Autonomes Performance-Management in dienstorientierten Architekturen

D I S S E R T A T I O N

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
im Fachbereich 16 — Elektrotechnik / Informatik
der Universität Kassel

vorgelegt von

Markus Schmid

Tag der Disputation: 30. November 2009

Erklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbständig und ohne unerlaubte Hilfe angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren verwendet worden.

Wiesbaden, 17.09.2009

Markus Schmid

Danksagung

Ich bedanke mich herzlich bei all jenen Personen, die mich auf unterschiedliche Weise unterstützt und zur Entstehung dieser Dissertation beigetragen haben.

Mein besonderer Dank gilt Prof. Dr. Reinhold Kröger, der meinen wissenschaftlichen Werdegang vom Diplom über mein Master-Studium bis hin zur Promotion entscheidend unterstützt hat und dabei stets Zeit für die Diskussion fachlicher Fragestellungen fand. Das durch ihn geleitete Labor für Verteilte Systeme der Hochschule RheinMain bot mir die sehr produktive Atmosphäre zur Erstellung dieser Arbeit.

Weiterhin gilt mein Dank Prof. Dr. Kurt Geihs für die gute und sehr effiziente Betreuung sowie die wertvollen inhaltlichen Anregungen zum bearbeiteten Themenbereich.

Ebenfalls danken möchte ich meinen Kollegen und den Studenten im Labor für Verteilte Systeme für die Bereitschaft, Themen aus meiner Arbeit im Detail mit mir zu diskutieren und Lösungsansätze kritisch zu hinterfragen. Den Diplomanden, Master-Studenten und studentischen Hilfskräften, die zur prototypischen Implementierung des Ansatzes beigetragen haben, gilt ebenfalls mein Dank.

Meiner Frau Carolin möchte ich für das Korrekturlesen der Dissertation und für ihr Verständnis und die Geduld danken, die sie insbesondere in der letzten Phase der Arbeit aufbrachte. Emma und Jonas danke ich dafür, dass sie sich und mir Zeit ausreichend gelassen haben, die Arbeit fertig zu stellen, bevor ich mich anderen Aufgaben zuwenden musste.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Dieses Kapitel führt in die Thematik der Arbeit ein. Abschnitt 1.1 beschreibt das thematische Umfeld und gibt eine Motivation für die behandelte Fragestellung. In Abschnitt 1.2 werden die Fragestellung und zentrale Lösungsansätze dieser Arbeit skizziert. Die zur Erarbeitung der Lösungsansätze verwendete Methodik wird in Abschnitt 1.3 dargelegt, Abschnitt 1.4 fasst den Aufbau und die Ergebnisse der Arbeit zusammen.

1.1 Motivation

In Unternehmen aller Branchen ist in den letzten Jahren ein starker Trend zur Einführung IT-gestützter Geschäftsprozesse zu beobachten. Auf technischer Ebene wird diese Entwicklung durch den Einsatz unternehmensweiter, verteilter Anwendungen ermöglicht, die klassischerweise als Client/Server-Systeme realisiert sind und von mehreren Benutzern gleichzeitig genutzt werden können. Solche Anwendungen werden auch als *Geschäftsanwendungen (Enterprise Applications)* bezeichnet.

Nichtfunktionale Anforderungen an Geschäftsanwendungen umfassen u. a. Skalierbarkeit und Nebenläufigkeit, Flexibilität zur Abbildung heutiger und zukünftiger Geschäftsprozesse, Integration mit im Unternehmen bereits existierenden Systemen und Interoperabilität mit Systemen kooperierender Unternehmen (*Business to Business, B2B*). Für die Anbieter von Geschäftsanwendungen sind zudem eine insgesamt geringe Entwicklungszeit, gute Wartbarkeit und die Möglichkeit zur flexiblen Anpassung ihres Systems an sich ändernde Geschäftsprozesse und Wünsche unterschiedlicher Kunden von Interesse.

Vor diesem Hintergrund wird für zukünftige Geschäftsanwendungen eine weitgehende Entkopplung von Anwendungskomponenten propagiert. Ziel ist die Definition einzelner, voneinander unabhängiger *Dienste (Services)*, die temporär zu Anwendungen verknüpft werden können. Diese Dienste werden von einzelnen Organisationseinheiten eines Unternehmens angeboten, von externen Anbietern zugekauft oder angemietet und fallen damit auch administrativ in unterschiedliche Zuständigkeitsbereiche. Sie verbergen ihre Implementierung vollständig hinter einer implementierungsunabhängigen Schnittstelle, sodass eine transparente Integration von Legacy-Systemen problemlos möglich ist. Als Anwendung fasst man jeweils einen IT-gestützten Geschäftsprozess des Unternehmens auf. Zur Umsetzung dieser Geschäftsprozesse werden einzelne Dienste zur Laufzeit durch formal beschriebene *Arbeitsabläufe (Workflows)*

verknüpft, die dann durch eine Laufzeitumgebung interpretiert und ausgeführt werden. Änderungen in Geschäftsprozessen bedingen somit lediglich die Adaption einzelner Workflow-Definitionen, die Diensteinfrastruktur des Unternehmens muss nicht angepasst werden. Diese Sichtweise auf die IT-Prozesse eines Unternehmens bezeichnet man als *dienstorientierte Architektur (Service Oriented Architecture, SOA)*. Die modulare Struktur einer SOA erlaubt neben der leichten Anpassung von Geschäftsprozessen auch die weitgehend transparente Auslagerung von Diensten und Teilprozessen an externe Dienstleister.

Die Realisierung der Geschäftsanwendungen eines Unternehmens in Form einer SOA bietet durch die Entkopplung von Diensten und Geschäftsprozessen und die implementierungsunabhängige Kapselung einzelner Dienste viele Vorteile für den Betreiber. Betrachtet man allerdings die technische Architektur insgesamt, kann man im Vergleich zu Applikationsserver-basierten Multi-Tier-Anwendungen nochmals eine deutliche Komplexitätszunahme feststellen. Problematisch ist insbesondere die Verteilung der administrativen Zuständigkeit für einzelne Teilsysteme auf unterschiedliche Abteilungen und externe Partner. In die Überwachung und Kontrolle der Funktion einzelner Prozesse, das für geschäftskritische Anwendungen unbedingt nötige *Dienstgüte-Management (Service Level Management, SLM)* [SMJ00, Lew99], muss damit eine Vielzahl von Komponenten einbezogen werden.

Das SLM beschäftigt sich mit der Einhaltung von Dienstgütevereinbarungen für IT-Dienste, die oftmals auf einer Reihe unterschiedlicher Hardware- und Softwarekomponenten aufbauen. Solche Dienstgütevereinbarungen werden formal als *Service Level Agreements (SLAs)* zwischen Anbieter und Nutzer eines Dienstes geschlossen. Als SLA bezeichnet man einen Vertrag, der Dienstgüteparameter – wie z. B. Antwortzeit- oder Verfügbarkeitskriterien für Funktionen eines Dienstes – in Form von *SLA-Parametern* spezifiziert. Die an der technischen Schnittstelle zwischen den Vertragspartnern zu erbringende Dienstgüte wird dabei für die einzelnen SLA-Parameter in Form von so genannten *Service Level Objectives (SLOs)* vertraglich im SLA festgeschrieben. Zur Laufzeit ist es Aufgabe des SLMs, die definierten SLA-Parameter mit ihren SLOs permanent zu überwachen, Übertretungen zu registrieren und ggf. korrigierend einzugreifen.

Dienstgüteanforderungen für Geschäftsanwendungen beziehen sich häufig auf Verfügbarkeit und Ausführungsgeschwindigkeit von Prozessen *(Performance-Anforderungen)*. Bei großen Anwendungen werden solche Anforderungen i. d. R. für einzelne Teilsysteme spezifiziert.

Diese Arbeit konzentriert sich auf den Bereich des *Performance-Managements* als Teilbereich des SLMs. Unter dem Begriff *Performance* werden Kenngrößen zusammengefasst, die sich mit zeitlichen Aspekten der Leistungserbringung von Diensten befassen. Typische Performance-Kenngrößen sind beispielsweise Festlegungen der *maximal zulässigen Antwortzeit* von Anfragen oder des *Mindestdurchsatzes* eines Dienstes.

Für die Definition von SLAs ist der modulare Aufbau einer SOA-basierten Geschäftsanwendung von Vorteil, da Dienstgüteanforderungen für die klar definierten Schnittstellen der einzelnen Dienste spezifiziert werden können. Für das SLM insgesamt ist die dezentrale Struktur

einer SOA mit unterschiedlichen administrativen Zuständigkeiten für einzelne Teilsysteme allerdings problematisch, da regelnde Eingriffe zum einen durch die Kapselung der Implementierung einzelner Dienste und zum anderen durch das Fehlen einer zentralen Kontrollinstanz nur sehr eingeschränkt möglich sind. Zudem entstehen in einem Netz wechselnder vertraglicher Abhängigkeiten zwischen unterschiedlichen Partnern Workflows und (aus anderen Diensten zusammengesetzte) Dienste, die nicht mehr eindeutig einer einzelnen administrativen Domäne zugeordnet werden können.

Das Dienstgüte-Management in solchen Umgebungen ist mit den bisher gängigen, zentralisierten Management-Plattformen nur sehr eingeschränkt möglich. Zu diesen zählen umfangreiche kommerzielle Systeme wie beispielsweise IBM Tivoli, CA Unicenter oder HP OpenView, die ursprünglich als Management Frameworks konzipiert wurden, mittlerweile aber aus einer Vielzahl spezialisierter, teilweise unabhängiger Module bestehen. Diese Systeme fokussieren allerdings primär auf das Management statischer Strukturen. Sie sind prinzipiell hierarchisch organisiert und verfügen über eine zentrale Management-Konsole, mit deren Hilfe die Management-Entscheidungen für ein bestimmtes Modul letztendlich durch einen Administrator getroffen werden. Eine zentrale Management-Konsole ist für die Verwaltung einer mehrere administrative Domänen umfassenden Architektur nur sehr bedingt geeignet. Auch ist der durch diese Lösungen zur Verfügung gestellte Management-Automatisierungsgrad noch unzureichend und begrenzt so eine noch weitergehende Skalierbarkeit. Gleichzeitig definiert die SOA eine homogene Sichtweise auf die in einem Unternehmen vorhandene Software und bietet somit bereits die Grundlage für ein einheitliches, automatisierbares Dienstgüte-Management auf Applikationsebene, die von bisherigen Ansätzen allerdings nicht genutzt wird.

Historisch wurden im IT-Management zunächst einzelne Systemschichten isoliert voneinander betrachtet. So entstanden unabhängige Plattformen für das Netzwerk-Management, das Systemmanagement, usw. bis hin zu Lösungen für das Management von Daten und Anwendungen. Diese traditionelle Aufgabenteilung findet sich auch in klassischen Berufsbezeichnungen wieder, z. B. Netzwerkadministrator, Systemadministrator oder Datenbankadministrator. Mit dem Aufkommen komplexerer Geschäftsanwendungen auf Applikationsserver-Basis wurde dieses horizontale Schichtenmanagement zunehmend durch vertikale Management-Ansätze verdrängt, bei denen Management-Plattformen und Administratoren aus Geschäftssicht zusammenhängende Bereiche abdecken („SAP-Administrator“).

In aktuellen SOA-basierten Geschäftsanwendungen kann aufgrund der vorhandenen Komplexität und Dynamik sowie der verteilten administrativen Zuständigkeiten weder ein rein horizontaler noch ein vertikaler Management-Ansatz verfolgt werden.

Für das SLM von SOA Workflows müssen temporär diensteübergreifende Kooperationen zur Erfüllung von Dienstgüteanforderungen etabliert werden, die sich auch über mehrere administrative Domänen erstrecken können. Eine solche zeitlich begrenzte Kooperation unterschiedlicher Teilsysteme kann sinnvollerweise nur dezentral erfolgen, da die jeweiligen Kooperationspartner im Vorfeld nicht bekannt sind und – je nach Lebensdauer einzelner Workflows – häufige Partnerwechsel zur Laufzeit auftreten können.

Für das Durchsetzen vereinbarter Dienstgüteanforderungen in der Implementierung einzelner Dienste ist ein vertikaler Management-Ansatz vorstellbar, da anzunehmen ist, dass alle an der Realisierung eines Dienstes beteiligten Systeme einer einheitlichen Administration unterliegen. Im Kontext des SLMs kann ein einzelner Dienst somit als äquivalent zu einer traditionellen, eigenständigen Anwendung betrachtet werden.

Ein SLM-Ansatz für SOA Workflows und Dienste muss dabei weitestgehend automatisiert agieren können, um die durch die SOA erreichbare Skalierbarkeit und Dynamik nicht zu beschränken.

1.2 Fragestellung und Lösungsansätze

Ziel dieser Arbeit ist die Entwicklung eines SLM-Ansatzes für das autonome Performance-Management in dezentralen dienstorientierten Architekturen.

Für SOA-basierte Geschäftsanwendungen werden Performance-bezogene Dienstgüteanforderungen und daraus resultierende Dienstgütevereinbarungen für einzelne Prozesse und Workflows formuliert. Um eine geordnete Leistungserbringung durch einen Workflow sicherzustellen, müssen die beteiligten Dienste ihrerseits gewisse Dienstgüteanforderungen erfüllen.

Wie in Abschnitt 1.1 dargestellt, ist für SOAs eine hohe Laufzeitdynamik charakteristisch, die sich durch häufige Workflow-Änderungen innerhalb der Architektur ausdrückt. Durch die starke Modularisierung der Architektur in Dienste und Workflows und durch verteilte administrative Zuständigkeiten entsteht gleichzeitig eine Vielzahl von Schnittstellen, für die formale Dienstgütevereinbarungen getroffen werden können und müssen. Gerade im Fall der Auslagerung von Teilsystemen an andere Unternehmen können sich Dienstgütevereinbarungen sowohl auf komplexe Arbeitsabläufe als auch auf Funktionen einzelner Basisdienste beziehen. Zudem werden Dienstgütevereinbarungen in komplexen Systemen häufig kundenspezifisch abgeschlossen, bei der Leistungserbringung wird dann intern eine Priorisierung der einzelnen Kunden vorgenommen.

Besondere Anforderung an ein SLM-System für dienstorientierte Architekturen ist somit die Möglichkeit einer automatisierten dynamischen Abbildung der für einen Workflow definierten Dienstgüteanforderungen auf die jeweils beteiligten Komponenten. Hierbei müssen die unter Umständen für die Nutzung dieser Dienste vorab verpflichtend getroffenen SLAs berücksichtigt werden. Ebenso muss ein solches System in der Lage sein, parallel mit unterschiedlichen kundenspezifischen Dienstgüteanforderungen für Workflows und Dienste umzugehen, und dabei den dezentralen Charakter der Architektur berücksichtigen.

Getroffene Dienstgütevereinbarungen auf Workflow- und Dienstebene müssen vom SLM-System durchgesetzt werden. Hierbei ergeben sich mitunter Situationen, in denen durch das

Management-System Workflow-übergreifende Priorisierungen vorgenommen werden müssen, ohne auf eine zentrale Entscheidungsinstanz zurückgreifen zu können. Gleichzeitig ergibt sich durch die einheitliche Betrachtung der gesamten Anwendungslandschaft eines Unternehmens die Möglichkeit, globale Optimierungen zur Minimierung des Gesamtressourcenverbrauchs vorzunehmen.

In Bezug auf die notwendige Reaktionsgeschwindigkeit des Systems ergeben sich für die Ebenen unterschiedliche Anforderungen: Während Rekonfigurationen auf Dienstebene als direkte Reaktion auf Zustandsänderungen der überwachten Implementierung rasch erfolgen müssen, ergibt sich für Management-Entscheidungen zur globalen Optimierung des Systems ein längerer Zeithorizont.

In einem SLM-System für dienstorientierte Architekturen sind somit prinzipiell zwei Ebenen zu unterscheiden: Zum einen das Dienstgüte-Management eines einzelnen Dienstes und seiner Implementierung, das auf Veränderungen weitgehend in Echtzeit reagieren muss, zum anderen das strategische Management von SLAs auf Workflow-Ebene, für das eine langsamere Reaktionsgeschwindigkeit als ausreichend erachtet wird.

