Inhalte der relevanten Subelemente `End` und `Predicate` und bestimmt seine Gültigkeit aus der Gültigkeit aller Subelemente. Die `SLA`-Klasse wiederum sammelt alle Objekte der Klasse `ServiceLevelObjective` in einem `Vector`-Objekt. Sind alle `ServiceLevelObjective`-Objekte im `Vector` gültig, wird das ganze SLA-Dokument für gültig erklärt und das Verfallsdatum des gesamten SLA-Dokumentes als frühestes Verfallsdatum unter allen `ServiceLevelObjective`-Objekten ermittelt.

Wurde ein unvollständiges oder nicht korrektes SLA-Dokument übermittelt, bekommt der Nutzer ausführliche Informationen darüber angezeigt, welche nicht-funktionalen Eigenschaften in seinem SLA-Dokument fehlen oder falsch spezifiziert wurden.

A.1.3 Status-Codes beim Logging

Die beim Logging verwendeten Status-Codes können Tabelle 16 entnommen werden.

Status-Code	Beschreibung	Ausgelöster HTTP-Fehler
`Received`	Eine Anfrage wurde vom `WSProxyClientThread` empfangen und in einem `WSProxyHTTPRequest`-Objekt abgelegt.	
`Routing`	Das `WSProxyHTTPRequest`-Objekt wurde an den `WSRouter` übergeben.	
`no route found`	Das Routing der Anfrage durch den `WSRouter` hat eine Ausnahme ausgelöst. Möglich Ursachen sind: • Die zu routende Anfrage war `NULL` • Zu dem angefragten WSID konnte kein USID gefunden werden • Zu der USID konnte kein Target gefunden werden • Das Target spezifiziert kein oder kein bekanntes Suchverfahren	`404 (file not found)`
`Searching endpoint url`	Das Target spezifiziert ein gültiges Suchverfahren und der Suchalgorithmus sucht momentan nach der Endpunkt-URL des Web Service.	

Status-Code	Beschreibung	Ausgelöster HTTP-Fehler
`remote call timeout`	Während des Web Service Aufrufs trat ein Timeout auf und die Kommunikation mit dem Web Service wurde abgebrochen.	`504 (gateway timeout)`
`remote call io error`	Beim Aufruf des Web Service wurde eine `IOException` ausgelöst. Eine mögliche Ursache hierfür ist, dass der Verbindungsaufbau zum Web Service gescheitert ist.	`502 (bad gateway)`
`remote call error`	Während des Web Service Aufrufs kam es zu einer Ausnahme, die weder auf einen Timeout noch auf einen IO-Fehler zurückzuführen ist. Die Ursache des Fehlers ist unbekannt.	`500 (internal server error)`
`Empty response`	Nach dem Aufruf des Web Service wurde eine leere Antwort vom Web Service empfangen.	`500 (internal server error)`
`sending response`	Der Aufruf des Web Service war erfolgreich und die empfangene (nicht leere) Antwort wird nun an den Client gesendet.	
`sending response failed`	Beim Senden der Antwort an den Client ist eine Ausnahme aufgetreten. Es ist unklar, ob die Antwort ganz, teilweise oder gar nicht übermittelt werden konnte.	
`Finished`	Die Antwort wurde fehlerfrei an den Client übermittelt. Die Verarbeitung der Anfrage durch den `WSProxyClientThread` wurde komplett und fehlerfrei abgeschlossen.	

Tabelle 16: Übersicht der beim Logging verwendeten Status-Codes

A.2 WSPortal

Abbildung 46 zeigt eine Eingabemaske, die es Anbietern ermöglicht, ihre Services zu registrieren. Hier haben Anbieter die Wahl, die nicht-funktionalen Eigenschaften ihrer Web Services in die Eingabemaske einzutragen oder das entsprechende SLA-Dokument über eine URL zu referenzieren.

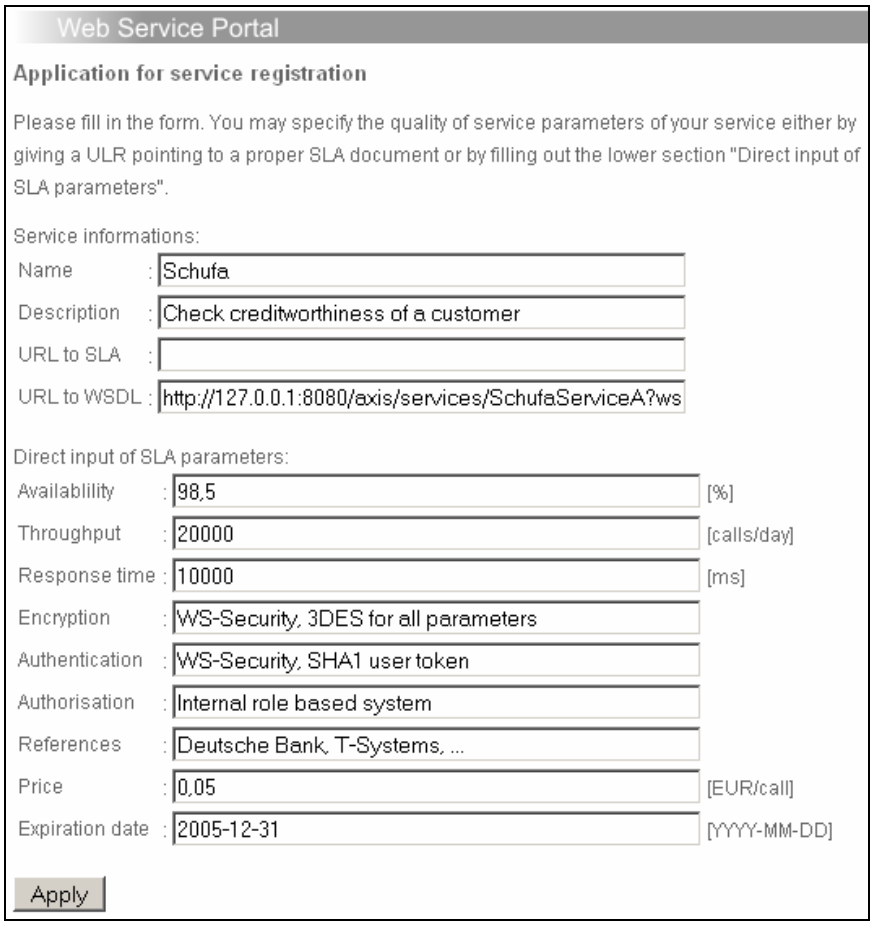


Abbildung 46: Registrierung von Web Services am WSPortal

A.3 WSProxyAdmin

WSProxyAdmin ist das Administrations-Frontend von WSQoSX. Funktionalitäten, die durch den WSProxyAdmin angeboten werden, umfassen:

- Konfiguration des Routings von Web Services durch WSProxy
- Bewertung von neu am WSPortal angemeldeten Web Services
- Verwaltung der Web Service Kategorien, wie z. B. die Gewichtung der nicht-funktionalen Eigenschaften und die Definition der Regeln zur Beschreibung von Mindestanforderungen an Web Services innerhalb der jeweiligen Kategorie
- Verwaltung der Web Services
- Verwaltung der am WSPortal registrierten Anbieter
- Einsicht der beim Aufruf von Web Services generierten Warnungen sowie der Call-History, der History-Statistik und der Logging-Tabelle

Die Komponenten von WSQoSX verwenden zur Speicherung der Daten den Datenbankserver MySQL 4.1.9. In jeder Kernkomponente lässt sich der Zugriff auf den Datenbankserver individuell konfigurieren, sodass es auf einfache Weise möglich ist, die einzelnen Kernkomponenten und den Datenbankserver auf mehrere unterschiedliche Systeme zu verteilen. In jeder

Komponente können folgende Angaben für den Zugriff auf die gemeinsame Datenbank individuell festgelegt werden:

- IP-Adresse und Port des Datenbankservers
- Zu verwendender Treiber für den Zugriff auf den Datenbankserver
- Benutzername und Passwort für die Anmeldung am Datenbankserver
- Name der Datenbank auf dem Datenbankserver
- Namen der Tabellen in der Datenbank

Eine Übersicht über die Tabellen der Datenbank kann Tabelle 17 entnommen werden.