Für die Architektur eines SLM-Systems für lose gekoppelte Architekturen erscheinen somit Skalierbarkeit, Dezentralität und Robustheit von zentraler Bedeutung zu sein. Eine Dezentralisierung der SLM-Architektur setzt die Etablierung lokaler Kontrollstrukturen und Regelungskreisläufe voraus. Im Zuge der zugleich geforderten hohen Skalierbarkeit des Systems sollen diese lokalen Kontrollstrukturen zur Laufzeit weitgehend ohne menschliche Eingriffe auskommen.

Ein aktuelles Forschungsgebiet der Informatik ist das *Selbst-Management* (auch *Autonomic Computing*) von IT-Komponenten [KC03], das eine Reduktion der nach außen sichtbaren Komplexität und eine Erhöhung der Skalierbarkeit durch Automatisierung von Management-Prozessen verfolgt. Ziel ist es dabei, Systeme zu kreieren, die sich auf Basis abstrakter Vorgaben selbst konfigurieren, optimieren und ggf. auch gegen Angriffe schützen oder reparieren. Solche Eigenschaften eines sich selbst managenden Systems werden in der Literatur auch als *Self-* Properties* [BJM+05] bezeichnet. Für derartige Systeme werden derzeit unterschiedlichste Steuerungsalgorithmen untersucht, darunter z. B. regelbasierte Ansätze, Neuronale Netze oder Verfahren aus der Kontrolltheorie.

Im Rahmen der Arbeit wird untersucht, inwiefern sich Selbst-Management- und Selbstorganisationsansätze einsetzen lassen, um eine flexible und skalierbare Lösung für das SLM in dienstorientierten Architekturen zu entwerfen. Weiter ist von Interesse, inwieweit sich ein solches selbstorganisierendes, dezentral aufgebautes System noch global kontrollieren lässt.

Hierzu werden die einzelnen Dienste innerhalb der Architektur mit einer Selbst-Management-Funktionalität ausgestattet, die für die Einhaltung der einem Dienst zugeordneten Dienstgüteanforderungen verantwortlich ist. Die Anpassung an heterogene Teilsysteme unterschiedlicher Hersteller wird durch die Nutzung standardisierter Management-Schnittstellen und die

Einbindung in Standard-Komponentenarchitekturen gewährleistet. Zur Optimierung der Ressourcennutzung kooperieren die Management-Komponenten der einzelnen Dienste miteinander und bilden temporäre Kommunikationsstrukturen, die sich an der Kommunikation auf der Geschäftsebene orientieren. Workflows werden mit einer Management-Komponente ausgestattet, die für die Aufteilung global gestellter Dienstgüteanforderungen auf die beteiligten Dienste verantwortlich ist. Entsprechend den globalen Vorgaben handelt die Workflow-Management-Komponente dann zur Durchführung des Workflows individuelle Dienstgütevereinbarungen mit den Management-Komponenten einzelner Dienste aus.

Zunächst wird für die Architektur ein abstraktes Konzept entwickelt, das dann beispielhaft auf ein gängiges SOA-Strukturmodell abgebildet wird. Zur Validierung des Ansatzes erfolgt eine prototypische Entwicklung des Systems.

1.3 Methodischer Ansatz

Als Grundlage für die in der Arbeit behandelten Fragestellungen werden grundlegende Anforderungen diskutiert, die eine SOA an ein Dienstgüte-Management-System stellt. Beispiele hierfür wurden bereits in Abschnitt 1.1 geschildert, sie umfassen Beschränkungen administrativer Art, die zu erwartende Komplexität, Skalierbarkeit und Dynamik des Gesamtsystems und Anforderungen, die z. B. beim Auslagern von Teilsystemen entstehen können.

Diese Fragestellungen bilden das Grundgerüst der Anforderungen an das zu entwerfende Dienstgüte-Management-System für SOAs, denen mithilfe von Methoden des Selbst-Managements und der Selbstorganisation begegnet werden soll.

Der Ansatz wird zunächst technologieunabhängig definiert und dann beispielhaft auf ein konkretes SOA-Strukturmodell abgebildet. Dieses Strukturmodell bildet wiederum die Grundlage der prototypischen Implementierung und damit auch der Evaluation des Ansatzes.

Für das detaillierte Design des Systems werden einige Schwerpunkte aus dem Gesamtkonzept ausgewählt, für andere Bereiche wird auf existierende Arbeiten verwiesen. Zur Demonstration des Ansatzes wird im Rahmen der Evaluation ein Beispielszenario beschrieben, das aus mehreren Diensten besteht. Anhand der prototypischen Implementierung erfolgt gleichzeitig die Überprüfung der für das System definierten Anforderungen.

Angestrebte nichtfunktionale Eigenschaften des Systems (Skalierbarkeit / Reaktionszeit) werden mithilfe von Simulationen evaluiert. Hierbei wird ein hybrider Simulationsansatz verfolgt, d. h. die Simulation nutzt ausgewählte Komponenten aus der prototypischen Realisierung des Systems zur Evaluation von Skalierbarkeitsaspekten.

1.4 Aufbau und Ergebnisse der Arbeit

Die Arbeit gliedert sich inhaltlich in drei Teile: Teil I führt in die Thematik der Arbeit ein und beschreibt verwandte Ansätze. In Teil II wird ein modularer Architekturansatz für ein autonomes Dienstgüte-Management-System für SOA-Umgebungen entwickelt. Auf dieser Architekturbasis werden Kooperationsmechanismen für Management-Komponenten diskutiert. Teil III beschreibt die prototypische Umsetzung und Evaluation zentraler Teile des entwickelten Dienstgüte-Management-Systems. Im Folgenden wird der Aufbau der Arbeit auf Kapitelebene beschrieben.

Kapitel 2 gibt einen Überblick über den Bereich dienstorientierter Architekturen. Dabei wird der Begriff der dienstorientierten Architektur sowohl aus Geschäftssicht als auch aus technischer Sicht beleuchtet, zusätzlich werden ausgewählte Technologien aus dem SOA-Kontext vorgestellt. Im Verlauf des Kapitels werden grundlegende Begriffe definiert. Diese bilden die Basis für die weiteren Ausführungen der Arbeit.

In Kapitel 3 wird ein Überblick über Begriffe und Technologien des IT-Managements gegeben. Der Fokus liegt hierbei auf dem SLM als der für das Verständnis dieser Arbeit relevanten Management-Teildisziplin. Weiterhin umreißt das Kapitel mit „Selbst-Management“ ein aktuelles Forschungsgebiet im Bereich des IT-Managements.

Kapitel 4 beschreibt Forschungsarbeiten, die die in dieser Arbeit behandelte Thematik inhaltlich ergänzen und abrunden.

In Kapitel 5 werden als Resultat der Betrachtungen aus Teil I Anforderungen für ein Dienstgüte-Management-System für SOA-Umgebungen definiert. Zentrale Design-Aspekte sind hierbei die Unterstützung von dezentraler Kontrolle und flexible Anpassbarkeit an unterschiedliche Dienste. Im Anschluss wird eine modulare Architektur für ein solches Dienstgüte-Management-System entwickelt.

Kapitel 6 beschreibt die zentrale Idee der Arbeit: Autonome, jeweils einem SOA-Dienst zugeordnete Management-Komponenten kooperieren zur Erfüllung übergeordneter Antwortzeitanforderungen miteinander. Für die Kooperation wird in der Arbeit ein dezentraler Ansatz verfolgt, der sich an Änderungen der Struktur von Workflows und Diensten innerhalb der unter Management stehenden dienstorientierten Architektur automatisch anpasst. Zu diesem Zweck bilden die beteiligten Dienste temporäre Kommunikationsstrukturen (Selbstorganisation). Ziel der Kooperation ist das Übertragen von ungenutzten Antwortzeit-SLO-Anteilen zwischen einzelnen Diensten, wodurch letztendlich eine Verringerung der benötigten Ressourcen bzw. Kosteneinsparungen erreicht werden können.

Kapitel 7 beschreibt die prototypische Umsetzung zentraler Architekturelemente und der entwickelten Verfahren zur Selbstorganisation beispielhaft für das Dienstgüte-Management konkreter Komponenten. Zur Abbildung größerer Szenarien wird ein hybrider Simulationsansatz

implementiert, der zur Kommunikation zwischen einzelnen Management-Komponenten auf Elemente der prototypischen Implementierung zurückgreift.

In Kapitel 8 werden Architektur und prototypische Implementierung im Hinblick auf die in Kapitel 5 definierten Anforderungen evaluiert. Anhand einer Fallstudie wird die Funktionsweise des Systems detailliert nachvollzogen. Zur Bewertung von Flexibilität und Skalierbarkeit des Ansatzes kommt der entwickelte Simulator zum Einsatz.

Kapitel 9 fasst die wichtigsten Ergebnisse der Arbeit zusammen. Im Anschluss werden weitergehende Fragestellungen diskutiert, die sich aus den Untersuchungen ergeben haben.

In der Arbeit wird weitgehend auf eine deutsche Terminologie zurückgegriffen. Englische Fachbegriffe werden synonym verwendet oder an Stellen eingesetzt, an denen keine äquivalente deutsche Terminologie existiert.

Teil I

Grundlagen

2 Dienstorientierte Architekturen

In diesem Kapitel wird eine Übersicht über grundlegende Begriffe aus dem Kontext dienstorientierter Architekturen gegeben. Abschnitt 2.1 beschreibt die historische Entwicklung von verteilten Geschäftsanwendungen bis hin zum Konzept der dienstorientierten Architekturen. Abschnitt 2.2 gibt einen Überblick über das Architekturkonzept SOA und wesentliche Architekturelemente. Im Anschluss wird die SOA in Abschnitt 2.3 zunächst aus der Geschäftsperspektive und in Abschnitt 2.4 aus technischer Sicht beleuchtet, bevor vertiefend auf die wesentlichen Architekturelemente „Dienste“ und „Workflows“ eingegangen wird.

Nach diesem allgemeinen Überblick werden exemplarisch einige Technologien zur Realisierung von SOA-Anwendungen vorgestellt, die teilweise für die praktische Umsetzung des in dieser Arbeit entwickelten Ansatzes zum Dienstgüte-Management in SOAs von Bedeutung sind: In Abschnitt 2.5 werden formale Workflow-Beschreibungssprachen für unterschiedliche Anwendungsbereiche vorgestellt, bevor in Abschnitt 2.6 mit SCA ein genereller Ansatz zur Beschreibung statischer Abhängigkeiten zwischen SOA-Diensten beschrieben wird.

2.1 Historische Entwicklung

Nichtfunktionale Anforderungen wie Interoperabilität und Skalierbarkeit haben ab Anfang der 1990er Jahre dazu geführt, dass Geschäftsanwendungen zunehmend auf der Basis von Standard-Middleware-Architekturen entwickelt wurden. Diese ermöglichen prinzipiell eine herstellerübergreifende Integration von Anwendungen, da sie eine grundlegende Kommunikationsinfrastruktur bereitstellen. Klassische Beispiele für kommunikationsorientierte Middleware-Technologien sind die *Common Object Request Broker Architecture (CORBA)* [OMG08] der OMG, *Microsoft DCOM* [HK97] oder *Java Remote Method Invocation (RMI)* [SUN03a] von Sun Microsystems. Die Verwendung solcher Middleware-Technologien konnte zwar eine grundlegende Interoperabilität unterschiedlicher Anwendungen gewährleisten, brachte aber in Bezug auf andere nichtfunktionale Anforderungen keine weiteren Vorteile. Durch den Einsatz von Middleware-Technologien erhöhte sich die Komplexität der Anwendungsarchitektur, gleichzeitig machte die explizit ausprogrammierte enge Verknüpfung unterschiedlicher Teilsysteme bei Änderungen in einem Teilbereich unter Umständen größere Umbauarbeiten an anderen Anwendungsteilen nötig.

Heute wird die Geschäftslogik von Geschäftsanwendungen auf Basis so genannter *Applikationsserver* implementiert, die dem Entwickler neben standardisierter Kommunikation zusätzliche Dienste wie beispielsweise Transaktionsunterstützung, Authentifizierung und Autorisierung von Benutzern, einheitliche Persistenzschnittstellen usw. bereitstellen. Die eigentliche Geschäftslogik läuft dabei in einer standardisierten Ablaufumgebung – einem so genannten *Container* – ab, die Art der Nutzung der vom Server bereitgestellten Dienste und Verbindungen zur Außenwelt wird nicht programmatisch festgelegt, sondern im Server konfiguriert. Dies stellt beispielsweise die Portierbarkeit der vom Hersteller implementierten Geschäftslogik zwischen Applikationsservern verschiedener Anbieter sicher. Gleichzeitig erlauben moderne Applikationsserver die transparente Verknüpfung unterschiedlicher Module zu einer Gesamtapplikation, wodurch das Zusammensetzen einer Geschäftsanwendung aus mehreren so genannten *Anwendungskomponenten* ermöglicht wird. Die durch die Definition von Anwendungskomponenten erreichbare Modularisierung einer Applikation erlaubt einem Softwareanbieter eine individuelle Anpassung seiner Anwendung an die Bedürfnisse einzelner Kunden und ermöglicht gleichzeitig die Integration von Komponenten anderer Hersteller.

Ein Beispiel für eine solche Komponententechnologie und Applikationsserver-Plattform ist die von Sun Microsystems entwickelte *Enterprise JavaBeans (EJB)* [SUN06b] Technologie, die als Bestandteil des *Java Enterprise Frameworks (JEE)* [SUN06a] spezifiziert ist und durch verschiedene Hersteller implementiert wird. Die EJB-Spezifikation definiert Anforderungen an Applikationsserver bezüglich Persistenz, Transaktionsverarbeitung, Nebenläufigkeit, Sicherheit, Anbindung an Namensdienste und externe Dienste.

Die oben beschriebene Entwicklung in der Architektur von Geschäftsanwendungen führt zur weitgehenden Verlagerung der Skalierbarkeitsproblematik auf die unterlagerte Applikationsserver-Infrastruktur, zugleich jedoch zu einer massiven Erhöhung der Komplexität der Softwarearchitektur insgesamt. Gleichzeitig steigt mit zunehmender Durchdringung von Unternehmen mit Geschäftsanwendungen und durch Integration unterschiedlicher Prozesse die Anzahl der an einer Anwendung beteiligten Anwendungskomponenten stetig. Restrukturierungen, Unternehmenszukäufe und Verkäufe von Unternehmensteilen sowie der wachsende Markt für Auslagerung *(Outsourcing)* von IT-gestützten Diensten machen Anpassungen der in einem Unternehmen eingesetzten Geschäftsanwendungen notwendig. Durch die in der Software-Entwicklung gegebenen kurzen Innovationszyklen verursachen Anpassungen an einer Geschäftsanwendung häufig zusätzlich technologiebedingte Änderungen. Diese wirken sich im ungünstigen Fall wiederum wegen der oben geschilderte Verflechtung unterschiedlicher Anwendungskomponenten auf andere Teilsysteme aus.

Vor diesem Hintergrund wird für zukünftige Geschäftsanwendungen eine noch weitergehende Entkopplung von Anwendungskomponenten propagiert. Ziel ist die Definition einzelner, voneinander unabhängiger *Dienste (Services)*, die temporär zu Anwendungen verknüpft werden können. Als Anwendung fasst man jeweils einen IT-gestützten Geschäftsprozess des Unternehmens auf. Zur Umsetzung dieser Geschäftsprozesse werden einzelne Dienste zur Laufzeit durch formal beschriebene *Arbeitsabläufe (Workflows)* verknüpft. Diese Sichtweise auf die

IT-Prozesse eines Unternehmens bezeichnet man als *dienstorientierte Architektur (Service Oriented Architecture, SOA)*. Die modulare Struktur einer SOA erlaubt neben einfacher Anpassung von Geschäftsprozessen auch die weitgehend transparente Auslagerung von Diensten und Teilprozessen an externe Dienstleister.

Die Bezeichnung „Service Oriented Architecture“ wurde für den Kontext von Geschäftsanwendungen im Jahr 1996 durch die Gartner Group [SN96] eingeführt. Gartner definiert eine dienstorientierte Architektur wie folgt:

Definition 2.1: SERVICE ORIENTED ARCHITECTURE
Service Oriented Architecture (SOA) ist ein Ansatz zum Design von Client/Server-Software, bei dem sich eine Anwendung aus Software Services (Diensten) und Software Service Konsumenten (auch als Clients oder Service Requesters bezeichnet) zusammensetzt. SOA unterscheidet sich vom allgemeinen Client/Server-Modell durch die starke Betonung der losen Kopplung von Software-Komponenten und der Nutzung explizit beschriebener Schnittstellen [NS03].

Im Folgenden wird ein allgemeiner Überblick über das Architekturkonzept SOA gegeben.

2.2 SOA im Überblick

Als Grundlage für eine Vereinheitlichung von Terminologie und Architekturkonzepten im SOA-Umfeld hat die Organization for the Advancement of Structured Information Standards (OASIS) im Jahr 2006 ein Referenzmodell für dienstorientierte Architekturen vorgestellt (vgl. [OAS06a]). Das Referenzmodell beschreibt das Konzept eines SOA-Dienstes wie folgt:

> *A **service** is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description* [OAS06a, Seite 12].

Das Referenzmodell spezifiziert damit einen Dienst als Schnittstelle zu einer gekapselten Funktionalität, der eine Dienst-Beschreibung beigefügt ist. Ein zum OASIS-Referenzmodell konformer Dienst ist damit selbstbeschreibend und kann Rahmenbedingungen für seine Nutzung durch Dritte vorgeben. Explizit verweist OASIS in der grundlegenden Definition einer SOA darauf, dass die Aufgabe einer SOA darin besteht, das Zusammenspiel unterschiedlicher Dienste über administrative Grenzen hinweg einheitlich zu strukturieren (vgl. [OAS06a, Seite 8]).

Die sehr abstrakte Sicht des Referenzmodells wird durch die OASIS-Referenzarchitektur für SOA [OAS08] weiter spezifiziert; diese liegt derzeit in einer Vorabversion vor. Sie beschreibt die Elemente einer SOA technologieunabhängig, strukturiert die Kommunikationsbeziehungen der unterschiedlichen Akteure und spezifiziert zusätzlich Governance- und Management-Aspekte für Dienste. Die OASIS-Referenzarchitektur stimmt mit der dieser Arbeit auf technologieunabhängiger Ebene zugrunde liegenden Sichtweise auf eine SOA überein.

Nach [ST07] definiert man in einer SOA Anwendungen als ausführbare Abbildung von Geschäftsprozessen, als so genannte *Arbeitsabläufe* oder *Workflows*. Bei der Ausführung eines Workflows interagieren einzelne geschäftsbezogene Software-Komponenten, so genannte *Geschäfts-* oder *Funktionskomponenten*, miteinander, die jeweils eine Reihe von *Geschäftsfunktionen* bereitstellen (vgl. Abbildung 2.1).

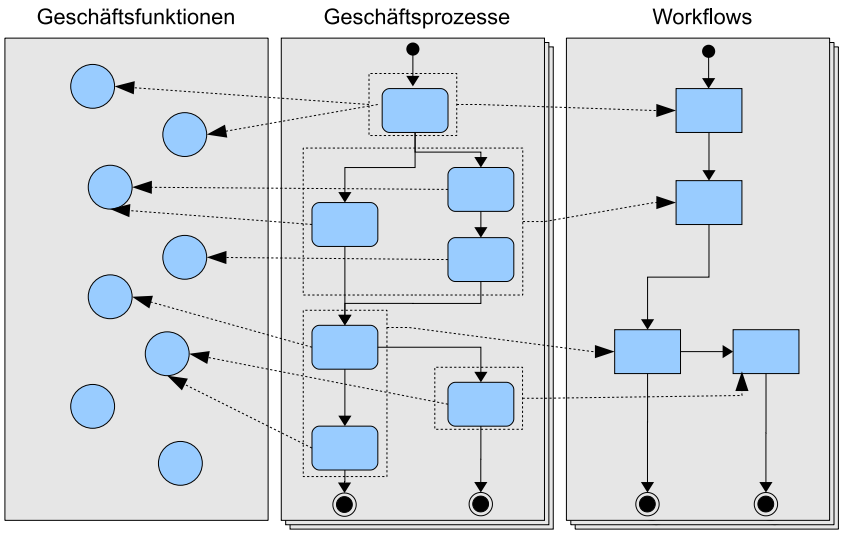

Abbildung 2.1: Beziehung zwischen Geschäftsfunktionen, Geschäftsprozessen und Workflows nach [ST07]

Nach [ST07] ist für Geschäftsfunktionen charakteristisch, dass diese kontextunabhängig bestimmte exakt definierte fachliche Aufgaben erfüllen. Eine SOA spezifiziert lediglich die zur Bereitstellung der Geschäftsfunktionen notwendigen Schnittstellen der einzelnen Geschäftskomponenten; der interne Aufbau einer Komponente sowie die Abbildung auf darunterliegende (Hardware-)Ressourcen ist nicht Teil der Architektur. Die eine Geschäftsfunktion bereitstellende Schnittstelle bezeichnet man als *Dienst* oder *Service*.

Im Rahmen dieser Arbeit wird ein SOA-Dienst wie folgt definiert:

Definition 2.2: SOA-DIENST
Ein SOA-Dienst stellt den formal spezifizierten Zugangspunkt für eine bestimmte, in sich geschlossene Funktionalität dar. Neben der eigentlichen Schnittstelle spezifiziert der Dienst auch nichtfunktionale Aspekte sowie Randbedingungen für seine Nutzung. Die Beschreibung des Dienstes ist unabhängig von Realisierungsplattform und Programmiersprache der Geschäftskomponente, die die Funktionalität letztendlich implementiert.

Definition 2.2 vereint damit die Sichtweisen aus [OAS06a] und [ST07].

Die Bausteine einer SOA bilden lose gekoppelte Dienste gemäß Definition 2.2. Diese können in einer standardisierten Art und Weise durch Workflows aufgerufen werden und werden so innerhalb der Architektur temporär zu Anwendungen verbunden. Workflows werden in einer formalen Workflow-Beschreibungssprache notiert.

Unter einem Workflow versteht man die „prozessorientierte Abwicklung arbeitsteiliger Vorgänge beziehungsweise Geschäftsvorfälle in Unternehmen und Behörden mit dem Ziel größtmöglicher Effizienz“ [Bro07]. Ein Workflow beschreibt damit eine vordefinierte Abfolge von Vorgängen innerhalb eines Unternehmens und grenzt sich vom – oftmals synonym verwendeten – Begriff des Geschäftsprozesses durch eine technischere Sichtweise auf die beschriebenen Prozesse und Teilsysteme ab. In der IT sind typische Anwendungsbereiche für Workflow-Technologien z. B. im Bereich Geschäftsprozessmodellierung und -koordination sowie in der Verwaltung von Dokumenten zu finden.

Im Rahmen dieser Arbeit wird ein Workflow wie folgt definiert:

Definition 2.3: WORKFLOW
Ein Workflow beschreibt eine vordefinierte Abfolge von Vorgängen in einer formalen Notation. Im SOA-Kontext spezifiziert ein Workflow eine maschinenlesbare Verknüpfung unterschiedlicher SOA-Dienste zur Implementierung eines Geschäftsprozesses.

Workflow-Beschreibungen werden von einem *Workflow Management-System (WfMS)* interpretiert und ausgeführt. Neue Workflow-Beschreibungen können auch zur Laufzeit entstehen, z. B. als Resultat eines Dienst-Aufrufs. Workflows können andere Workflows auf die gleiche Weise wie einfache Dienste integrieren, sodass eine komplexe Workflow-Hierarchie entstehen kann. Der Benutzer interagiert mit SOA-Anwendungen über ein Portal, das Anwendungen aus unterschiedlichen fachlichen Domänen vereinen kann.

Abhängig vom Anwendungsumfeld unterscheidet man zwei unterschiedliche Vorgehensweisen beim Design eines Workflows:

Kann die Abarbeitung der abzubildenden Geschäftsprozesse durch eine zentrale WfMS-Instanz im Unternehmen gesteuert werden und bestehen keine externen Abhängigkeiten, spricht man bei der Verknüpfung von Diensten zu einem Workflow von *Orchestrierung (Orchestration)*. Im Rahmen der Orchestrierung von Diensten wird eine klare Hierarchie etabliert, die Steuerung der resultierenden Anwendung erfolgt durch ein zentrales WfMS. [OAS08] beschreibt die Orchestrierung von Diensten wie folgt:

> *A technique used to compose hierarchical and self-contained service-oriented business processes that are executed and coordinated by a single agent acting in a „conductor" role* [OAS08, Seite 69].

Kooperieren unterschiedliche Akteure (z. B. Partnerunternehmen) zur Implementierung gemeinsamer Geschäftsprozesse, kann keine klare Hierarchie mit zentraler Kontrolle der Prozesse etabliert werden. Vielmehr implementiert jeder Akteur seine eigenen Workflows, die durch ein lokales WfMS abgearbeitet werden und durch Nachrichtenaustausch mit den Workflows der anderen Akteure kommunizieren. Bei der Implementierung solcher administrative Grenzen überschreitender Geschäftsprozesse spricht man von *Choreographie (Choreography)*. In [OAS08] wird die Choreographie von Diensten wie folgt beschrieben:

> *A technique used to characterize and to compose service-oriented business collaborations based on ordered message exchanges between peer entities in order to achieve a common business goal* [OAS08, Seite 71].

In der Praxis findet man oftmals eine Kombination beider Verfahren: Innerhalb einer Organisation werden Dienste zunächst hierarchisch orchestriert, zwischen kooperierenden Unternehmen werden übergreifende Prozesse choreographiert.

Im Folgenden wird das Architekturkonzept SOA zunächst aus der Geschäftsperspektive betrachtet, im Anschluss wird dann näher auf Möglichkeiten zur technischen Realisierung eingegangen.

2.3 Geschäftssicht

In [STS+04] wird die Geschäftssicht auf eine SOA am Beispiel des SAP-Systems so beschrieben, wie sie in Abbildung 2.2 dargestellt ist.

Zentrales Element der Architektur ist eine *Business Process Engine (BPE)*, die Portal und WfMS umfasst und über *Enterprise Application Integration (EAI)*-Schnittstellen zur Anbindung unterschiedlichster Backend-Systeme und Dienste verfügt. Das zentrale *Enterprise Resource Planning System (ERP)* offeriert als Backend-System seine Funktionalität als eine Reihe von SOA-Diensten; in [STS+04] wird SAP Netweaver als Beispiel für ein ERP-System

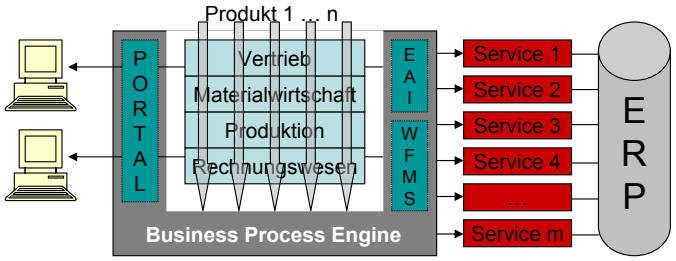

Abbildung 2.2: Geschäftssicht auf eine SOA nach [STS+04]

beschrieben. Die einzelnen Fachabteilungen nutzen eine BPE, um auf die offerierten Dienste zuzugreifen.

Die Geschäftsprozesse eines Unternehmens können auf diese Weise flexibel auf das IT-System und seine Dienste abgebildet werden, die Nutzung des gemeinsamen ERP-Systems bleibt für die Nutzer transparent, sodass beispielsweise Aktualisierungen von unternehmensweiten ERP-Komponenten durchgeführt werden können, ohne dass die einzelnen Anwender direkt davon betroffen sind.

Historisch hat sich dieser Ansatz zur Realisierung von Geschäftsanwendungen schrittweise entwickelt: Zunächst etablierten sich in Unternehmen abteilungsspezifische, voneinander unabhängige Applikationen, beispielsweise zur Unterstützung der Buchhaltung oder Logistik. Datenaustausch zwischen diesen Systemen erfolgte zunächst nur punktuell und nicht standardisiert. Heute werden solche Fachanwendungen auf Basis von ERP-Systemen (wie beispielsweise SAP) integriert, wodurch eine standardisierte Kommunikation und einheitliche Datenhaltung ermöglicht wird. Durch steigende Anforderungen an die Unternehmens-IT und damit einhergehende höhere Komplexität der Gesamtapplikation erscheint der Integrationsansatz auf Basis eines monolithischen ERP-Systems allerdings zunehmend schwer wartbar, sodass eine Entkopplung der Funktionalität durch Bereitstellung von Dienst-Schnittstellen auf Basis der in Abbildung 2.2 dargestellten Gesamtarchitektur angestrebt wird.

[STS+04] trifft allerdings keine Aussagen zur Vorgehensweise bei der Überführung existierender Systeme in eine SOA. Offene Fragen betreffen hierbei beispielsweise den anzustrebenden Zuschnitt und die Granularität der resultierenden SOA-Dienste.

In [HVH06] wird eine Methodik zur Strukturierung von Anwendungen nach SOA-Prinzipien vorgestellt. Die Autoren beschreiben die dienstorientierte Architektur als ein Architekturmuster zur Modellierung unternehmensweiter *IT-Anwendungslandschaften*. Als IT-Anwendungslandschaft bezeichnet man die Gesamtheit der IT-Anwendungen einer Organisation, einschließlich der vorhandenen Datenflüsse und Schnittstellen zur Vernetzung einzelner Anwendungen. Das Idealmodell einer solchen IT-Anwendungslandschaft mitsamt Abhängigkeiten zwischen einzelnen Komponenten ist in Abbildung 2.3 dargestellt.

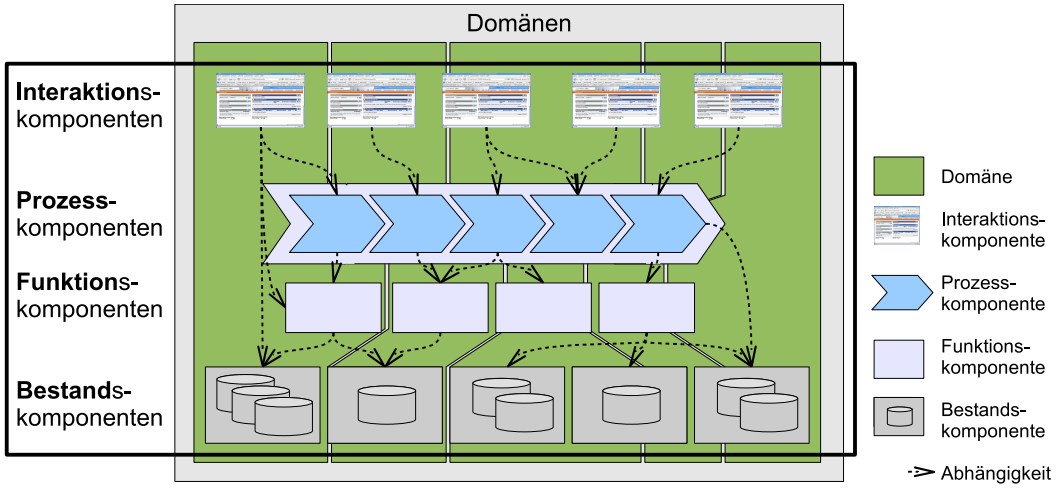

Abbildung 2.3: Referenzarchitektur einer IT-Anwendungslandschaft nach [HVH06]

In der Referenzarchitektur werden Bestands-, Funktions-, Prozess- und Interaktionskomponenten unterschieden. Bestandskomponenten verwalten den Datenbestand eines Unternehmens. Funktionskomponenten realisieren zuvor definierte Geschäftsfunktionen (und entsprechen damit SOA-Diensten). Prozesskomponenten implementieren die in der Architektur vorhandenen Geschäftsprozesse (und entsprechen damit formal beschriebenen Workflows). Interaktionskomponenten dienen der Interaktion mit den Benutzern des Systems.

Die Autoren definieren konkrete Leitlinien für den Zuschnitt von SOA-Diensten. Sie gehen dabei zum einen auf die inhaltliche Zuordnung einer Komponente zu den oben genannten Kategorien ein, zum anderen spezifizieren sie Richtlinien, nach denen bestehende Funktionen in grobgranulare, lose gekoppelte Dienste zerlegt werden können. [HVH06] geht allerdings nicht auf das weite Problemfeld der *SOA Governance* ein. SOA Governance befasst sich mit Entscheidungs-, Berichts- und Überwachungsprozessen rund um den Lebenszyklus von SOA-Komponenten und bildet damit die organisatorische Grundlage für die Verknüpfung der SOA und ihrer Entwicklung mit den Geschäftsinteressen eines Unternehmens.