Tabelle	**Kurzbeschreibung des Inhalts**
ws_mapping	Zuordnung eines WSID zu einem USID
ws_routing	Zuordnung eines USID zu einem Target
ws_portal_categories	Web Service Kategorien sowie die jeweiligen Gewichtung der nicht-funktionale Eigenschaften
ws_portal_categories_rules	Regeln zur Definition von Mindestanforderungen in den einzelnen Kategorien
ws_portal_services	Am WSPortal angemeldete Web Services sowie deren jeweilige Bewertung
ws_portal_services_parameters	Garantien bzgl. der nicht-funktionalen Eigenschaften eines Web Service, die bei der Anmeldung durch den Anbieter zugesichert werden.
ws_portal_services_points	Bewertung für die nicht-funktionalen Eigenschaften eines Web Service, die vom Administrator vergeben oder durch Normalisierung aus den garantieren Eigenschaften berechnet wird.
ws_logging	Logging-Tabelle mit detaillierten Informationen zu jedem durch den WSProxy verarbeiteten HTTP-Request
ws_history_calls	Daten der Call-History
ws_history_statistics	Daten der History-Statistik
ws_history_warnings	Warnungen, die bei einem Web Service Aufruf erzeugt werden.
ws_portal_user	Am WSPortal registrierte Anbieter

Tabelle 17: Kurzbeschreibung der zur Datenhaltung verwendeten Tabellen

B WorkflowOptimizer

Der WorkflowOptimizer hat die Aufgabe, die Auswahl der entsprechenden Web Services in Abhängigkeit von deren Dienstgüteeigenschaften zu simulieren. Hierzu werden alle Prozessschritte des Workflows nacheinander durchlaufen, wobei zu jedem Prozessschritt ein Web Service ausgewählt wird, der die im Prozessschritt geforderte Funktionalität bereitstellt. Hierbei protokolliert der WorkflowOptimizer das Laufzeitverhalten und die Lösungsgüte der simulierten Testfälle. Zur Entwicklung des WorkflowOptimizer wurde Java 2 Platform Standard Edition (J2SE) in der Version 5.0 verwendet.

Dieses Kapitel ist wie folgt gegliedert: In Abschnitt B.1 wird die Implementierung der Simulationsumgebung WorkflowOptimizer vorgestellt. Die Definition der Testfälle wird in Abschnitt B.2 beschrieben. In Abschnitt B.3 wird die Implementierung des in Abschnitt 4.3 entwickelten mathematischen Modells erläutert. Die Generierung der definierten Testfälle ist Gegenstand von Abschnitt B.4. Die Benutzungsoberfläche des WorkflowOptimizer wird in Abschnitt B.5 beschrieben. In Abschnitt B.6 werden die Statusinformationen diskutiert, die der WorkflowOptimizer während eines Simulationslaufes generiert.

B.1 Implementierung der Simulationsumgebung

Die Simulationsumgebung WorkflowOptimizer wird durch die Klasse `SequentialWorkflowEngine` implementiert. Nach der Initialisierung eines Testfalls im `TestSetProcessor`-Objekt wird eine Instanz der Klasse `SequentialWorkflowEngine` erzeugt und hierbei das `Data`-Objekt des Testfalls im Konstruktor übergeben. Hierdurch verfügt die Workflow-Engine über alle Daten des Testfalls. Durch den Aufruf der Methode `simulateExecution` wird die simulierte Ausführung des Workflows gestartet. Nachdem der Workflow durch das `SequentialWorkflowEngine`-Objekt ausgeführt wurde, können die den Prozessschritten zugeordneten Web Services dem Attribut `usedWebService` der jeweiligen `TaskItem`-Objekte des `Workflow`-Objekts entnommen werden. Die Klasse `ExecutionPlan` stellt darüber hinaus die statische Funktion `createExecutionPlan` bereit, die aus dem übergebenen `Workflow`-Objekt einen Ausführungsplan (`ExecutionPlan`-Objekt) erstellt. Dieser enthält die tatsächlich ausgewählten Web Services. Über die Funktion `quality` des `ExecutionPlan`-Objekts lässt sich anschließend der Zielfunktionswert des Ausführungsplans berechnen.

Im Folgenden wird eine Übersicht über die wichtigsten Pakete und Klassen der Simulationsumgebung gegeben.

Paket de.tud.kom.dynws.optimizer

Dieses Paket enthält die benötigten Klassen (siehe Tabelle 18) zur Ausführung des WorkflowOptimizer. Neben der ausführbaren Klasse zum Starten der Anwendung zählen hierzu:

- Klassen für die Nachrichtenverarbeitung und die Ausführung der Lösungsverfahren in eigenen Threads

- Klassen zum Kapseln der Definitionsdateien, aus denen sich ein Testfall zusammensetzt
- Klassen zum Kapseln des linearen Optimierungsproblems, das aus einem Testfall erzeugt wird
- Klassen zum Messen von Rechenzeiten und dem Speichern der Messergebnisse
- Klassen zur Berechnung statistischer Werte aus der Datenbasis eines Testfalls

Klasse	Beschreibung
`WorkflowOptimizerGUI`	`WorkflowOptimizer`-Anwendung mit grafischer Benutzungsoberfläche
`WorkflowOptimizerGUIMessageDispatcher`	Dispatcher zur Verarbeitung von Statusnachrichten in eigenem Thread
`WorkflowOptimizerGUICSVDispatcher`	Dispatcher zur Ausgabe von Messwerten in CSV-Dateien in eigenem Thread
`TestCase`	Optimierungsproblem in Form eines Testfalls
`TestSetProcessor`	Einlesen aller Testfälle und Weitergabe an `SequentialWorkflowEngine`
`TestSetProcessorLinear`	Einlesen aller Testfälle und Ermittlung der exakten Lösung durch den Solver
`MIPModel`	Gemischt-ganzzahliges lineares Optimierungsproblem eines Testfalls
`PerformanceCase`	Messwerte bezüglich Rechenzeit, Iterationen und Zielfunktionswert eines vollständigen Testfalls
`PerformanceStep`	Messwerte bezüglich Rechenzeit, Iterationen und Zielfunktionswert der Heuristik in einem Abarbeitungsschritt der `SequentialWorkflowEngine`
`Statistics`	Berechnung statistischer Informationen zu den Daten eines Testfalls
`Timer`	Durchführung von Messungen der Rechenzeit

Tabelle 18: Klassenübersicht de.tud.kom.dynws.optimizer

Paket de.tud.kom.dynws.optimizer.data

Dieses Paket enthält Klassen zur Kapselung der Daten, die ein Optimierungsproblem definieren, wie z. B. Workflows, Prozessschritte, Kategorien, Web Services, Eigenschaften von Web Services und Parameterwerte (siehe Tabelle 19).