2.4 Technische Sicht

Aus technischer Sicht handelt es sich bei einer SOA um ein geschichtetes System. Software-Komponenten, die Teilbereiche der Funktionalität einer Geschäftsapplikation realisieren, operieren auf den IT-Systemen eines Unternehmens. Diese Komponenten stellen jeweils einen Teil ihrer Funktionalität über Dienst-Schnittstellen bereit. Komplexe Dienste können hierbei Schnittstellen unterschiedlicher Basisdienste zusammenfassen. In der SOA werden Dienste zu Geschäftsprozessen (Workflows) orchestriert bzw. choreographiert. Ein Portal stellt die

Schnittstelle der Prozesse zur Kundenseite bereit. Abbildung 2.4 zeigt modellhaft den technischen Aufbau einer SOA.

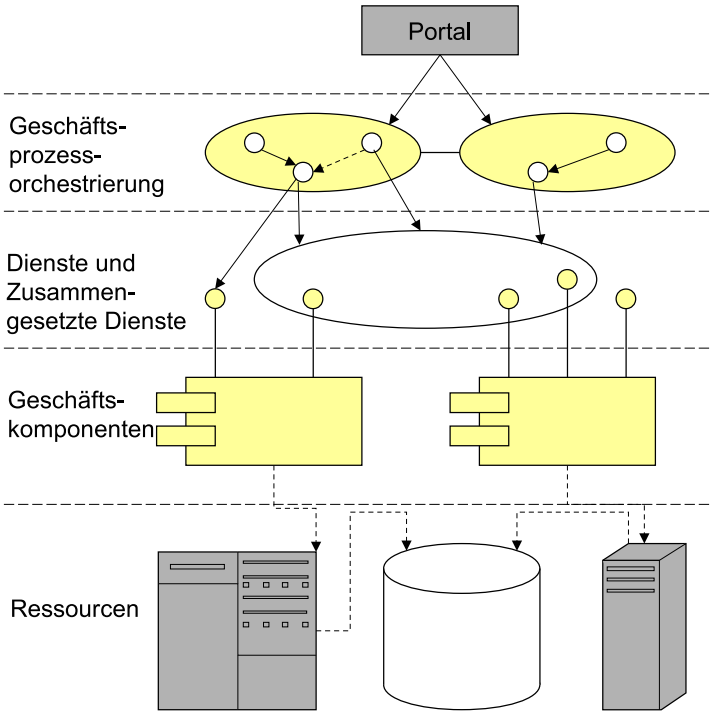

Abbildung 2.4: Technische Sicht auf eine dienstorientierte Architektur

Häufig eingesetzte Technologien zur Realisierung einer unternehmensweiten SOA sind Web Services, basierend auf der *Web Services Description Language (WSDL)* [W3C01] zur Schnittstellenbeschreibung, *SOAP* [W3C03] als Kommunikationsprotokoll, *Universal Description, Discovery and Integration (UDDI)* [OAS03] zum Service Lookup und der *Business Process Execution Language (WS-BPEL)* [OAS07] zur Beschreibung von Workflows. Hinweise zur Realisierung einer SOA mit diesen Technologien gibt [M+07].

Existierende Komponenten werden entweder mithilfe von so genannten *Wrapper Interfaces* in eine dienstorientierte Architektur eingebunden (z. B. um eine existierende Schnittstelle als Web Service verfügbar zu machen) oder über einen *Enterprise Service Bus (ESB)* [SHLP05, KAB+04] angesprochen, der zur Laufzeit zwischen verschiedenen Protokollen übersetzt und so Verbindungen zwischen Diensten mit unterschiedlichen Protokollanbindungen (z. B. Web Services und CORBA) realisieren kann.

In den folgenden Abschnitten wird detaillierter auf den Aufbau von SOA-Diensten und die Definition von SOA-Workflows eingegangen.

2.4.1 Dienste

[KBS05] beschreibt den technischen Aufbau eines Dienstes als eine aus den drei Teilen „Service Contract“, „Interfaces“ und „Implementation“ bestehende Software-Komponente (vgl. Abbildung 2.5).

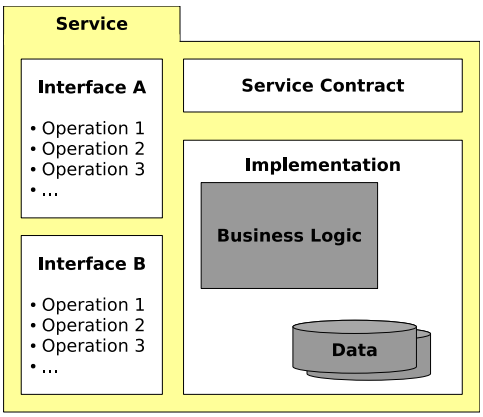

Abbildung 2.5: Eine Software-Komponente als Dienst nach [KBS05]

Der *Service Contract* definiert die angebotene Funktionalität sowie die Rahmenbedingungen, unter denen diese genutzt werden darf. Bestandteile des Service Contracts sind sowohl eine formale Schnittstellenbeschreibung (z. B. in Form eines WSDL-Dokuments) als auch Spezifikationen zu Sicherheits- oder Dienstgüteanforderungen. Er wird als eine Sammlung strukturierter Dokumente veröffentlicht. Der Service Contract beschreibt neben der Syntax der Dienstschnittstelle, was der Aufruf einer Methode inhaltlich bewirkt (z. B. welche Geschäftsfunktion durch einen Aufruf erbracht wird). Weiterhin ist im Service Contract definiert, über welche Protokolle die Dienstschnittstelle technisch erreichbar ist. Zusätzlich können im Service Contract Policies definiert werden, die sich z. B. auf Authentifizierungs- oder Verschlüsselungsanforderungen an die Kommunikation mit dem Dienst beziehen. Verfügbare Dienstgütegarantien werden hier ebenfalls bekannt gegeben. In der Beschreibung wird der Dienst konsequent aus einer äußeren Perspektive beschrieben, Implementierungsdetails sind nicht Bestandteil des Service Contracts, er beinhaltet aber alle Informationen, die ein Dritter zur Nutzung des Dienstes benötigt.

Mithilfe von *Interfaces* wird die eigentliche Funktionalität der Software-Komponente von außen zugänglich gemacht. Im Kontext dienstorientierter Architekturen stellt ein Interface die in Abschnitt 2.2 beschriebenen Geschäftsfunktionen bereit. Eine Software-Komponente kann dabei ihre Funktionalität in Form mehrerer unterschiedlicher Interfaces bereitstellen und so beispielsweise die inhaltliche Gruppierung von Geschäftsfunktionen erleichtern.

Die eigentliche *Implementierung* der Geschäftsfunktionen wird durch die Komponente gekapselt und ist damit für den Nutzer transparent. Sie umfasst sowohl die Geschäftslogik zur

Realisierung als auch die zugehörige Datenhaltung, wobei man diese bei der Umsetzung komplexerer Systeme in der Regel in unterschiedliche Stränge separiert.

2.4.2 Workflows

Im SOA-Kontext werden Prozesse in einer ausführbaren Workflow-Beschreibungssprache formuliert und zur Laufzeit durch ein WfMS interpretiert. Dabei koordiniert und überwacht das WfMS das Fortschreiten des Workflows und regelt die Interaktion zwischen den beteiligten Systemen.

Im nächsten Abschnitt wird auf mögliche Perspektiven bei der Modellierung von Workflows eingegangen. In Abschnitt 2.5 wird exemplarisch eine Auswahl von Workflow-Beschreibungssprachen vorgestellt.

2.4.2.1 Modellierung aus unterschiedlichen Perspektiven

Workflows können aus unterschiedlichen Sichtweisen definiert und beschrieben werden (vgl. [JB96]): Man unterscheidet allgemein Kontrollfluss-, Daten-, Ressourcen- und Betriebsperspektive.

Die *Kontrollfluss*- bzw. *Prozess*-Perspektive beschreibt Aktivitäten und die Reihenfolge ihrer Ausführung durch unterschiedliche Konstrukte wie beispielsweise Sequenzen, Fallunterscheidungen, Schleifen, Parallelisierung und Synchronisation, die im Folgenden genauer dargestellt sind.

- *Sequenzielle Ausführung:* Bei der sequenziellen Ausführung eines Workflows werden einzelne Teilschritte nacheinander abgearbeitet. Hierbei werden in den einzelnen Teilschritten in der Regel Ergebnisse aus früheren Teilschritten weiterverarbeitet.
- *Verzweigung:* Bei Verzweigungen in Workflows wird aufgrund bisheriger Ergebnisse oder anderer Informationen einer von mehreren alternativen Bearbeitungswegen ausgewählt und durchlaufen. Gegebenenfalls münden diese Bearbeitungswege später wieder in einen gemeinsamen Pfad.
- *Schleife:* Schleifen sorgen für die wiederholte Ausführung bestimmter Abschnitte eines Workflows. Die Anzahl der Schleifendurchläufe wird mithilfe einer Abbruchbedingung kontrolliert.
- *Parallele Ausführung:* Bei der parallelen Abarbeitung eines Workflows werden einzelne Teilschritte gleichzeitig abgearbeitet. Am Ende einer solchen Phase werden in der Regel die Ergebnisse der einzelnen Teilschritte an einem Synchronisationspunkt zusammengefasst, d.h. der Workflow fährt erst dann fort, wenn alle zuvor auszuführenden Operationen abgeschlossen wurden (`fork`/`join`).

Komplexere Abläufe werden mit diesen Elementen als Zusammensetzung der unterschiedlichen atomaren Basisaktivitäten modelliert.

Betrachtet man einen Workflow aus der *Daten*-Perspektive, unterscheidet man prinzipiell Geschäftsdaten und Workflow-interne Daten. Hierbei bilden Dokumente und andere Geschäftsdaten, die kommuniziert werden, zusammen mit prozessinternen Variablen effektiv Vor- und Nachbedingungen für die Ausführung von Aktivitäten.

In der *Ressourcen*-Perspektive wird die organisatorische Struktur eines Workflows in Form von Rollen menschlicher Akteure und beteiligter Systeme modelliert.

Die *Betriebs*-Perspektive beschreibt die Abbildung von Workflow-Aktivitäten auf Aktionen innerhalb der am Workflow beteiligten Systeme und Applikationen.

Derzeit existiert eine große Anzahl unterschiedlicher Workflow-Beschreibungssprachen, die schwerpunktmäßig jeweils eine oder mehrere der oben beschriebenen Workflow-Sichten unterstützen. Im Folgenden werden beispielhaft drei Sprachen zur Workflow-Beschreibung vorgestellt, die in letzter Zeit an Bedeutung gewonnen haben.

2.5 Ansätze zur Workflow-Beschreibung

Aufgrund der steigenden Bedeutung von Workflow-basierten Ansätzen bei der Realisierung von Geschäftsanwendungen wurde 1993 die *Workflow Management Coalition (WfMC)*[1] gegründet. Die WfMC arbeitet aktiv an der Standardisierung von Workflow-Beschreibungssprachen und bemüht sich um Interoperabilität zwischen Ausführungsplattformen und Entwicklungswerkzeugen.

2.5.1 Business Process Management Notation

Die *Business Process Management Notation (BPMN)* [OMG07] ist ein OMG-Standard zur Beschreibung von Workflows, der sich in erster Linie an Anwender richtet, die Geschäftsprozesse definieren und überwachen. Aus in BPMN beschriebenen Workflows können durch Transformation wiederum Beschreibungen in automatisch interpretierbaren Beschreibungssprachen gewonnen werden. Somit nimmt BPMN eine Brückenfunktion in der Kommunikation zwischen Anwendern und Technikern ein.

BPMN spezifiziert eine Reihe von Objekten, die in *Flows*, *Connecting Objects*, *Swimlanes* und *Artifacts* unterteilt werden können.

[1] Details unter `http://www.wfmc.org`

Geschäftsprozesse werden als Flows modelliert. Sie bestehen aus *Events*, *Activities* und *Gateways*. Beispiele für BPMN Events sind Start und Ende eines Prozesses. Die eigentlichen Arbeitsschritte werden durch Activities modelliert, diese stehen entweder für elementare Operationen (z. B. `send` oder `receive` von Nachrichten) oder repräsentieren einen vollständigen Unterprozess. Als Gateways werden alle Arten von Verzweigungen (`decision`, `fork`, `join`) im Prozess modelliert.

Mithilfe von Connecting Objects werden die Elemente eines Flows verknüpft. Verknüpfungen können unterschiedlicher Natur sein, hierunter fallen sowohl Zustandsübergänge als auch Assoziationen zwischen Datenelementen und Activities.

Ein BPMN-Beispielprozess mit mehreren Swimlanes ist in Abbildung 2.6 dargestellt. Die Abbildung zeigt die Abwicklung eines Bestellvorgangs bei einem Anbieter von Waren. Swimlanes dienen der Strukturierung von Prozessen. Sie werden in *Pools* und *Lanes* unterschieden und gruppieren logisch zusammenhängende Prozessteile. Dabei repräsentiert ein Pool jeweils einen *Participant* eines Prozesses, dessen Aktivitäten durch Lanes innerhalb des Pools weiter strukturiert werden können. In der grafischen Repräsentierung eines BPMN-Prozesses durchzieht eine Lane den jeweiligen Pool von einem Ende zum anderen.

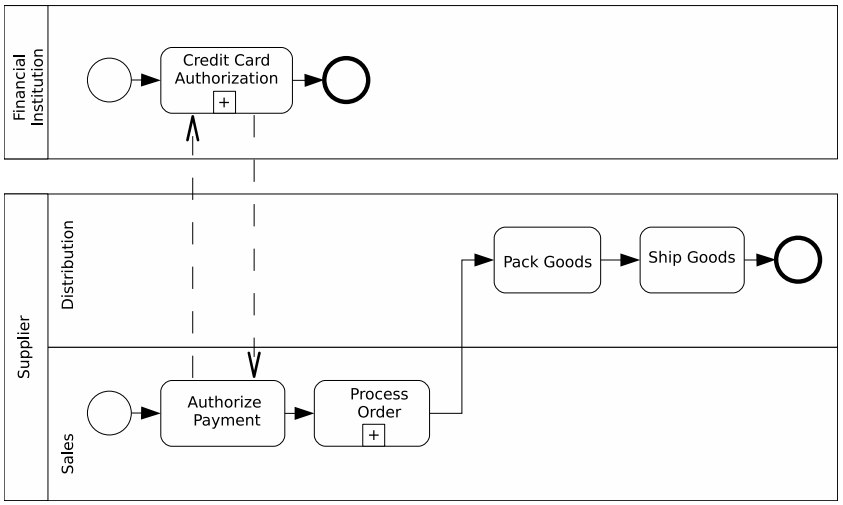

Abbildung 2.6: Beispiel eines BPMN-Prozesses (aus [OMG07])

Der in Abbildung 2.6 dargestellte Geschäftsprozess ist zunächst in zwei Pools unterteilt, die die Aktivitäten der teilnehmenden Organisationen („Financial Institution“ und „Supplier“) kapseln. Die Abwicklungsschritte des Suppliers werden durch Lanes den Abteilungen „Sales“ und „Distribution“ zugeordnet.

Für BPMN existiert keine fest definierte XML-Repräsentierung. Ursprünglich wurde die durch WfMS interpretierbare *Business Process Modelling Language (BPML)* als Standard-Transformationsziel von BPMN favorisiert. Aufgrund der Popularität von WS-BPEL wird BPML aber

inzwischen nicht mehr weiterentwickelt, sodass mittlerweile WS-BPEL das Standardziel für durch WfMS ausführbare Transformationen von in BPMN notierten Workflows geworden ist. Auf WS-BPEL wird im folgenden Abschnitt genauer eingegangen.