Klasse	Beschreibung
`Workflow`	Beschreibt einen `Workflow` bestehend aus Prozessschritten (`TaskItems`). Zusätzlich wird das Optimierungsproblem durch Angabe von Restriktionen und Gewichten für die Zielfunktion (als `ParameterSet`-Objekte) definiert.
`TaskItem`	Prozessschritt im Workflow. Die geforderte Funktionalität wird durch die zugehörige Kategorie (`Category`) spezifiziert.
`Categories`	Container für alle Kategorien (`Category`-Objekte)
`Category`	Kategorie als Container für alle Web Services (`WebService`-Objekte) gleicher Funktionalität
`WebService`	Beschreibt die Eigenschaften (`ParameterSet`) des Web Service
`ParameterDefinitions`	Container für alle Parameterdefinitionen (`ParameterDefinition`-Objekte)
`ParameterDefinition`	Parameterdefinition (Name, Bezeichnung der Einheit, Art des Parameters [additiv | multiplikativ | Minimaloperator], Verbesserung durch Zu- oder Abnahme des Parameterwerts)
`Parameter`	Eigenschaft eines Web Service, ausgedrückt als Parameterwert gemäß einer Parameterdefinition (`ParameterDefinition`)
`ParameterSet`	Menge von Eigenschaften bestehend aus einem Parameterwert (`Parameter`) zu jeder im Container `ParameterDefinitions` enthaltenen Eigenschaftsart (`ParameterDefinition`)
`Data`	Container für Daten eines Testfalls. Dieser enthält: • Definition des Workflows • Definition des Optimierungsproblems (`Workflow`-Objekt) • Definition des linearen Optimierungsproblems (`MIPModel`-Objekt) • Definition der Kategorien (`Categories`-Objekt) • Parameterdefinitionen (`ParameterDefinitions`-Objekt)

Tabelle 19: Klassenübersicht de.tud.kom.dynws.optimizer.data

Paket de.tud.kom.dynws.optimizer.sequentialheuristics

Dieses Paket enthält Klassen zur Optimierung eines Ausführungsplans (siehe Tabelle 20).

Klasse	Beschreibung
`ExecutionPlan`	Ausführungsplan zu einem Workflow (`Workflow`) bestehend aus Prozessschritten (`ExecutionPlanItems`). Der `ExecutionPlan` enthält die auszuführenden Web Services und kapselt eine Lösung des Optimierungsproblems.

Klasse	Beschreibung
`ExecutionPlanItem`	Ein Ausführungsschritt im Ausführungsplan (`ExecutionPlan`), der einen auszuführenden Web Service (`WebService`) definiert.
`H1RelaxIP`	Implementierung des Eröffnungsverfahrens H1_RELAX_IP
`H2SWAP`	Implementierung des Verbesserungsverfahrens H2_SWAP
`H3SimulAnneal`	Implementierung des Verbesserungsverfahrens H3_SIMUL_ANNEAL
`SequentialWorkflowEngine`	Simuliert eine Workflow-Engine, die zu einem Workflow (`Workflow`) mittels der Heuristik einen optimierten Ausführungsplan (`ExecutionPlan`) erstellt und diesen schrittweise abarbeitet.
`SolveSingle`	Findet die optimale Besetzung eines Ausführungsplans (`ExecutionPlan`), der nur noch einen einzigen Prozessschritt (`ExecutionPlanItem`) umfasst.

Tabelle 20: Klassenübersicht de.tud.kom.dynws.optimizer.sequentialheuristics

Paket de.tud.kom.dynws.optimizer.datagenerator

Dieses Paket enthält Klassen zur Realisierung eines Datengenerators, der aus den Angaben einer Parameter- und einer Generatordefinitionsdatei entsprechende Testfälle erzeugt (siehe Tabelle 21).

Klasse	Beschreibung
`DataGeneratorStatistical`	Datengenerator zum Erzeugen von Testfällen gemäß den Angaben in der Parameter- und Generatordefinitionsdatei
`ParameterGenerationDefinition`	Kapselung der Definition eines einzelnen Parameters sowie der entsprechenden Parameterwerte

Tabelle 21: Klassenübersicht de.tud.kom.dynws.optimizer.datagenerator

B.2 Definition der Testfälle

Ein Testfall wird durch die drei Dateien `Workflow_<Nummer>.in`, `ParameterDefinitions_<Nummer>.in` und `Categories_<Nummer>.in` definiert, wobei `<Nummer>` ein Platzhalter für eine Zahl darstellt. Dateien, die gemeinsam einen Testfall definieren, tragen dieselbe Nummer. Bei den einzelnen Dateien handelt es sich um Textdateien, die jeweils einen spezifischen Teil der Definition eines Optimierungsproblems abbilden. Ihr Aufbau wird im Folgenden erläutert.

Parameterdefinitionsdatei

Die Parameterdefinitionsdatei `ParameterDefinitions_<Nummer>.in` definiert die einzelnen Parameter, durch die ein Web Service im Optimierungsproblem beschrieben wird. Abbildung 47 zeigt das Beispiel einer Parameterdefinitionsdatei, die drei Parameter enthält.

```
Response time;ms;<;+
Availability;%;>;*
Throughput;calls/min;>;<
```

Abbildung 47: Beispiel einer Parameterdefinitionsdatei

Jede Zeile entspricht der Definition eines Parameters. Jeder Parameter wird durch vier Angaben spezifiziert:

- Name des Parameters
- Name der Maßeinheit
- Ein Zeichen, welches definiert, ob größere („>“) oder kleinere („<“) Parameterwerte einen höheren Nutzen stiften.
- Ein Zeichen, welches definiert, wie einzelne Parameterwerte des Parameters zu einem Gesamtparameterwert eines Ausführungsplans aggregiert werden. Additive Parameter werden durch „+“, multiplikative Parameter durch „*“ und Minimaloperator-Parameter durch „<“ definiert.

In einer Parameterdefinitionsdatei muss mindestens ein Parameter definiert werden. Eine Beschränkung der Anzahl *K* der definierbaren Parameter nach oben besteht nicht.

Workflowdefinitionsdatei

Die Workflowdefinitionsdatei `Workflow_<Nummer>.in` beschreibt einen sequenziellen Workflow, die auf den Gesamtparameterwerten anzuwendenden Restriktionen in Form der Schrankenwerte sowie die Gewichte zur Umrechnung der Gesamtparameterwerte in Nutzeneinheiten. Abbildung 48 zeigt das Beispiel einer Workflowdefinitionsdatei.

```
W;0,03125;5000;0,1
c;3000;-;-
1/1→2/2→3/3
```

Abbildung 48: Beispiel einer Workflowdefinitionsdatei

Eine Workflowdefinitionsdatei umfasst stets drei Zeilen. Die erste Zeile beginnt mit dem Zeichen *w* und definiert die Gewichte in der Reihenfolge, in der die zugehörigen Parameter in der Parameterdefinitionsdatei festgelegt wurden. Die zweite Zeile beginnt mit dem Zeichen *c* und definiert die Schrankenwerte, welche die jeweiligen Gesamtparameterwerte beschränken. Durch die dritte Zeile wird der eigentliche Workflow als Abfolge von Prozessschritten definiert, deren jeweilige Funktionalität mittels einer Kategorie spezifiziert wird. Ein Prozessschritt wird durch eine Prozessschrittnummer und eine Kategorienummer definiert, die durch das Zeichen „/“ separiert werden. Die Prozessschrittnummer dient lediglich der eindeutigen Bezeichnung eines Prozessschrittes, wohingegen die Kategorienummer die Nummer der Ka-

tegorie spezifiziert, durch die alle Web Services gruppiert werden, welche die benötigte Funktionalität des Prozessschrittes erbringen können. Wird die Funktionalität einer Kategorie mehrmals in einem Workflow benötigt, können mehrere Prozessschritte mit derselben Kategorienummer definiert werden. Ist ein Prozessschritt der Nachfolger eines anderen Prozessschrittes, wird er durch das Zeichen „→“ von diesem getrennt und direkt hinter diesem angegeben.

Kategoriedefinitionsdatei

In der Kategoriedefinitionsdatei `Categories_<Nummer>.in` wird festgelegt, welche Web Services mit welchen Eigenschaften in den einzelnen Kategorien zur Bildung eines entsprechenden Ausführungsplans verfügbar sind.

Die Definition einer Kategorie beginnt stets mit einer Zeile, welche die Nummer der Kategorie beschreibt. Anschließend folgt eine Zeile, die mit dem Zeichen `c` beginnt und Schrankenwerte für die Parameterwerte der Web Services in dieser Kategorie festlegt. Diese Zeile ist analog zur Definition der Schrankenwerte für Gesamtparameterwerte in der Workflowdefinitionsdatei aufgebaut, jedoch werden durch diese Schrankenwerte lediglich die zulässigen Web Services innerhalb einer Kategorie definiert. Danach werden Web Services für diese Kategorie spezifiziert. Die Zeile beginnt jeweils mit der Nummer des Web Service, die ihn innerhalb dieser Kategorie eindeutig identifiziert. Anschließend folgen durch ein Semikolon getrennt die Parameterwerte des Web Service in der Reihenfolge der Definition der zugehörigen Parameter in der Parameterdefinitionsdatei. Die Definition einer Kategorie wird durch eine Zeile abgeschlossen, die das Zeichen „-“ enthält. Das Beispiel in Abbildung 49 zeigt die Definition der drei Kategorien `1`, `2` und `3`, die jeweils die Web Services `1`, `2` und `3` enthalten. In keiner der Kategorien erfolgt eine Einschränkung der zulässigen Web Services.

```
1
c;-;-;-
1;1000;0,995;2500
2;1500;0,99;5000
3;500;0,999;1000
-
2
c;-;-;-
1;250;0,999;1250
2;500;0,996;1500
3;750;0,991;2000
-
3
c;-;-;-
1;3000;0,9995;4500
2;2000;0,9999;3000
3;4000;0,999;6000
-
```

Abbildung 49: Beispiel einer Kategoriedefinitionsdatei

Einlesen von Testfällen

Beim Start des Optimierungsprozesses werden die zu simulierenden Testfälle durch ein Objekt der Klasse `TestSetProcessor` eingelesen. Nach dem Start der Simulation wird eine neue Instanz der Klasse `TestSetProcessor` erzeugt, die über ihre Methode `run` das Interface `Runnable` implementiert. Anschließend wird eine neue Instanz der Klasse `Thread` erzeugt, wobei das `TestSetProcessor`-Objekt als im Thread auszuführende `Runnable`-Instanz übergeben wird. Die Verarbeitung der Testfälle geschieht nun in einem neuen Thread, unabhängig vom Thread zur Ausführung der grafischen Benutzungsoberfläche der WorkflowOptimizer-Anwendung. Die Testfälle werden über die Methode `loadTestCase` des `TestSetProcessor`-Objekts eingelesen. Für jeden Testfall wird ein `TestCase`-Objekt erzeugt, um über dessen Funktion `load` alle drei Definitionsdateien des Testfalls einzulesen. Liefert diese Funktion als Rückgabewert `false`, so ist das Einlesen mindestens einer zu diesem Testfall gehörenden Definitionsdatei fehlgeschlagen und der Testfall wird nicht in die Liste der abzuarbeitenden Testfälle übernommen.

Die Funktion `load` des `TestCase`-Objekts (siehe Abbildung 50) liest die einzelnen Definitionsdateien eines Testfalls nicht selbst ein, sondern erzeugt zum Einlesen und Speichern aller Definitionsdaten ein `Data`-Objekt und delegiert das Einlesen der Testfälle an dessen `load`-Funktion. Parameterdefinitionen werden in einem `ParameterDefinitions`-Objekt abge-

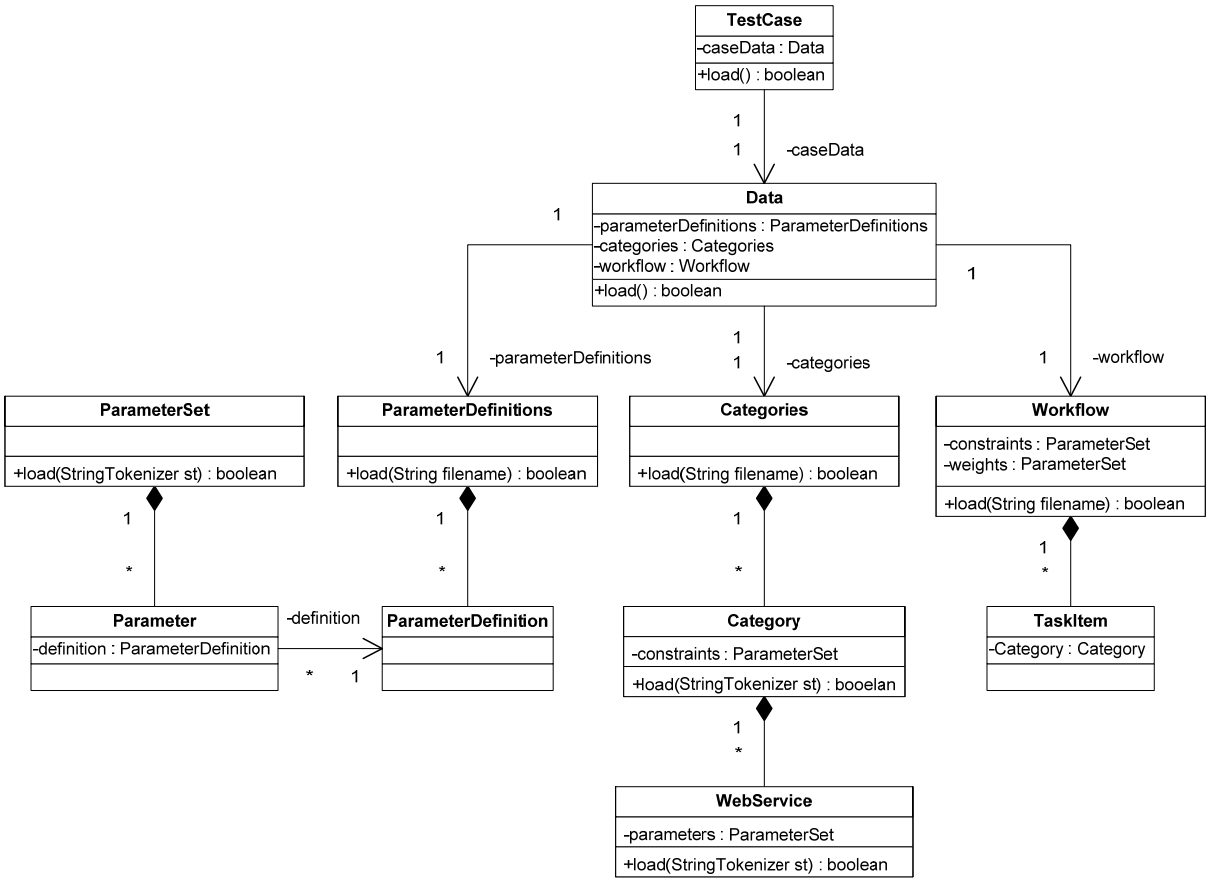


Abbildung 50: Datenstruktur eines Testfalls

legt, Kategoriedefinitionen in einem `Categories`-Objekt und die Workflowdefinition in einem `Workflow`-Objekt.

Das `ParameterDefinitions`-Objekt erzeugt beim Einlesen für jede Parameterdefinition ein `ParameterDefinition`-Objekt, das es in einem `Vector`-Attribut speichert und durch öffentliche Funktionen gekapselt zugreifbar macht. Analog legt das `Categories`-Objekt für jede Kategorie der Kategoriedefinitionsdatei ein `Category`-Objekt an. Hierbei wird das Einlesen der Kategoriedaten weiter an die `load`-Funktion des `Category`-Objekts delegiert. Die Restriktionen einer Kategorie werden durch ein `ParameterSet`-Objekt repräsentiert. Ein `ParameterSet`-Objekt erzeugt hierbei für jeden eingelesenen Parameter ein `Parameter`-Objekt, das den Parameterwert kapselt und die Art des Parameters durch einen Verweis auf das entsprechende `ParameterDefinition`-Objekt festlegt. Weiterhin legt das `Category`-Objekt für jeden Web Service der Kategorie ein `WebService`-Objekt an. Zum Einlesen und Speichern der Parameterwerte eines Web Service verwendet das `WebService`-Objekt wiederum ein `ParameterSet`-Objekt.