2.5.2 Business Process Execution Language

Die Business Process Execution Language [OAS07] ist eine XML-basierte Workflow-Beschreibungssprache, die zur Orchestrierung von über das SOAP-Protokoll [W3C03] ansprechbaren Web Services verwendet wird. Sie nimmt in Bezug auf die Modellierung von Workflows primär eine Kontrollfluss-Perspektive ein. Im Gegensatz zu BPMN ist WS-BPEL stark auf die Modellierung von IT-Workflows zur Verknüpfung von Web Services ausgerichtet und adressiert eher Entwickler als Anwender.

WS-BPEL wurde (unter dem Namen BPEL4WS) in einer ersten Version im Jahr 2002 von IBM, Microsoft und BEA vorgestellt und wird mittlerweile durch OASIS[2] weiterentwickelt. Im Jahr 2007 wurde Version 2.0 der Spezifikation verabschiedet.

BASIC ACTIVITIES

`assign`	Verändern des Inhalts einer Variablen
`invoke`	Synchroner oder asynchroner Aufruf eines Web Services
`receive / reply`	Anbieten einer synchron oder asynchron aufrufbaren Web Service Schnittstelle
`throw`	Explizites Signalisieren eines Fehlers
`wait`	Warten (auf einen Zeitpunkt oder für eine bestimmte Zeitspanne)
`exit`	Sofortiges Beenden eines Prozesses
`empty`	Platzhalter-Aktivität ohne Funktion

Tabelle 2.1: Übersicht über Basic Activities in WS-BPEL 2.0

Die Sprache definiert eine Reihe so genannter *BPEL Activities*. Diese repräsentieren einzelne Ausführungsschritte eines Workflows wie z. B. synchrone oder asynchrone Aufrufe an Web Services, Variablenzuweisungen, Auswertung von Ausdrücken, Fallunterscheidungen, Schleifen etc. Hierbei wird zwischen so genannten *Basic Activities* (vgl. Tabelle 2.1) und *Structured Activities* (vgl. Tabelle 2.2) unterschieden. Bei Basic Activities handelt es sich aus Sicht des Workflows um atomare Operationen, in Kombination mit Structured Activities können komplexe Kontrollstrukturen aufgebaut werden.

[2] `http://www.oasis-open.org`

STRUCTURED ACTIVITIES	
`sequence`	Sequenzielle Abarbeitung von Activities
`while/ repeatUntil/ forEach`	Wiederholte Ausführung von Activities
`if / elseif / else`	Bedingte Ausführung von Activities
`flow`	Parallele Abarbeitung von Activities
`pick`	Auswahl durch externe Ereignisse
`scope`	Kapselung untergeordneter Strukturen

Tabelle 2.2: Übersicht über Structured Activities in WS-BPEL 2.0

Gegenüber der Vorgängerversion wurden in WS-BPEL 2.0 einige Elemente umbenannt und die Sprache wurde um mehrere Activities erweitert. So wurde beispielsweise eine Activity zur Initialisierung von Prozessvariablen eingeführt und die Möglichkeit geschaffen, Variableninhalte mittels XSLT zu transformieren. Damit ist WS-BPEL 2.0 allerdings inkompatibel zu älteren BPEL-Versionen.

WS-BPEL erweitert die WSDL-Beschreibung eines Web Services um das Konzept der *PartnerLinks*. Durch Definition eines PartnerLinks erfolgt eine Verknüpfung zwischen dem BPEL-Prozess und einem im WSDL-Dokument spezifizierten *Service Port*. Der auf diese Weise spezifizierte Dienst kann dann über den symbolischen PartnerLink-Namen im Prozess adressiert werden. Eine detaillierte Analyse der sprachlichen Konstrukte von BPEL findet sich in [WvdADtH03], einen detaillierten Einblick in die Anwendung gibt [Mar07].

WS-BPEL Workflows werden durch *BPEL Engines* genannte WfMSs interpretiert und ausgeführt. Ein in Ausführung befindlicher WS-BPEL Workflow ist selbst wiederum über eine als WSDL-Dokument beschriebene SOAP-Schnittstelle als Web Service ansprechbar und kann so z. B. transparent in andere WS-BPEL Workflows integriert werden.

WS-BPEL definiert keine eigene grafische Notation sondern beschränkt sich auf die Spezifikation des WS-BPEL XML-Formats. Um das Erstellen von WS-BPEL-Prozessen zu erleichtern, existiert allerdings eine Reihe grafischer Editoren mit proprietärer Syntax.

2.5.3 XML Process Definition Language

Die *XML Process Definition Language (XPDL)* [WFM05] ist eine Sprache zur Beschreibung und zum Austausch von Workflow-Diagrammen. Der Schwerpunkt von XPDL liegt nicht auf der Modellierung ausführbarer Prozesse, sondern auf der grafischen Beschreibung von Geschäftsprozessen. Hierzu bietet die Sprache Elemente zur Kodierung von Vektor-Koordinaten,

Linien und Punkten, die der Visualisierung von Abläufen in Workflows dienen. Dadurch ist XPDL in der Lage, einen in BPMN beschriebenen Workflow verlustfrei abzulegen und wird derzeit de-facto als Standardformat zur Serialisierung von BPMN verwendet.

XPDL unterscheidet manuelle und automatisierte Aktivitäten und erlaubt so die Einbeziehung von durch Menschen durchzuführende Handlungen in Geschäftsprozesse. Im Unterschied zu BPEL werden allerdings keine Konstrukte zur Fehlerbehandlung und keine Transaktionen unterstützt, sodass XPDL in der Praxis zur Modellierung von direkt ausführbaren Geschäftsprozessen, die unterschiedliche Systeme und Partner einbeziehen, nur eingeschränkt nutzbar ist.

Einen ausführlichen technischen Vergleich zwischen BPML, WS-BPEL und XPDL liefert [Sha03].

2.6 Service Component Architecture

Die *Service Component Architecture (SCA)* ist ein Ansatz zur Vereinheitlichung der Sicht auf Komponenten sowie zur Strukturierung von Komponenten und ihrem Zusammenspiel innerhalb einer SOA. SCA verfolgt damit ähnliche Ziele wie die *Windows Communication Foundation (WCF)* [Cha05] von Microsoft (die sich explizit auf .Net-Applikationen beschränkt) und die *Java Business Integration (JBI)*-Spezifikation [SUN05] (die Java-basierte Web Services betrachtet). Im Gegensatz zu WCF und JBI ist SCA aber implementierungsunabhängig ausgerichtet.

Die derzeit vorliegende Version 1.0 der modular aufgebauten SCA-Spezifikation wurde von der *Open SOA Collaboration*[3] entwickelt, einem Zusammenschluss mehrerer im SOA-Umfeld aktiver Unternehmen (u.a. IBM, Sun, Oracle, SAP und BEA). Der weitere Standardisierungsprozess ist in der Zwischenzeit an OASIS übergeben worden.

Zentraler Teil der Spezifikation ist das SCA Assembly Model [OSO07b]. Es definiert die Struktur einer SCA-Anwendung als Komposition unterschiedlicher Architekturelemente, so genannter Komponenten. Zusätzlich werden im Assembly Model auch die Kommunikation betreffende Aspekte wie z. B. Verschlüsselung, Authentifizierung oder Transaktionsverhalten zwischen Diensten und SCA-Anwendungen definiert.

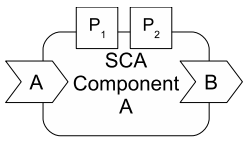

Abbildung 2.7: Aufbau einer SCA-Komponente nach [OSO07b]

[3] Details unter `http://www.osoa.org`

SCA definiert eine Komponente (*SCA Component*) als ein Element mit beliebig vielen Schnittstellen (*Services*) und Abhängigkeiten (*References*). Ein Service kann durch andere SCA Components referenziert werden, dies wird dann als *Binding* bezeichnet. Neben Services und References definiert SCA noch *Properties*, auf die zur Laufzeit durch andere Komponenten zugegriffen werden kann. Abbildung 2.7 zeigt die grafische Repräsentation einer SCA Component. Im Bild erkennt man, dass die SCA Component einen Service `A` offeriert und einen Service `B` referenziert. Weiterhin verfügt die dargestellte Komponente über zwei Properties P_1 und P_2. Referenzen auf andere Services können mit Attributen belegt werden, die die Art der Referenz näher kennzeichnen, beispielsweise `OneWay` für einen Aufruf ohne Rückgabewert oder `Callback` für bidirektionale, asynchrone Kommunikation über ein Callback Interface.

SCA erlaubt die Gruppierung mehrerer SCA Components mitsamt ihren Bindings zu einem *SCA Composite*, das sich nach außen wiederum als eine SCA Component darstellt und wie diese über Services, Properties und References verfügt. Ein Beispiel für eine solche Komposition ist in Abbildung 2.8 dargestellt. Das Beispiel zeigt ein Composite aus drei SCA Components, das Composite stellt einen Service `A` bereit und referenziert einen Service `D`. Intern wird das Interface `A` auf den durch die Komponente `A` bereitgestellten Service propagiert, der wiederum die Services `B` und `C` referenziert, die jeweils von einer Komponente `B` bzw. `C` bereitgestellt werden. Im Beispiel hat die Komponente `B` eine Abhängigkeit zu einer Datenbank, die nicht durch SCA abgebildet wird. Die Referenz auf `D` ergibt sich durch Übertragung der in der Komponente `C` definierten Referenz auf das umschließende Composite.

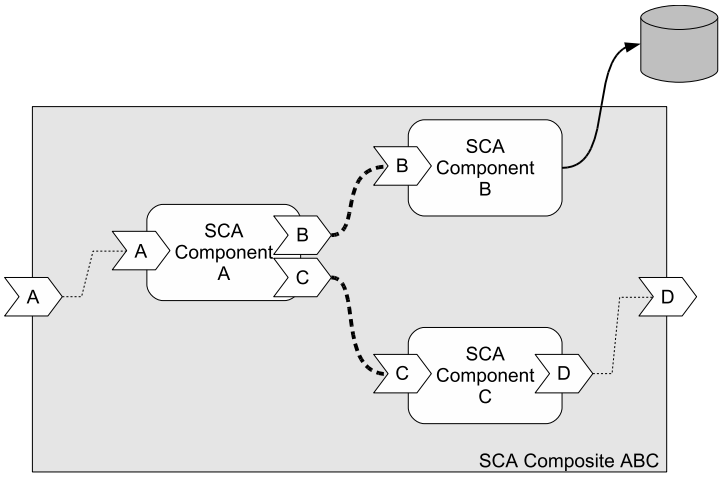

Abbildung 2.8: Beispiel für ein SCA Composite

Zur Unterstützung administrativer Prozesse und Zuständigkeiten definiert SCA das Konzept der *SCA Domain*. Die Spezifikation geht davon aus, dass innerhalb einer SCA Domain Komponenten und Composites unter Verwendung der SCA Runtime lediglich eines einzelnen Herstellers bereitgestellt werden. Damit wird es beispielsweise einem Hersteller ermöglicht, proprietäre Binding-Protokolle zwischen Komponenten zu implementieren. Die aktuelle SCA-

Spezifikation sieht nicht vor, dass Composites sich über Domain-Grenzen hinweg erstrecken können, die Kommunikation zwischen SCA Components unterschiedlicher Domains ist jedoch möglich.

SCA definiert ein *Policy Framework* [OSO07a] zur Spezifikation von Interaktions- und Implementierungsvorgaben für SCA Components innerhalb einer SCA Domain. Derzeit beschreibt SCA exemplarisch Policies zur Definition von Eigenschaften eines Kommunikationskanals (Zuverlässigkeit, Verschlüsselung) und für Authentifizierung. Da Policies momentan lediglich auf Domain-Ebene spezifiziert werden können, schreibt SCA keine einheitliche Policy-Beschreibungssprache vor, die Verwendung von WS-Policy [W3C07] wird aber empfohlen.

Die Spezifikation sieht in Version 1.0 bereits mehrere unterschiedliche Möglichkeiten zur Implementierung einer SCA Component, so genannte *Implementation Bindings*, vor, darunter neben reinen Java- und C++-Bindings auch BPEL, Enterprise JavaBeans (EJB), Java Messaging Service (JMS) und Spring. Mithilfe einer *Service Component Description Language (SCDL)*-Datei kann ein Binding dann zur Laufzeit entsprechenden Code als SCA Komponente bereitstellen oder ggf. auch erst bei Bedarf instanziieren.

3 IT-Management

Als IT-Management bezeichnet man die Überwachung und Steuerung des Verhaltens von IT-Systemen mit dem Ziel des Erreichens oder Erhaltens eines geforderten Systemzustands. Allgemein werden IT-Management-Aufgaben in die Bereiche Überwachung, Analyse und Steuerung unterteilt.

Um ein System effizient steuern zu können, müssen im Rahmen der Überwachung (Monitoring) zunächst aktuelle Informationen über das System beschafft werden. Diese bilden die Basis für eine Analyse (Analysis) des für das Management relevanten Systemzustands. Weicht dieser vom Sollzustand ab, kann der Systemzustand mithilfe entsprechender Steuerungsoperationen (Control) korrigiert werden. In großen Umgebungen wird der Administrator durch IT-Management-Software bei der Durchführung von Management-Aufgaben unterstützt.

Für diese Arbeit ist insbesondere der Teilbereich des IT-Managements relevant, der sich mit der durch ein System erbrachten Dienstgüte befasst. Weiterhin sind konkrete Vorgehensmodelle für das IT Management von SOA-Umgebungen von Bedeutung.

Im Folgenden wird zunächst kurz auf unterschiedliche Klassifikationsansätze für IT-Management eingegangen. Im Anschluss wird das Management von Dienstgüteaspekten eines Systems genauer betrachtet. Exemplarisch werden konkrete Technologien vorgestellt, die jeweils einen Teilaspekt des Dienstgüte-Managements abdecken.

Zum Abschluss des Kapitels wird ein Überblick über aktuelle Forschungsansätze zur Automatisierung von IT-Management, dem sog. Selbst-Management, gegeben.

3.1 Klassifikationsansätze für IT-Management

Eine häufig in der Praxis angewandte Möglichkeit zur Klassifikation zu managender Systeme ist die Einordnung nach System-Schichten (vgl. [HAN99]): Netzwerk-Management, System-Management, Middleware-Management und Dienste-Management. Typischerweise sind in größeren Unternehmen unterschiedliche Abteilungen mit dem Management von Systemen

aus jeweils einer dieser Schichten betraut, d. h. Netzwerk-Management und System-Management sind z. B. in der Regel organisatorisch getrennt. Zur Einordnung von Dienstgüte-Management eignet sich die Kategorisierung nach Systemschichten jedoch nicht, da Dienstgüteaspekte in jeder Schichte von Interesse sind. Für höhere Schichten können keine verlässlichen Dienstgütegarantien gegeben werden, wenn die darunterliegenden Schichten diese nicht unterstützen.

Als Teil der Open Systems Interconnection (OSI) System Management Spezifikation [ISO92] definiert die International Standards Organization (ISO) fünf Funktionsbereiche des IT-Managements:

- Fehler-Management (Fault Management)
- Konfigurations-Management (Configuration Management)
- Abrechnungs-Management (Accounting Management)
- Leistungs-Management (Performance Management)
- Sicherheits-Management (Security Management)

Diese werden aufgrund der Anfangsbuchstaben ihrer englischen Bezeichnungen zusammenfassend als *FCAPS* bezeichnet. Die Management-Funktionsbereiche des OSI-Managements sind nicht vollständig unabhängig voneinander; so führen Konfigurationsänderungen häufig auch zu Änderungen in der Leistungsfähigkeit eines Systems usw.

Bezogen auf die FCAPS-Einteilung der ISO lässt sich das Dienstgüte-Management am ehesten in der Kategorie Leistungsmanagement einordnen:

Nach [ISO92] befasst sich das Leistungsmanagement mit der Analyse und Optimierung der Systemleistung sowie der Minimierung des Ressourcenverbrauchs eines Systems. Ziel ist dabei die Identifikation von Leistungsengpässen und das Sicherstellen des Einhaltens vereinbarter Leistungsparameter.

Das OSI-Modell beschränkt sich allerdings auf eine technische Sicht und berücksichtigt keine organisatorischen Aspekte (z. B. die Einbindung des IT-Managements in Organisationsprozesse, Zuständigkeitsdomänen etc.).