Beim Einlesen der Workflowdefinitionsdatei liest das `Workflow`-Objekt zunächst die Gewichte und die Schrankenwerte bezüglich der Restriktionen auf den Gesamtparameterwerten ein. Hierbei wird in beiden Fällen ein `ParameterSet`-Objekt zum Einlesen und Ablegen der Gewichte bzw. Schrankenwerte verwendet. Anschließend wird die Zeile zur Definition des Workflows in einzelne Prozessschritte zerlegt und für jeden Prozessschritt ein `TaskItem`-Objekt erzeugt, wobei die im Prozessschritt benötigte Funktionalität durch einen Verweis auf das entsprechende `Category`-Objekt festgelegt wird.

B.3 Erstellen des mathematischen Modells

Zur Vorbereitung auf die Optimierung durch das heuristische Lösungsverfahren werden zunächst die Kategorien durchlaufen und alle Web Services aus den Kategorien entfernt, welche den jeweiligen Restriktionen der Kategorie nicht entsprechen. Hierdurch befinden sich in den Kategorien nur noch jene Web Services, die zur Bildung eines Ausführungsplans tatsächlich verwendet werden können. Anschließend wird aus dem Testfall ein gemischt-ganzzahliges lineares Optimierungsproblem in einer Form erzeugt, sodass es durch den Solver lp_solve verarbeitet werden kann.

Das GLOP wird durch ein `MIPModel`-Objekt gekapselt, das in einem Attribut des `Data`-Objekts des Testfalls abgelegt wird. Die Transformation des Testfalls in ein GLOP wird durch die Methode `create` des `MIPModel`-Objekts vorgenommen. Das Optimierungsproblem wird hierbei wie in Abschnitt 4.3 beschrieben aufgebaut. Das API des Solvers unterstützt zur Definition des Optimierungsproblems die Spezifikation einer Zielfunktion in der Form $F(\vec{x}) = \vec{w}^T \cdot \vec{x}$ sowie die Spezifikation von Nebenbedingungen in der Form $\vec{a}^T \cdot \vec{x} \leq \vec{c}$, $\vec{a}^T \cdot \vec{x} = \vec{c}$ und $\vec{a}^T \cdot \vec{x} \geq \vec{c}$.

Bevor die Nebenbedingungen und die Zielfunktion des Optimierungsproblems festgelegt werden können, ist es zunächst nötig, den Vektor $\vec{x}$ der Entscheidungsvariablen festzulegen

und den Wertebereich jeder Variablen darin zu definieren. Dazu werden alle Prozessschritte (`TaskItem`-Objekte) des Workflows (`Workflow`-Objekt) durchlaufen und zu jedem Prozessschritt über die ihm zugeordnete Kategorie (`Category`-Objekt) alle möglichen Web Services (`WebService`-Objekte) zur Realisierung der im Prozessschritt benötigten Funktionalität ermittelt. Für alle m_i Web Services zur Realisierung des Prozessschrittes i wird eine binäre Entscheidungsvariable $x_{i,j}$ eingeführt.

Im Anschluss daran wird zur Abbildung der Gesamtparameterwerte für jeden der definierten Parameter eine reellwertige Variable eingeführt. Nachdem die Entscheidungsvariablen des Lösungsvektors $\vec{x}$ im Solver deklariert wurden, kann eine matrixorientierte Abbildung des Optimierungsproblems durchgeführt werden. Die beim Aufbau des mathematischen Modells benötigten Informationen werden der Datenstruktur des `TestCase`-Objekts entnommen. Im Folgenden wird die Herkunft der einzelnen Informationen kurz erläutert.

$p^+, p^\bullet, p^{\min}$ Die Definitionen der Parameter ist im `ParameterDefinitions`-Objekt abgelegt. Aus dieser Definition ergibt sich, ob es sich bei einem Parameter um einen additiven, multiplikativen oder einen Minimaloperator-Parameter handelt und ob größere oder kleinere Parameterwerte einen höheren Nutzen stiften.

$c^+, c^\bullet, c^{\min}$ Die Schrankenwerte eines Gesamtparameterwertes können als `Parameter`-Objekte dem `ParameterSet`-Objekt im Attribut `constraints` des `Workflow`-Objekts entnommen werden.

$p_{i,j}^+, p_{i,j}^\bullet, p_{i,j}^{\min}$ Die Parameterwerte können als `Parameter`-Objekte dem `ParameterSet`-Objekt im Attribut `parameters` des `WebService`-Objekts entnommen werden. Die Art des Parameters wird hierbei über das `ParameterDefinition`-Objekt bestimmt, das dem `Parameter`-Objekt im Attribut `definition` zugeordnet ist.

$w^+, w^\bullet, w^{\min}$ Das Gewicht, mit dem der Gesamtparameterwert in der Zielfunktion multipliziert wird, um den Gesamtparameterwert in Nutzeneinheiten zu transformieren, kann als `Parameter`-Objekt aus dem `ParameterSet`-Objekt im Attribut `weights` des `Workflow`-Objekts entnommen werden. Über das dem `Parameter`-Objekt zugeordneten `ParameterDefinition`-Objekt wird der Parameter bestimmt, der in der Zielfunktion zu gewichten ist. Ob das Gewicht positiv oder negativ in die Zielfunktion eingeht, wird über die entsprechende Angabe im `ParameterDefinition`-Objekt festgelegt.

B.4 Generierung der Testfälle

Der WorkflowOptimizer verfügt über einen Datengenerator zur automatischen Erstellung von Testfällen. Zur Datengenerierung werden die Parameter- und die Generatordefinitionsdatei benötigt. Der Aufbau der Parameterdefinitionsdatei wurde in Abschnitt B.2 erläutert. Nach

dem Einlesen der Parameterdefinitionsdatei sind alle Parameter in Form von `ParameterDefinition`-Objekten in einem `ParameterDefinitions`-Objekt abgelegt. Die Generatordefinitionsdatei entspricht in ihrem Aufbau einer Java-Properties-Datei und wird über die Klasse java.util.Properties eingelesen. Über die Funktion `getProperty` des zum Einlesen verwendeten `java.util.Properties`-Objekts kann über den Namen einer Eigenschaft auf deren Wert zugegriffen werden.

In der Generatordefinitionsdatei werden zuerst jene Eigenschaften definiert, welche die Größe und Anzahl der zu erzeugenden Testfälle festlegen. Die hierbei zur Definition verwendeten Eigenschaften können Tabelle 22 entnommen werden.

Eigenschaftsname	**Beschreibung**
`generator.cases.number`	Anzahl der zu erzeugenden Testfälle
`generator.cases.categories`	Anzahl der Prozessschritte des zu erzeugenden Workflows
`generator.cases.webservices`	Anzahl alternativer Web Services, die jeweils für die Ausführung eines Prozessschrittes zur Verfügung stehen

Tabelle 22: Eigenschaften zur Definition von Testfällen

Um Parameterwerte zur Beschreibung der Web Services in den Kategorien erzeugen zu können, muss im weiteren Teil der Generatordefinitionsdatei für jeden einzelnen in der Parameterdefinitionsdatei definierten Parameter festgelegt werden, wie die Parameterwerte erzeugt werden sollen. Zur Erzeugung eines Parameterwertes werden ein Intervall und ein Bezugswert *r* festgelegt, um den die erzeugten Parameterwerte gleichverteilt im definierten Intervall variieren. Dieser Wert *r* ist entweder ein fixer Wert oder aber mit einem bereits zuvor erzeugten Parameterwert korreliert.

Alle Eigenschaften zur Definitionen von Parametern haben einen Namen der mit `generator.param.<ID>` beginnt. `<ID>` steht hierbei für die Identifikationsnummer des Parameters, dessen Erzeugung durch die Eigenschaft beeinflusst werden soll. Die Variation um einen Wert wird für jeden Parameter durch die beiden Eigenschaften `generator.param.<ID>.variation.down` und `generator.param.<ID>.variation.up` festgelegt. Die Eigenschaftswerte bezeichnen hierbei die maximale prozentuale Schwankung um den Bezugswert *r* nach unten bzw. oben. Ein Parameter, der um einen festen Wert *r* variieren soll, wird durch die Belegung der Eigenschaft `generator.param.<ID>.type` mit dem Wert `fixed` definiert. Der Wert *r* wird über die Eigenschaft `generator.param.<ID>.value` festgelegt (siehe Tabelle 23).

generator.param.<ID>.type = [fixed|correlated]		
`fixed`	`value =` r	
`correlated`	Definiert einen Wert r in Abhängigkeit eines anderen Parameters.	
	`correlation.param= <id>`	Der Wert r ergibt sich aus Wert p des Parameters mit Index `<id>` in der Gestalt $r = ap + b$.
	`correlation.factor = a`	
	`correlation.fixed = b`	
generator.param.<ID>.variation=[up|down]		
`variation`	`up =` $\underline{e}$	Definiert die prozentuale Abweichung $[\underline{e}, \bar{e}]$.
	`down =` $\bar{e}$	

Tabelle 23: Eigenschaften zur Definition von Parametern

Um einen Parameter zu definieren, der mit einem anderen korreliert, wird der Bezugswert r in Abhängigkeit von dem Parameterwert p eines anderen Parameters ausgedrückt. Die Abhängigkeit wird durch eine lineare Funktion $r = ap + b$ festgelegt. Diese Art der Parametergeneration wird durch die Belegung der Eigenschaft `generator.param.<ID>.type` mit dem Wert `correlated` definiert. Welcher andere Parameter als Variable p der linearen Gleichung verwendet werden soll, wird durch die Eigenschaft `generator.param.<ID>.correlation.param` bestimmt, die auf die Identifikationsnummer des entsprechenden Parameters gesetzt wird. Die Steigung wird durch den Wert der Eigenschaft `generator.param.<ID>.correlation.factor` und der Achsenabschnitt durch den Wert der Eigenschaft `generator.param.<ID>.correlation.fixed` festgelegt.

Die Generatordefinitionen der einzelnen Parameter werden aus einem `java.util.Properties`-Objekt extrahiert und die Generatordefinition jedes einzelnen Parameters in einem `ParameterGenerationDefinition`-Objekt abgelegt. Die Klasse `ParameterGenerationDefinition` kapselt die Daten der Generatordefinition eines Parameters.

Im Beispiel aus Abbildung 51 wird der Parameter mit der Identifikationsnummer 1 definiert, der um den fixen Wert 1.000 variiert. Die maximale Variation nach oben beträgt 100% und die maximale Schwankung nach unten 60%. Hieraus ergibt sich für diesen Parameter der Wertebereich [400; 2.000], aus dem der Parameterwert als Ausprägung einer gleichverteilten Zufallsvariable bestimmt wird. Der zu generierende Testfall beschreibt einen Workflow, der aus 80 Prozessschritten besteht. Für jeden Prozessschritt wird eine eigene Kategorie erstellt und jeweils mit 40 Web Services zur Ausführung dieses Prozessschrittes gefüllt. In diesem Beispiel wird der Parameter mit der Identifikationsnummer 2 definiert, der um einen Wert variiert, der mit dem Parameterwert des ersten Parameters korreliert ist. Nach der Erzeugung eines Parameterwertes p_1 für den ersten Parameter, wird der Bezugswert r_2 des zweiten Parameters durch die Funktion $r_2 = -0{,}000005\, p_1 + 0{,}015$ berechnet. Anschließend ergibt sich der Parameterwert p_2 durch eine Variation des Referenzwertes im Bereich von maximal 50%

nach unten und maximal 50% nach oben. Die gewählte Abweichung ergibt sich als Ausprägung einer gleichverteilten, reellwertigen Zufallsvariable aus dem definierten Intervall. Da p_1 aus dem Wertebereich [400; 2.000] stammt und sich der Bezugswert r_2 gemäß $r_2 = -0{,}000005\,p_1 + 0{,}015$ berechnet, besitzt r_2 einen Wertebereich von [0,005; 0,013]. Durch das definierte Intervall ergibt sich daraus für den Parameterwert des zweiten Parameters ein Wertebereich von [0,0025; 0,0195].

```
# Number of test cases to generate
generator.cases.number = 1

# Number of categories (and number of workflow steps)
# to generate per test case
generator.cases.categories = 80

# Number of web services per category
generator.cases.webservices = 40

# Definition of parameter 1 generation
generator.param.1.type = fixed
generator.param.1.value = 1000
generator.param.1.variation.down = 0.6
generator.param.1.variation.up = 1

# Definition of parameter 2 generation
generator.param.2.type = correlated
generator.param.2.correlation.param = 1
generator.param.2.correlation.fixed = 0.015
generator.param.2.correlation.factor = -0.000005
generator.param.2.variation.down = 0.5
generator.param.2.variation.up = 0.5

# Definition of parameter 3 generation
generator.param.3.type = fixed
generator.param.3.value = 0.999
generator.param.3.variation.down = 0.009
generator.param.3.variation.up = 0.001

# Definition of parameter 4 generation
generator.param.4.type = fixed
generator.param.4.value = 100000
generator.param.4.variation.down = 0.8
generator.param.4.variation.up = 1
```

Abbildung 51: Beispiel einer Generatordefinitionsdatei

B.5 Benutzungsoberfläche

Zur Konfigurierung der Simulationsumgebung WorkflowOptimizer wurde eine Benutzungsoberfläche entwickelt (siehe Abbildung 52).