Die britische Central Computer and Telecommunication Agency (CCTA) definierte seit 1989 die *IT Infrastructure Library (ITIL)*, die organisatorische Prozesse des IT-Managements in Form von „Best Practices“ beschreibt. Zwischen 1999 und 2003 wurde von der CCTA-Nachfolgebehörde Office of Government Commerce (OGC) Version 2 der ITIL-Spezifikation herausgegeben, die im Juni 2007 durch ITIL V3 abgelöst wurde.

ITIL V3 [Tay07] betrachtet den vollständigen Lebenszyklus eines IT-Services aus Managementsicht in Form von fünf Büchern, die jeweils eine Phase des IT-Service-Lebenszykluses beschreiben. Diese sind Servicestrategie [IN07], Serviceentwurf [RL07], Serviceüberführung [LM07], Servicebetrieb [CW07] und kontinuierliche Serviceverbesserung [Spa07].

Dienstgüteaspekte finden primär im Bereich Servicebetrieb und teilweise im Bereich kontinuierliche Serviceverbesserung Beachtung: Der Servicebetrieb betrachtet die Produktivphase eines Services, hierzu gehören alle Prozesse, die im Kern zur Unterstützung des Operativbetriebs notwendig sind, sowie der Umgang mit Dienstgüteanforderungen und entsprechenden Verträgen.

Die bisher betrachteten Ansätze zur Kategorisierung sind unabhängig von einer konkreten Anwendungsdomäne. Fokussiert man auf Dienstgüteaspekte von SOA-Anwendungen, so wird man bei der OASIS Reference Architecture for Service Oriented Architecture [OAS08] fündig: Hier wird im Rahmen der Beschreibung des Service Contracts als Bestandteil eines SOA-Dienstes (vgl. Abschnitt 2.4.1) auf die Modellierung von Dienstgüte eingegangen.

Weiterhin beschreibt [OAS08] Ansätze zum Management von Diensten: In der Referenzarchitektur wird vorgeschlagen, das Management von Diensten wiederum als SOA-Dienst zu modellieren. Hierbei wird angenommen, dass die einzelnen Aspekte eines Dienstes durch unterschiedliche Management-Komponenten verwaltet werden können. Im Einzelnen werden die folgenden Management-Bereiche für einen Dienst identifiziert:

- *Lebenszyklus (Life-cycle manageability)*
  Management des Lebenszykluses eines Dienstes: Erzeugung und Terminierung von Instanzen, Verwaltung von Abhängigkeiten
- *Konfiguration (Configuration manageability)*
  Verwaltung der Konfiguration eines Dienstes
- *Ereignisüberwachung (Event monitoring manageability)*
  Management der Überwachung von Ereignissen und der Information über aufgetretene Fehler
- *Accounting (Accounting manageability)*
  Verwaltung der Accounting-Funktionalität, beinhaltet neben der Erfassung der Dienstnutzung auch die Erfassung der Benutzer und die Zuordnung von Gebühren zu einzelnen Benutzern
- *Dienstgüte (Quality of service manageability)*
  Management der Dienstgütegarantien eines Dienstes, hierzu gehören klassischerweise Bandbreite, Antwortzeiten und Verfügbarkeit
- *Business-Performance (Business performance manageability)*
  Management von Aspekten eines Dienstes, die über die erreichte Business-Performance Auskunft geben, hierzu zählen beispielsweise SLAs, die sich auf abstrakte, geschäftsorientierte Metriken beziehen
- *Policies (Policy manageability)*
  Management von Policies innerhalb eines Dienstes, hierzu gehören das Etablieren neuer und das Deaktivieren bestehender Policies sowie die Verwaltung von Policy Decision und Policy Enforcement Points

Die Umsetzung eines effektiven Managements für einige der genannten Bereiche ist sehr komplex, teilweise überlappen diese inhaltlich oder nutzen Management-Funktionen anderer Bereiche bei der Umsetzung. Beispielsweise kann das Dienstgüte-Management bei der Umsetzung entsprechender Vorgaben Gebrauch von Funktionen des Policy-Managements machen. [OAS08] gibt jedoch keine Hinweise zur Implementierung der skizzierten Management-Bereiche.

Der folgende Abschnitt widmet sich der detaillierten Betrachtung von Dienstgüte-Management und Dienstgüteeigenschaften.

3.2 Dienstgüte-Management

3.2.1 Überblick

Zur effektiven Überwachung der Dienstgüte geschäftskritischer IT-Prozesse und Anwendungen bedarf es der Etablierung geeigneter Kontrollmechanismen im Rahmen des IT-Managements. Bei Geschäftsanwendungen werden in der Regel formale Vereinbarungen über die durch ein System zu erbringende Dienstgüte geschlossen, so genannte *Service Level Agreements* (SLAs) [Lew99, Ver99, SMJ00]. Die Anwendung von Mechanismen zur Überwachung und Durchsetzung vereinbarter Dienstgütekriterien bezeichnet man in diesem Kontext als *Dienstgüte-Management (Service Level Management, SLM)*.

ITIL beschreibt die Ziele des Service Level Managements wie folgt:

> *The goal for Service Level Management is to maintain and improve IT Service quality, through a constant cycle of agreeing, monitoring and reporting upon IT Service achievements and instigation of actions to eradicate poor service – in line with business or cost justification. Through these methods a better relationship between IT and its Customers can be developed* [OGC01, Seite 27].

ITIL bleibt bei der Beschreibung von Anforderungen sehr abstrakt, sodass sich aus den zur Verfügung gestellten Dokumenten nur allgemeine Empfehlungen zur Vorgehensweise aber keine konkreten Vorgaben für Unternehmen ableiten lassen. Die ISO veröffentlichte im Jahr 2005 den Standard ISO/IEC 20.000 [ISO05], der sich mit der Definition von Prozessen im Bereich des IT Service Managements beschäftigt und ITIL V3 in diesem Punkt ergänzt. Der ISO-Standard beschreibt nachprüfbare Service-Management-Anforderungen an IT-Organisationen, ITIL zeigt Wege auf, wie diese Anforderungen erreicht werden können. Eine Einführung in die Anwendung der durch ISO 20.0000 spezifizierten Prozesse gibt [DSZS09].

Technische Grundlage für die Realisierung von SLM bildet eine Management-Infrastruktur, die auf Ebene der zu verwaltenden Komponenten geeignete Sensorik- und Aktorikschnittstellen definiert. Die Zustandsüberwachung einer unter Management stehenden Komponente erfolgt über Sensorikschnittstellen, ein regelnder Eingriff des Management-Systems erfolgt über entsprechende Aktorikschnittstellen. Um die Skalierbarkeit des Systems für große Umgebungen sicherzustellen, wird typischerweise ein Teil der Funktionalität des Management-Systems an sog. Management-Agenten delegiert. Diese kommunizieren mit den zu verwaltenden Systemen und aggregieren ggf. auch Informationen über mehrere Systeme (vgl. Abbildung 3.1).

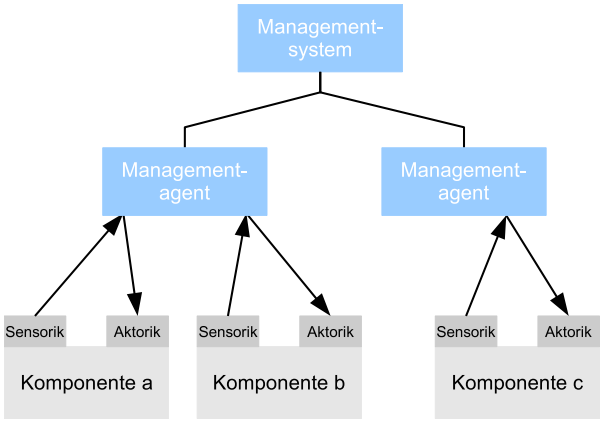

Abbildung 3.1: Management mehrerer Komponenten mithilfe von Sensorik- und Aktorikschnittstellen

Sensorik- und Aktorikschnittstellen zu verwaltender Komponenten sind auf unterschiedlichste Weise realisierbar: Im einfachsten Fall kann ein Management-System beispielsweise über Log- und Konfigurationsdateien mit der von ihm verwalteten Komponente kommunizieren, häufig erfolgt eine Kommunikation aber über speziell für Management-Zwecke bereitgehaltene Schnittstellen und Protokolle wie z. B. SNMP [Sta99], CIM/WBEM [DMT99], WSDM [OAS05a, OAS05b, OAS05c] oder Application Response Measurement (ARM) [TOG04]. Auf ARM wird in Abschnitt 3.2.3.2 genauer eingegangen.

Das Verwenden solcher Schnittstellen bietet einige Vorteile: Die Entwicklung von Management-Systemen und den von ihnen verwalteten Komponenten kann entkoppelt erfolgen, gleichzeitig erhöht sich die Effizienz eines Management-Systems, wenn damit unterschiedliche Typen von Komponenten über eine einheitliche Schnittstelle verwaltet werden können.

Die technischen Ursprünge der meisten der aktuell verwendeten Management-Systeme liegen im Netzwerk-Management. Daher findet man oft eine hierarchische Organisation solcher Systeme vor, die zur Laufzeit mit statischen Strukturen arbeitet. Als ranghöchstes Element wird der Administrator in das System eingebunden. Er trifft letztendlich auf Basis der Informationen aus der Sensorik die notwendigen Management-Entscheidungen, und veranlasst mithilfe der Aktorik entsprechende Konfigurationsänderungen.

Diese Vorgehensweise birgt im Hinblick auf die Komplexität und Dynamik moderner dienstorientierter Software-Architekturen eine Reihe von Problemen: Die Anzahl der Fehler bei Entscheidungen eines Administrators wächst mit steigender Komplexität des zu verwaltenden Systems, da Auswirkungen auf voneinander abhängige Systemkomponenten nicht mehr vollständig verstanden werden. Gleichzeitig können in großen Systemen Fehler fatale Auswirkungen haben. Um dieser Entwicklung entgegenzuwirken, wird derzeit an der Automatisierung von Management-Vorgängen gearbeitet. Entsprechende Ansätze werden in Abschnitt 3.3 vorgestellt.

3.2.2 Dienstgüteanforderungen

Unter dem Begriff Dienstgüteanforderungen werden allgemein nicht-funktionale Anforderungen an Dienste verstanden, aus denen sich jedoch oftmals Strukturierungsmerkmale für die Realisierung von Diensten ableiten lassen. Dienstgüteanforderungen betreffen beispielsweise Vorgaben für das Antwortzeitverhalten, für den von einem Dienst minimal zu bewältigenden Durchsatz, für die Verfügbarkeit eines Dienstes oder für den Umgang eines Dienstes mit übermittelten Informationen (Vertraulichkeit).

Unterschiedliche Typen von Dienstgüteparametern können auch als Dienstgütedimensionen dargestellt werden. Für ein konkretes Anwendungssystem existiert eine Beziehung zwischen dem (für das SLM relevanten) Systemzustand und der in einer bestimmten Dimension erreichten Dienstgüte. Der Systemzustand wird durch eine Menge von Attributen beschrieben. Pro Dienstgütedimension ist dabei jeweils nur eine Untermenge dieser Attribute relevant. Für von einander unabhängige Dienstgütedimensionen sind die relevanten Attributmengen überschneidungsfrei, in der Regel existieren jedoch Abhängigkeiten zwischen unterschiedlichen Dienstgütedimensionen.

Für einen Workflow ergibt sich die erbrachte Dienstgüte aus dem Zusammenspiel der am Workflow beteiligten Dienste und deren Dienstgüteeigenschaften. Unterschiedliche Dienstgütedimensionen unterscheiden sich darin, inwieweit und auf welche Weise die Struktur eines Workflows (und damit das Zusammenspiel der beteiligten Dienste) die insgesamt erbrachte Dienstgüte beeinflusst. Beispielsweise wird die Dienstgütedimension *verschlüsselte Datenhaltung* eines Workflows durch strukturelle Änderungen weniger tangiert als z. B. Anforderungen bezüglich der zu erwartenden Antwortzeit. Charakteristisch für Dienstgütedimensionen wie *verschlüsselte Datenhaltung* ist, dass ihre Einhaltung durch das schwächste Glied der Kette involvierter Komponenten definiert wird – und das weitgehend unabhängig von der Anordnung der Kettenglieder im Workflow.

Zur Ermittlung der durch einen Workflow erbrachten Dienstgüte müssen somit neben dessen Struktur sowohl die durch die beteiligten Komponenten erbrachte Dienstgüte als auch die zur Ermittlung der Gesamtdienstgüte notwendige Berechnungsvorschrift bekannt sein. Sollen

für einzelne Komponenten Dienstgüteziele auf Basis eines Gesamtzieles vorgegeben werden, muss die jeweilige Berechnungsvorschrift umgekehrt eingesetzt werden.

Im Folgenden wird die Dienstgüte-Dimension *Antwortzeit* genauer betrachtet.

3.2.2.1 Dienstgütedimension Antwortzeit

Die Antwortzeit eines geschachtelten Systems setzt sich aus mehreren Komponenten zusammen. Beispiele für derartige Systeme sind Workflows oder auch aus mehreren Einzeldiensten zusammengesetzte Dienste.

In [Jai91] werden zwei alternative Definitionen für die Antwortzeit eines Systems vorgestellt (vgl. Abbildung 3.2).

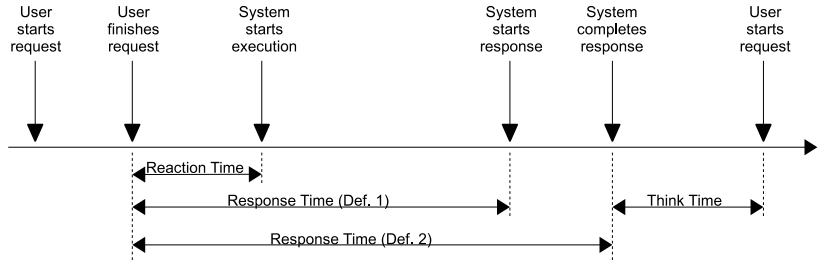

Abbildung 3.2: Antwortzeitmodell aus [Jai91]

Die beiden Modelle unterscheiden sich darin, ob sie die Antwortdauer selbst als Teil der Antwortzeit betrachten. *Definition 1* betrachtet die Zeitspanne vom Absenden eines Requests bis zum Beginn der Antwort des Systems als Antwortzeit, *Definition 2* legt das Ende der Antwort des Systems als Endzeitpunkt fest. Dem in dieser Arbeit verwendeten Antwortzeitbegriff liegt die zweite Definition zugrunde, da nur diese eine Ende-zu-Ende-Betrachtung der Antwortzeit aus Client-Sicht ermöglicht. Insbesondere beim (im SOA-Kontext regelmäßig auftretenden) Fall des verteilten Dienstaufrufs müssen durch die Netzübertragung bedingte Latenzen bei der Betrachtung der Antwortzeit eines Dienstes berücksichtigt werden, diese finden in der ersten Definition keine Beachtung.

Im Folgenden werden am Beispiel des Zusammenspiels zweier SOA-Dienste die Begriffe Antwortzeit und Bedienzeit – in Übereinstimmung mit den Definitionen aus [SMJ00] und [Jai91] – erläutert.

Abbildung 3.3 zeigt beispielhaft einen Dienst a, der im Rahmen seiner Ausführung einen zweiten Dienst b aufruft. Aus Sicht des Dienstes a besteht die Antwortzeit $t_{response}(b)$ des Dienstes b aus der Zeitdauer für die Übermittlung des Auftrages an b, der eigentlichen Bedienzeit des Dienstes b und der Zeitdauer für die Rückübermittlung des Ergebnisses:

$$t_{response}(b) = t_{invoke}(b) + t_{work}(b) + t_{reply}(b) \tag{3.1}$$

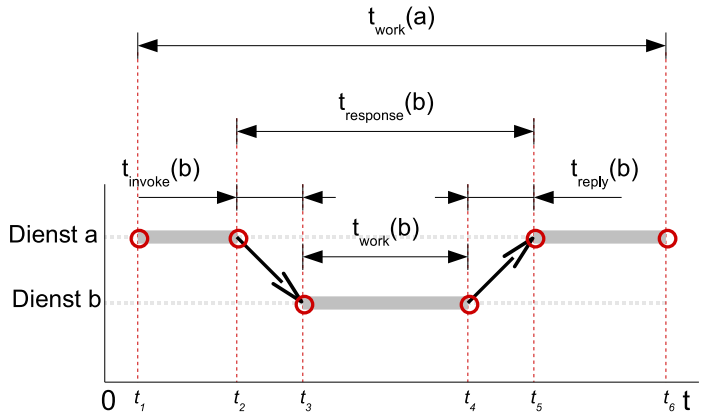

Abbildung 3.3: Antwortzeitmodell eines geschachtelten Dienstes

Definition 3.1: ANTWORTZEIT
Als Antwortzeit wird die Zeitspanne vom Absenden eines Auftrages an einen Dienst bis zum vollständigen Empfang der zugehörigen Antwort durch den Dienstnutzer bezeichnet.

Die einzelnen Antwortzeit-Komponenten können je nach Art des Auftrages und Übertragungsweg in ihrer Bedeutung für die Gesamtantwortzeit variieren.

In der in Abbildung 3.3 dargestellten Situation geht – bedingt durch die Schachtelung der Aufrufe – die Antwortzeit des Dienstes b in die Bedienzeit $t_{work}(a)$ des Dienstes a ein.

Definition 3.2: BEDIENZEIT
Die Bedienzeit eines Auftrags definiert die Zeitspanne vom Eintreffen des Auftrages eines Dienstnutzers bis zum Beginn der Rücksendung der Antwort an den Dienstnutzer.

Die Dauer der einzelnen Zeitkomponenten (vgl. Formel 3.1) kann zur Laufzeit durch entsprechende Messungen im System ermittelt werden. Die entsprechenden Messpunkte t_1 bis t_6 sind in Abbildung 3.3 durch Kreise dargestellt.

Messungen der Bedienzeit erfolgen lokal, im Beispiel ermittelt man die Bedienzeit t_{work} des Dienstes a wie folgt: $t_{work}(a) = t_6 - t_1$. Messungen der Antwortzeit erfolgen aus Sicht des jeweiligen Dienstnutzers. Im Beispiel ermittelt man die Antwortzeit $t_{response}$ des Dienstes b wie folgt: $t_{response}(b) = t_5 - t_2$. Entsprechend kann man auch $t_{invoke}(b)$ und $t_{reply}(b)$ ermitteln. Voraussetzung dafür sind aber eine Uhrensynchronisation innerhalb des Systems sowie die komponentenübergreifende Verfügbarkeit der relevanten Messwerte.

3.2.2.2 Definition von Antwortzeit-SLOs

Nimmt man an, dass unter den beteiligten Akteuren Konsens über zu betrachtende Zeiträume und zu verwendende Größeneinheiten herrscht, kann das einem Workflow P zugeordnete Antwortzeit-SLO als natürliche Zahl aufgefasst werden:

$$\mathrm{sloResponse}(P) \in \mathbb{N} \tag{3.2}$$

Ebenso werden SLOs für einzelne Aktivitäten $a_i \in A_P$ eines Workflows P festgelegt:

$$\mathrm{sloResponse}(a_i) \in \mathbb{N} \tag{3.3}$$

Betrachtet man $a_1 \dots a_j$ als Folge von Aktivitäten, die einen möglichen Pfad durch einen Workflow P definieren, so gilt für die Definition der Antwortzeit-SLOs der einzelnen Aktivitäten a_i die Bedingung:

$$\sum_{i=0}^{j} \mathrm{sloResponse}(a_i) \leq \mathrm{sloResponse}(P) \tag{3.4}$$

Für jede Aktivität $a_i \in A_P$ kann weiterhin ein Bedienzeit-SLO definiert werden:

$$\mathrm{sloWork}(a_i) \in \mathbb{N} \tag{3.5}$$

Aus Formel 3.1 ergibt sich für alle $a_i \in A : \mathrm{sloResponse}(a_i) \geq \mathrm{sloWork}(a_i)$. Nimmt man für $t_{invoke}(a)$ und $t_{reply}(a)$ konstante Werte an, so kann man den Maximalwert von $\mathrm{sloWork}(a)$ aus $\mathrm{sloResponse}(a)$ wie folgt herleiten:

$$\mathrm{sloWork}(a) = \mathrm{sloResponse}(a) - (t_{invoke}(a) + t_{reply}(a)) \tag{3.6}$$

3.2.3 Technologien aus dem Bereich des Dienstgüte-Managements

Im Folgenden werden beispielhaft zwei Technologien aus dem Bereich des Dienstgüte-Managements vorgestellt, die für die praktische Umsetzung des in dieser Arbeit beschriebenen Ansatzes relevant sind.

Zunächst wird in Abschnitt 3.2.3.1 auf WS-Agreement – eine Spezifikation zur Unterstützung der automatisierten Aushandlung von Dienstgütevereinbarungen zwischen Diensten – eingegangen. Im Anschluss wird in Abschnitt 3.2.3.2 mit dem Application Response Measurement beispielhaft eine Technologie zum Performance Monitoring in verteilten Anwendungssystemen vorgestellt.

3.2.3.1 WS-Agreement

Die erste Version der *Web Services Agreement Specification (WS-Agreement)* [ACD+07] wurde im März 2007 von der Grid Resource Allocation Agreement Protocol Working Group des Open Grid Forums spezifiziert. WS-Agreement spezifiziert ein Protokoll zur Aushandlung von SLAs zwischen Anbietern und Nutzern von Diensten. Bedingt durch ihre Herkunft aus dem Grid-Umfeld baut die Spezifikation dabei auf etablierten Web-Services-Protokollen auf. WS-Agreement definiert XML-Schemata zur Beschreibung von SLAs und anderen zur Kommunikation notwendigen Dokumenten sowie Operationen zur SLA-Aushandlung und zur Verwaltung des Lebenszykluses eines SLAs.

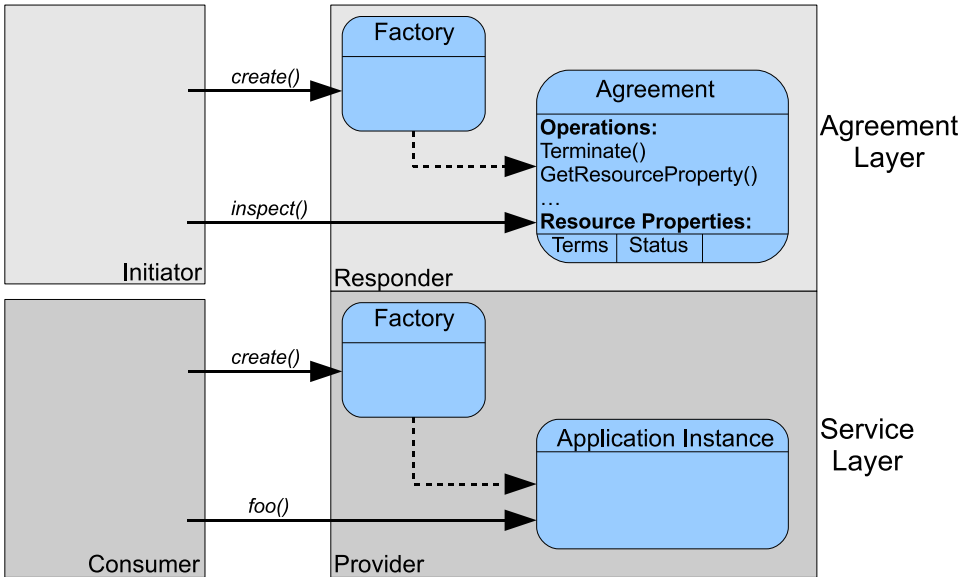

Abbildung 3.4: Konzeptuelles Schichtenmodell von WS-Agreement nach [ACD+07]

Wie in Abbildung 3.4 dargestellt, führt die Spezifikation eine zusätzliche Schicht, den so genannten *Agreement Layer* ein. Sowohl Diensteanbieter *(Provider)* als auch Nutzer *(Consumer)* müssen zur Nutzung von WS-Agreement entsprechende Schnittstellen des Agreement Layers implementieren. Bevor der Nutzer den durch den Diensteanbieter bereitgestellten Dienst nutzen kann, muss zunächst auf Ebene des Service Layers ein SLA über die Dienstnutzung geschlossen werden.

Im einfachsten Falle erfolgt das Schließen eines SLAs über den Austausch von zwei XML-Dokumenten. Zunächst sendet der sog. *Initiator* (i. d. R. der Consumer) ein WS-Agreement *Offer*-Dokument an den *Responder* (i. d. R. den Provider). Dieses Offer-Dokument enthält die Bedingungen, zu denen der Initiator gewillt ist, ein SLA über die Nutzung eines Dienstes mit dem Responder zu vereinbaren. Ist der Responder mit den Bedingungen des Offers einverstanden, antwortet er, indem er ein *Agreement*-Dokument zurücksendet, das den Bedingungen des Offers genügt. Die Ablehnung eines Offers wird durch eine Fehlerantwort ausgedrückt.

In der Praxis orientiert sich der Initiator i. d. R. beim Erstellen des SLA Offers an *Template*-Dokumenten, die durch den Responder bereitgestellt werden. Diese beschreiben die Rahmenbedingungen, unter denen der Provider gewillt ist, seinen Dienst einem Consumer anzubieten. WS-Agreement Templates entsprechen in ihrer Struktur weitgehend einem finalen SLA, können aber Teilbereiche offen lassen und verfügen über einen zusätzlichen Abschnitt zur Definition von Constraints zur SLA-Erzeugung.

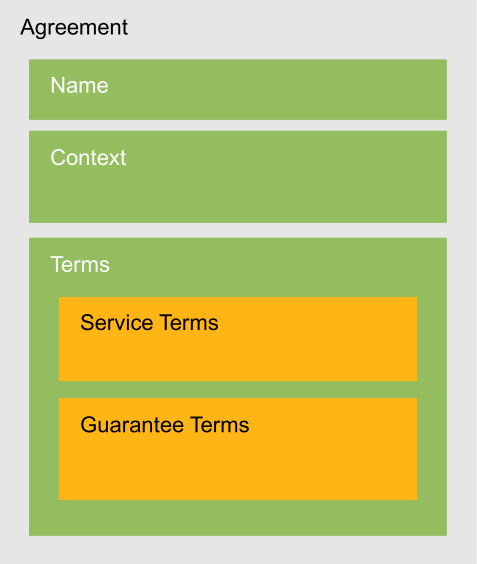

Abbildung 3.5: Struktur eines SLA-Dokuments nach [ACD+07]

Abbildung 3.5 zeigt den Aufbau eines WS-Agreement-konformen SLA-Dokuments. Neben dem optionalen Namen enthält jedes SLA einen *Context*-Bereich. Dieser beinhaltet die beteiligten Parteien (Initiator, Responder und Provider), ggf. eine Referenz auf das zugrunde liegende Template und den Zeitpunkt zu dem das SLA abläuft. Der eigentliche Inhalt des SLAs ist im Abschnitt *Terms* enthalten. Dieser untergliedert sich in *Service Terms* und *Guarantee Terms*.

Die Service Terms beschreiben den Service, über den das SLA geschlossen wird (unabhängig davon, ob dieser bereits existiert). Verpflichtende Teile der Service-Beschreibung sind

die Spezifikation eines Service-Namens und einer URI, unter der der Service für den Consumer bereitgestellt wird. Optional können messbare Eigenschaften (z. B. mittlere Antwortzeit, Durchsatz) eines Services spezifiziert werden, auf die die im Rahmen des SLAs definierten SLOs Bezug nehmen können. Weiterhin bietet WS-Agreement die Möglichkeit, die Service Terms um eine domänenspezifische Service-Beschreibung zu erweitern.

Die eigentliche Dienstgütevereinbarung wird in den Guarantee Terms geschlossen. Ein Guarantee Term bezieht sich auf eine entweder durch den Provider oder den Consumer zu erbringende Leistung und beschreibt Vorbedingungen, SLOs und eine Liste so genannter *Business Values*. Diese Business Values codieren Konzepte wie beispielsweise die relative Wichtigkeit der beschriebenen SLOs im Vergleich zu anderen SLOs im selben SLA oder bei SLO-Verletzung zu zahlende Strafen. Die Beschreibung von SLOs und Business Values erfolgt durch domänenspezifische Erweiterungen (beispielsweise die von IBM entwickelte WSLA-Spezifikation [KL03]) und wird nicht von WS-Agreement abgedeckt.

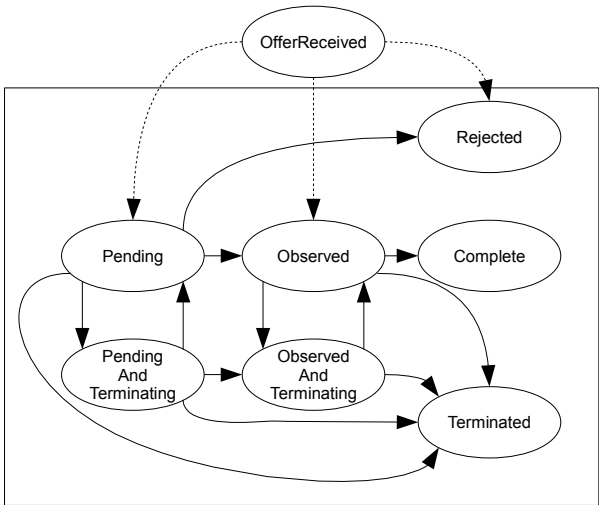

Abbildung 3.6: Für den Nutzer sichtbare Zustände eines SLAs nach [ACD+07]

Abbildung 3.6 zeigt die in der Spezifikation definierten, für den Consumer sichtbaren Zustände eines SLAs. Ausgehend von einem empfangenen Agreement Offer können drei unterschiedliche Zustände eingenommen werden: *Pending*, *Observed* und *Rejected*. Im Fall von Pending wurde noch nicht über die Annahme entschieden; Observed bedeutet, dass ein SLA geschlossen wurde und derzeit aktiv ist; Rejected bedeutet, dass das SLA abgelehnt wurde. Wurde das SLA angenommen und abgearbeitet (z. B. durch Erreichen des Ablaufzeitpunkts), befindet es sich im Zustand *Complete*. Die beteiligten Parteien haben die Möglichkeit ein vorzeitiges Auflösen des SLAs zu verlangen – in einem solchen Fall wechselt das SLA in den Zustand *Terminated*.

3.2.3.2 Application Response Measurement

Application Response Measurement (ARM) [TOG04] ist ein Open Group Standard für Zeitmessungen in verteilten Anwendungen. ARM unterstützt das Vermessen von *Response Times*, die Ausführungszeiten von Code-Abschnitten innerhalb einer Anwendung entsprechen. Der Standard bezeichnet diese Messungen als *ARM Transactions*. ARM Transactions werden im Quellcode einer Anwendung durch Messpunkte gekennzeichnet. Um die Unabhängigkeit von globalen Zeitquellen sicherzustellen, muss jede Messung innerhalb des Prozesses enden, in dem sie gestartet wurde. ARM erlaubt allerdings das Verknüpfen semantisch voneinander abhängiger Messungen, auch über Prozess- und Maschinengrenzen hinweg. Hierzu definiert der Standard so genannte *ARM Correlators*, eindeutige Tokens, die jeweils einer ARM Transaction zugeordnet sind.

Zur Instrumentierung einer Anwendung mithilfe von ARM muss ein Entwickler ARM-API-Aufrufe in den Quellcode der Anwendung integrieren. Diese Aufrufe werden zur Laufzeit durch eine ARM-Bibliothek entgegengenommen, die dann die eigentlichen Zeitmessungen durchführt. ARM definiert APIs für C und Java. Durch die Kombination der standardisierten ARM API mit herstellerspezifischen ARM-Bibliotheken wird eine direkte Integration in Enterprise Management Systeme erreicht. Entsprechende ARM-Bibliotheken werden von Herstellern wie IBM und HP angeboten. Um beispielsweise Messergebnisse in die IBM Tivoli Management-Umgebung einfließen zu lassen, muss lediglich die durch Tivoli bereitgestellte ARM-Bibliothek zur instrumentierten Anwendung hinzugebunden werden. Sollen zur Laufzeit keine Messungen vorgenommen werden, wird die Anwendung durch eine funktionslose so genannte Null-Bibliothek ergänzt. Auf diese Weise können die ARM-Aufrufe im Code der Anwendung verbleiben. Abbildung 3.7 gibt einen Überblick über den Ablauf einer ARM-Messung.