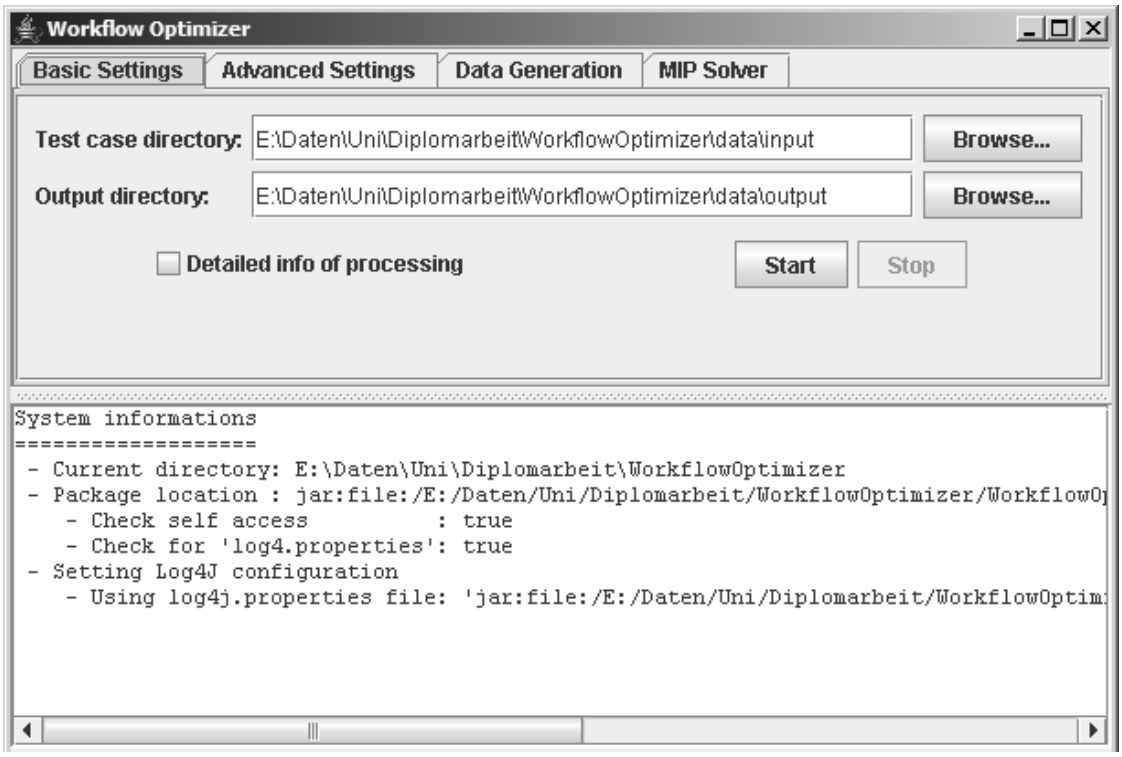


Abbildung 52: Benutzungsoberfläche des WorkflowOptimizer

So können durch entsprechende Kontrollelemente Programmparameter verändert und die verschiedenen Simulationsverfahren gestartet werden. Auch die Generierung der Testfälle kann über die Benutzungsoberfläche gesteuert werden. Um den Benutzer während der Ausführung von Aktionen über den aktuellen Zustand und eventuelle Zwischenergebnisse zu informieren, werden Protokollinformationen im Statusbereich der Benutzungsoberfläche und in einer Protokolldatei ausgegeben.

B.6 Statusinformationen

Im Folgenden werden die Statusinformationen erläutert, die der WorkflowOptimizer während eines Simulationsdurchlaufes erstellt. Zu Beginn der Verarbeitung eines Testfalls durch das `TestSetProcessor`-Objekt wird die Nummer des verarbeiteten Testfalls ausgegeben sowie die benötigte Zeit zur Initialisierung des Testfalls (siehe Abschnitt 1 in Abbildung 53). Nachdem ein `SequentialWorkflowEngine`-Objekt erstellt und die Simulation des Workflows gestartet wurde, werden zu jedem durchlaufenen Schritt Informationen über die erzeugten Teilergebnisse ausgegeben. Zunächst wird angezeigt, für welchen Prozessschritt des Workflows ein Web Service gefunden werden muss (siehe Abschnitt 2 in Abbildung 53). Im Beispiel wird die Abarbeitung des ersten Schrittes dargestellt. Der erste Prozessschritt muss durch einen Web Service aus der Kategorie 1 realisiert werden (`item (1/1)`). In Angaben der Form `(1/1)` entspricht die erste Zahl der Identifikationsnummer des Prozessschrittes und die zweite Zahl nach dem Schrägstrich der Identifikationsnummer der Kategorie, aus der ein Web Service zur Realisierung des Prozessschrittes zu entnehmen ist. Für welchen Teil des Workflows in diesem Schritt ein Ausführungsplan erstellt wird, ist unter dem Abschnitt *Now*

to solve angegeben. Im Beispiel muss in diesem Schritt ein Ausführungsplan für den Workflow `(1/1)→(2/2)→(3/3)` erzeugt werden.

```
====================================================================          1
== TEST CASE  1                                                    ==
====================================================================
 - Time for preparing data of test case: 67.35043 ms.
[...]

***********************************                                           2
* Workflow step   1, item (  1/  1) *
***********************************
 - Now to solve:
    - ExecutionPlan:
       - (  1/  1)  →  (  2/  2)  →  (  3/  3)
[...]

- Time for calculating initial valid solution: 28.272476 ms.                 3
 => Initial valid solution found by H1_RELAX_IP:
    - ExecutionPlan:
       - (  1/  1)  →  (  2/  2)  →  (  3/  3)
       -   3 (  2)        1 (  0)        2 (  1)
       [...]
       - Overall parameters:   0. Response time:    2750,00000
                               1. Availability:        0,99790
                               2. Throughput:       1000,00000
       - Quality: 5003.5684
[...]
- Time for processing this step: 33.507706 ms.                              4
```

Abbildung 53: Protokoll der simulierten Ausführung eines Workflows

Nach Ausführung der Heuristik H1_RELAX_IP (vgl. Abschnitt 4.4.2) wird der erste zulässige Ausführungsplan für den aktuellen Workflowteil angezeigt (siehe Abschnitt 3 in Abbildung 53). Im Beispiel benötigt H1_RELAX_IP 28,272476 ms. Der konkrete Ausführungsplan wird durch die Angabe der Web Services dargestellt, welche für die jeweiligen Prozessschritte des Workflows verwendet werden. Die Angabe `3(2)` bedeutet, dass im erzeugten Ausführungsplan für den ersten Prozessschritt der Web Service mit der Identifikationsnummer 3 und der Indexnummer 2 vorgesehen ist. Die Identifikationsnummer entspricht der Nummer, unter welcher der Web Service in der Kategoriedefinitionsdatei definiert wurde. Die Indexnummer entspricht dem Index des Web Service in der zugehörigen Kategorie. Unter `Overall parameters` werden die Gesamtparameterwerte des erzeugten Ausführungsplans angezeigt. Die sich aus der gewichteten Summe der Gesamtparameterwerte ergebende Zielfunktionswert bzw. Gesamtnutzen des Ausführungsplans wird unter `Quality` aufgeführt (Gesamtnutzen von 5003,5684 Einheiten, siehe Abschnitt 3 in Abbildung 53). Zum Abschluss eines Schrittes wird die benötigte Rechenzeit ausgegeben (siehe Abschnitt 4 in Abbildung 53). Nachdem alle Schritte abgearbeitet wurden, wird abschließend der tatsächlich ausgeführte Ausführungsplan, seine Gesamtparameterwerte, sein Zielfunktionswert und die insgesamt zur Ausführung des Workflows benötigte Zeit in einem Abschnitt `Whole workflow processed` ausgegeben.

C Publikationen des Verfassers

C.1 Wissenschaftliche Veröffentlichungen

Berbner, R.; Spahn, M.; Repp, N.; Heckmann, O.; Steinmetz, R.: *Dynamic Replanning of Web Service Workflows*. In: IEEE International Conference on Digital Ecosystems and Technologies 2007 (DEST 2007). Cairns, Australien, 2007.

Berbner, R.; Grollius, T.; Repp, N., et al.: *Management of Service-oriented Architecture (SoA) based Application Systems*. In: Enterprise Modelling and Information Systems Architectures - An International Journal, 2 (2007) 1, S. 14-25.

Berbner, R.; Spahn, M.; Repp, N.; Heckmann, O.; Steinmetz, R.: *Heuristics for QoS-aware Web Service Composition*. In: 4th IEEE International Conference on Web Services (ICWS 2006). Chicago, IL, USA, 2006, S. 72-82.

Berbner, R.; Spahn, M.; Repp, N.; Heckmann, O.; Steinmetz, R.: *An Approach for Replanning of Web Service Workflows*. In: 12th Americas Conference on Information Systems (AMCIS 2006). Acapulco, Mexiko, 2006.

Berbner, R.; Grollius, T.; Repp, N., et al.: *An approach for the Management of Service-oriented Architecture (SoA) based Application Systems.* In: Enterprise Modelling and Information Systems Architectures (EMISA 2005), Klagenfurt, Österreich, 2005, S. 208-221.

Berbner, R.; Heckmann, O.; Mauthe A.; Steinmetz, R.: *Eine Dienstgüte unterstützende Web Service-Architektur für flexible Geschäftsprozesse*. In: Wirtschaftsinformatik, 47 (2005) 4, S. 268-277.

Berbner, R.; Heckmann, O.; Steinmetz, R.: *An Architecture for a QoS driven composition of Web Service based Workflows*. In: Networking and Electronic Commerce Research Conference (NAEC 2005), Riva Del Garda, Italien, 2005.

Berbner, R.; Mauthe, A.; Steinmetz, R.: *Unterstützung dynamischer E-Finance-Geschäftsprozesse*. In: Elektronische Geschäftsprozesse (EGP 2004), Klagenfurt, Österreich, 2004, S. 44-54.

C.2 Patentanmeldungen

Berbner, R.; Grimminger, J.; Zaid, F.: Method and System for Execution of a Process. Eingereicht beim Europäischen Patentamt. 06017662.5.

Grimminger, J.; Berbner, R.; Zaid, F.: Method and Apparatus for performing a Business Process of a Service Provider. Eingereicht beim Europäischen Patentamt. 06018270.6.

C.3 Mitautorenschaft bei Veröffentlichungen

Repp, N.; Schulte, S.; Eckert, J.; Berbner, R.; Steinmetz, R.: *An Approach to the Analysis and Evaluation of an Enterprise Service Ecosystem*. Akzeptierter Beitrag: ICSOFT 2007 Workshop on Architectures, Concepts and Technologies for Service Oriented Computing, Barcelona, Spanien.

Eckert, J.; Repp, N.; Schulte, S.; Berbner, R.; Steinmetz, R.: *An Approach for Capacity Planning for Web Service Workflows.* Akzeptierter Beitrag: 13th Americas Conference on Information Systems (AMCIS 2007), Keystone, CO, USA.

Schulte, S.; Repp, N.; Berbner, R.; Steinmetz, R.; Schaarschmidt, R.: *Service-Oriented Architecture Paradigm: Major Trend or Hype for the German Banking Industry.* Akzeptierter Beitrag: 13th Americas Conference on Information Systems (AMCIS 2007), Keystone, CO, USA.

Repp, N.; Heckmann, O.; Berbner, R., et al.: *Der Einfluß von Transportschicht-Anomalien auf die Performanz von Web Services.* In: 15. ITG/ GI-Fachtagung Kommunikation in Verteilten Systemen (KIVS 2007), Bern, Schweiz, 2007, S. 117-122.

Schulte, S.; Berbner, R.; Steinmetz, R.; Uslar, M.: *Implementing and evaluating the Common Information Model in a relational and RDF-based Database.* In: 3rd International ICSC Symposium on Information Technologies in Environmental Engineering (ITEE 2007), Oldenburg, Deutschland, 2007, S. 109-118.

Repp, N.; Schulte, S.; Eckert, J.; Berbner, R.; Steinmetz, R.: *Service-Inventur: Aufnahme und Bewertung eines Services-Bestands.* In: Workshop MDD, SOA und IT-Management (MSI 2007), Oldenburg, Deutschland, 2007, S. 13-22.

Repp, N.; Berbner, R.; Heckmann, O.; Steinmetz, R.: *A Cross-Layer Approach to Performance Monitoring of Web Services*. In: ECOWS Workshop on Emerging Web Services Technology (WEWST 2006), Zürich, Schweiz, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-234/paper2.pdf, 2006, Abruf am 09.02.2007.

Repp, N.; Berbner, R.; Lenz, J., et al.: *Digitalisierte eigenhändige Unterschrift im Online-Banking*. In: D-A-CH Security 2006, Düsseldorf, Deutschland, 2006, S. 381-391.

Steinmetz, R.; Berbner, R.; Martinovic, I.: *Web Services zur Unterstützung flexibler Geschäftsprozesse in der Finanzwirtschaft*. In: Sokolovsky Z. und Loeschenkohl S. (Hrsg.): Handbuch Industrialisierung der Finanzwirtschaft, Wiesbaden, Gabler, 2004, S. 641-654.

C.4 Weitere Veröffentlichungen

Berbner, R.; Repp, N.; Heckmann, O.; Steinmetz, R.: *SOA zum Durchbruch verhelfen. Management der Dienstgüte in Service-orientierten Architekturen.* In: IT-Management (2006) 3, S. 23-27.

Berbner, R.; Johannsen, W.; Goeken, M.; Repp, N.; Eckert, J.; Steinmetz, R.: *SOA Governance - Management of Opportunities and Risks*. In: Newsletter des E-Finance Lab e.V., Ausgabe 4/ 2006, S. 4-5.

Berbner, R.; Heckmann, O.; Steinmetz, R.: *Management of Service-oriented Architectures (SoA) within the E-Finance Industry*. In: Newsletter des E-Finance Lab e. V., Ausgabe 2/ 2005, S. 6-8.

C.5 Technical Reports

Spahn, M.; Berbner, R.; Heckmann, O.; Steinmetz, R.: *Ein heuristisches Optimierungsverfahren zur dienstgütebasierten Komposition von Web Service Workflows.* Technische Universität Darmstadt, Fachgebiet KOM. Technical Report KOM 02-2006, 2006.

Spahn, M.; Berbner, R.; Heckmann, O.; Mauthe, A.; Steinmetz, R.: *WSQoSX – Eine dienstgütebasierte Web Service Architektur.* Technische Universität Darmstadt, Fachgebiet KOM. Technical Report KOM 07-2004, 2004.

D Lebenslauf des Verfassers

Persönliche Daten

Name:	Rainer Berbner
Geburtsdatum:	4. November 1976
Geburtsort:	Mannheim
Nationalität:	Deutsch

Schulausbildung

1983 – 1987	Grundschule Mörlenbach-Weiher
1987 – 1996	Überwald Gymnasium Wald-Michelbach, Abschluss: Abitur

Wehrdienst

1996 – 1997	Grundwehrdienst Germersheim, Karlsruhe

Studium

1997 – 2003	Technische Universität Darmstadt Studiengang: Wirtschaftsinformatik, Abschluss: Dipl.-Wirtsch.-Inform., Note „Sehr gut“
2000	Technische Universität Eindhoven, Niederlande (Auslandssemester)

Berufliche Tätigkeiten

1997 – 1998	Freudenberg Informatik KG, Weinheim (Prakikant und Werkstudent)
1999 – 2001	Deltaconor GmbH, Darmstadt, Freier Mitarbeiter
2000	T-Online International GmbH, Weiterstadt (Praktikant)
2002	Mercedes Benz US International, Tuscaloosa, AL, USA (Praktikant)
Seit 2003	Technische Universität Darmstadt, Wissenschaftlicher Mitarbeiter am von Prof. Dr.-Ing. Ralf Steinmetz geleiteten Fachgebiet Multimedia Kommunikation (KOM)
Seit 2003	Stipendiat bzw. Kollegiat im DFG Graduiertenkolleg „Enabling Technologies for Electronic Commerce”
Seit 2006	Leiter der Forschungsgruppe IT-Architekturen am Fachgebiet KOM
Seit 2006	Mitbegründer und Leiter des SOA Competence Centers (httc e. V.)

Mitwirkung in Programmkomitees

2006	Workshop im Rahmen der Jahrestagung Informatik 2006, Dresden
2007	Workshop IAMCOM 2007 (IEEE COMSWARE 2007), Bangalore, Indien

Lehre

Betreuung von Seminararbeiten zum Thema Web Service Technologie am Fachgebiet Softwaretechnik im WS 2004/05, WS 2005/06 und SS 2006.

Betreuung von 14 Studien-/ Diplomarbeiten.

E Eidesstattliche Erklärung laut §9 PromO

Ich versichere hiermit an Eides statt, dass ich die vorliegende Dissertation alleine und nur unter Verwendung der angegebenen Literatur und Hilfsmittel erstellt habe.

Die Arbeit hat noch nicht zu Prüfungszwecken gedient.

Darmstadt,