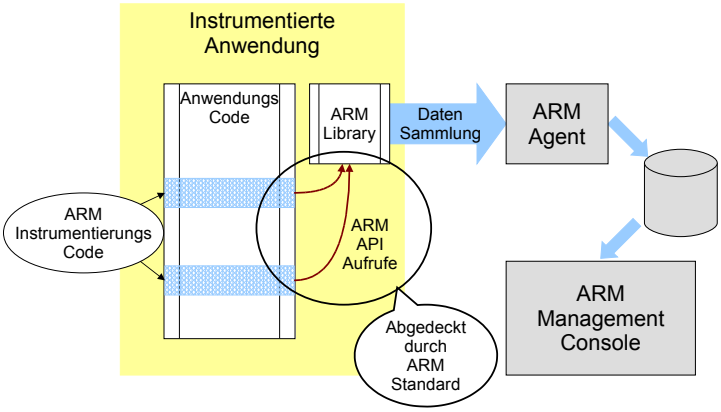

Abbildung 3.7: Performance-Instrumentierung einer Anwendung mit ARM

Semantisch wird eine instrumentierte Anwendung in ARM auf `ARMApplication`- und `ARMTransaction`-Objekte abgebildet, die eine laufende Applikationsinstanz und Zeitmes-

sungen instrumentierter Code-Abschnitte repräsentieren. Messungen werden im Quellcode durch `start()`- und `stop()`-Methoden des `ARMTransaction`-Objekts repräsentiert. Diese Aufrufe bilden damit das Fundament einer ARM-Instrumentierung. Zur Verknüpfung zweier Messungen kann einem `ARMTransaction`-Objekt der ARM Correlator einer anderen Messung übergeben werden. Bei derart verknüpften Messungen spricht man auch von Vater- und Kind-Transaktion.

Durch die Notwendigkeit, auch verteilte Anwendungen mit nachrichtenorientierter Kommunikation vermessen zu können, wurden mit der aktuellen Version 4.1 des ARM-Standards semantische Erweiterungen wie *asynchronous flows* und *message events* eingeführt. Diese erlauben es – in Ergänzung der oben beschriebenen Vater-Kind-Transaktionsbeziehung – lose verknüpfte, asynchron verlaufende Messungen zu modellieren. Zusätzlich erlauben ARM 4.1 API-Erweiterungen die Abbildung von Ereignissen (z. B. das Senden oder Empfangen einer Nachricht), wodurch m:n-Beziehungen zwischen Messungen unterstützt werden können.

3.3 Selbst-Management

3.3.1 Überblick

Mit der Forschung im Bereich "Selbst-Management" [PH07] werden aktuell Ansätze entwickelt, die den Administrator von Routineaufgaben entlasten sollen bzw. die der einfacheren Verwaltbarkeit von Komponenten dienen. Dabei wird allgemein davon ausgegangen, dass in zukünftigen Systemen ein Komplexitätsgrad erreicht wird, der menschliche Administratoren überfordert, sodass – um die Managebarkeit solcher Systeme weiterhin zu gewährleisten – „sich selbst managende" Software entwickelt werden muss, die gegenüber dem Administrator die Komplexität des zu verwaltenden Systems teilweise verbirgt.

Da Selbst-Management-Ansätze innerhalb der Informatik ein junges Forschungsgebiet darstellen, existieren für diesen Bereich derzeit weder eine allgemein anerkannte Taxonomie noch allgemeine Architektur-Patterns oder Vorgehensweisen. [KC03] formuliert einige grundlegende Definitionen zum IBM-Selbst-Management-Ansatz, dem *Autonomic Computing*. Die Definitionen wurden später durch andere Beiträge ergänzt und teilweise präzisiert. Sie werden in den folgenden Abschnitten zusammengefasst.

Wesentliche Elemente des Autonomic Computings – wie z. B. das Ziel der Automatisierung komplexer Regelungs- und Management-Vorgänge – finden sich auch im Intel-Ansatz *Proactive Computing* [Ten00] und in der deutschen *Organic Computing* Initiative [MSvdMW04] wieder.

3.3.2 Self-Management und Self-X-Properties

Als *Self-Management (Selbst-Management)* beschreibt [KC03] das teilweise Ersetzen von Management-Entscheidungen eines Administrators durch automatisierte Regelungskreisläufe, die eine Komponente in Bezug auf bestimmte Parameter in einem definierten Bereich halten oder versuchen, eine Komponente im Hinblick auf ein bestimmtes Ziel zu optimieren. Ziel des Selbst-Managements ist es langfristig, Systeme so zu konstruieren, dass diese ihre eigene Funktion und Leistung permanent selbst überwachen und an die durch ihre Umgebung vorgegebene Situation anpassen. Der Artikel definiert vier unterschiedliche Teildisziplinen des Selbst-Managements:

- *Self-configuration*: Ein sich selbst managendes System konfiguriert sich anhand von *High Level Policies* selbstständig. Administratoren spezifizieren lediglich ein gewünschtes Ergebnis, das System wählt eigenständig einen Weg, um dieses Ziel zu erreichen.
- *Self-optimization*: Ein sich selbst managendes System überwacht sich zur Laufzeit und optimiert sich auf Basis dieser Überwachung.
- *Self-healing*: Hard- und Softwareprobleme werden selbsttätig erkannt und – insofern möglich – automatisiert behoben.
- *Self-protection*: Ein sich selbst managendes System wehrt Angriffe selbsttätig ab und verhindert auf diese Weise größere Ausfälle.

Diese allgemeiner auch als *Self-X-Properties* bezeichneten Teildisziplinen werden allerdings in der Literatur nicht als erschöpfend angesehen [BJM+05]. Teilweise existieren Abgrenzungsprobleme zwischen den einzelnen Eigenschaften, so ist die Grenze zwischen Self-configuration und Self-optimization nicht klar definiert, Self-healing und Self-protection benutzen i. d. R. ebenfalls das Mittel der Konfigurationsänderung, um ihr Ziel zu erreichen.

3.3.3 Autonomic Elements

In [KC03] wird eine sich selbst managende Komponente (bestehend aus Geschäfts- und Management-Anteil) als *Autonomic Element* bezeichnet. Der Geschäftsanteil eines Autonomic Elements kann ein beliebiges *Managed Element* sein, beispielsweise eine Hard- oder Softwarekomponente mit Management-Schnittstellen. Der Management-Anteil eines Autonomic Elements wird durch einen *Autonomic Manager* implementiert, der den Geschäftsanteil über dessen Management-Schnittstellen verwaltet. Der Management-Prozess innerhalb eines Autonomic Managers untergliedert sich in vier Schritte, die in Abbildung 3.8 dargestellt sind:

- *Monitoring:* Über eine geeignete Überwachung der Geschäftskomponente werden zunächst für das Management relevante Informationen erhoben.

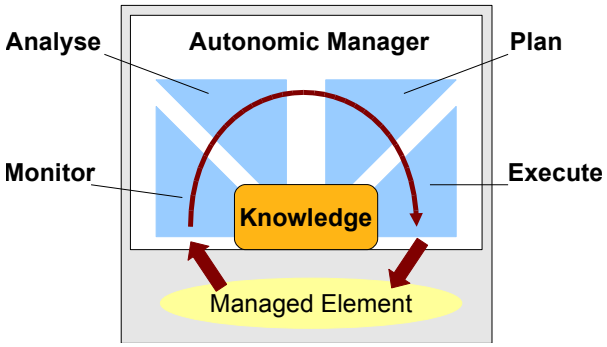

Abbildung 3.8: Struktur eines Autonomic Elements nach [KC03]

- *Analysis:* Im Analyse-Schritt wird aus den Monitoring-Daten der aktuelle Zustand der Geschäftskomponente ermittelt. Diese Analyse bildet die Grundlage für die Planung zukünftiger Management-Aktionen.
- *Planning:* Im Planning-Schritt wird geprüft, ob seitens des Managers ein Eingriff in die Konfiguration der Geschäftskomponente erforderlich ist (beispielsweise zur Optimierung des Arbeitsverhaltens oder zur Behebung eines Fehlers), ggf. werden geeignete Management-Operationen vorbereitet.
- *Execution:* Im Execution-Schritt werden die durch den Manager geplanten Operationen dann umgesetzt, die Geschäftskomponente also beispielsweise entsprechend den Management-Zielen rekonfiguriert.

Ein alternativer Ansatz zur Realisierung eines Autonomic Managers wird von [Str05] propagiert und durch andere Arbeiten aufgegriffen [SRS07, JvdMB+07]. Der in Abbildung 3.9 dargestellte Manager ist schwergewichtiger als der eben diskutierte Ansatz. Das Management der Geschäftskomponente erfolgt hier explizit über ein Management-Modell. Der Autonomic Manager verfügt über eine Reihe von Policies, die das Management der Geschäftskomponente steuern. Hierzu existieren – analog zum erstgenannten Ansatz – Aktionen zum Überwachen (*Observe*) der Geschäftskomponente, zum Planen (*Plan*) von Management-Aktionen und zum Ausführen (*Execute*) derselben. Zusätzlich existieren allerdings noch Module zur Analyse des Verhaltens der Geschäftskomponente (*Understand*) und – basierend darauf – zum Überarbeiten des vorhandenen Policy-Sets (*Learn*). Der Ansatz verzichtet auf die klare zeitliche Anordnung der einzelnen Aktionen in aufeinanderfolgende Schritte und konzentriert sich auf die Realisierung eines in Bezug auf seine Wissens- und Regelbasis adaptiven Autonomic Managers.

Unabhängig von der konkreten Ausgestaltung eines Autonomic Managers liegt diesem im Allgemeinen das Prinzip eines geschlossenen Regelkreises zugrunde, wie in Abbildung 3.10 dargestellt. Eine Ressource wird mithilfe von Sensoren beobachtet, d.h. bestimmte für das Management relevante Kenngrößen werden zunächst durch Messungen erfasst bzw. aus Messdaten abgeleitet. Basierend auf diesen Kenngrößen wird in Abstimmung mit externen Stellgrößen/

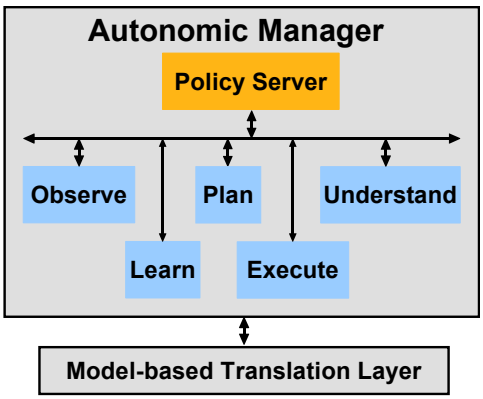

Abbildung 3.9: Modularer Aufbau eines Autonomic Managers nach [Str05]

Zielvorgaben/Policies eine Entscheidung über die Beeinflussung des Verhaltens der Ressource (*Regelung*) getroffen. Der Regelungseingriff wird dann mithilfe von entsprechenden Effektoren umgesetzt.

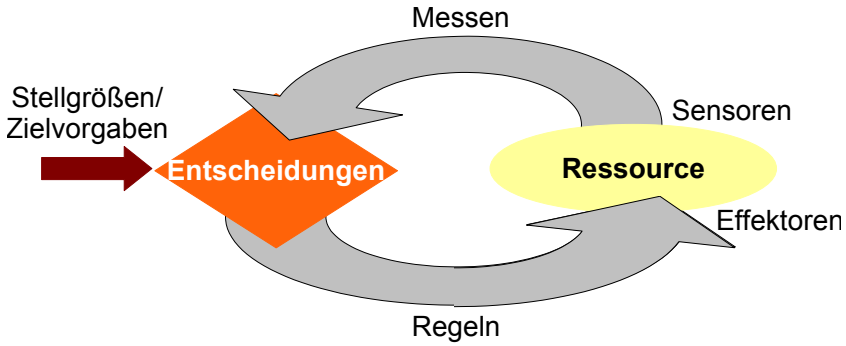

Abbildung 3.10: Prinzip eines geschlossenen Regelkreises nach [GC03]

In der Regelungstechnik unterscheidet man eine solche, über Sensorik rückgekoppelte *Regelung* von der einfacheren *Steuerung*, bei der die Entscheidungskomponente – ohne Wissen über den aktuellen Zustand des zu steuernden Systems – allein auf Basis von externen Stellgrößen agiert.

Mehrere aktuelle Arbeiten setzen Ansätze aus der Regelungstechnik, beispielsweise P-Regler oder PID-Regler, zur Realisierung von Kontrollalgorithmen für Autonomic Manager ein (vgl. [PGH+01, DEF+03]). Abbildung 3.11 zeigt beispielhaft den prinzipiellen Aufbau eines solchen Reglers. Die stetige Differenzierbarkeit der Kenngrößen eines Systems ist mathematische Grundlage für die Modellierung klassischer Regler [PB98, KBE99]. Daher ist für die Verwendung von regelungstechnischen Ansätzen auf Softwaresysteme deren in der Regel nicht stetig differenzierbares Verhalten problematisch.

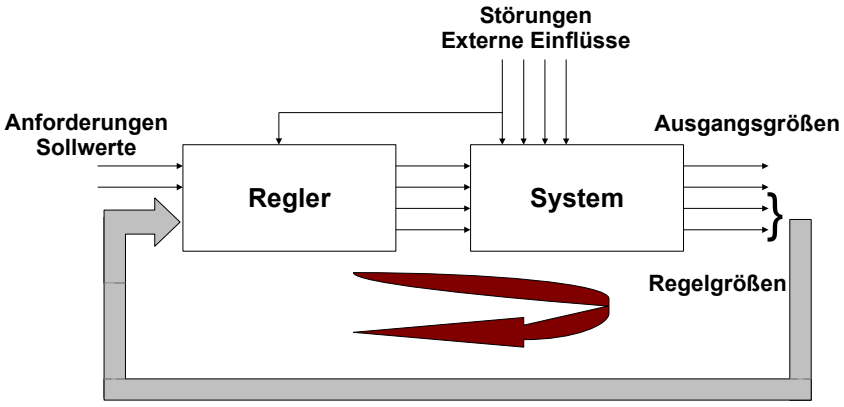

Abbildung 3.11: Regelung: Steuerung mit Rückkopplung

Alternativ werden oftmals Analogien aus der Biologie [SS05, SH06, BBD+06], aus sozialen oder Wirtschaftssystemen [WHH+92, Wel93, FGLS04] für das Design von Regelung und Zusammenarbeit von Autonomic Elements herangezogen. Beispiele hierfür sind das Nervensystem, Schwarm-Algorithmen, Systeme auf Vertrauens- und Bewertungsbasis, Auktionen und Märkte.

3.3.4 Abgrenzung zwischen Selbst-Management und Selbstorganisation

Allgemein wird für den Bereich des Selbst-Managements davon ausgegangen, dass sich mehrere Autonomic Elements ohne Zutun eines Administrators entsprechend der ihnen vorgegebenen Management-Ziele koordinieren können. Auf diese Weise soll eine sich selbst organisierende und sich entsprechend abstrakter Vorgaben selbst managende IT-Infrastruktur entstehen, die für die Administration eine deutliche Komplexitätsreduktion, aber auch verminderte Eingriffsmöglichkeiten bedeutet.

Derzeit existiert keine allgemein anerkannte Definition von Selbst-Management und Selbstorganisation im IT-Kontext, sodass heute unterschiedlichste Ansätze für sich beanspruchen können, Systeme mit Selbst-Management-Fähigkeiten zu realisieren. Ein vielversprechender Ansatz zur Definition und Abgrenzung beider Begriffe findet sich in [MWJ+07] und [HWM06].

In [MWJ+07] definieren die Autoren Kriterien zur Klassifizierung von adaptiven Systemen, selbstmanagebaren Systemen, selbstmanagenden Systemen und selbstorganisierenden Systemen auf Basis einer gemeinsamen Systemdefinition. Ein System $\mathfrak{S} = (\mathcal{I}, \mathcal{O}, \mathcal{B})$ wird dabei als Tupel aus Eingabe $\mathcal{I}$, Ausgabe $\mathcal{O}$ und einem Systemverhalten $\mathcal{B}$ modelliert. Systemeingaben werden in reguläre Eingaben und Eingaben zur Systemsteuerung unterschieden.

Ein adaptives System wird definiert als ein System, das für unterschiedliche gültige Eingabefunktionen ein akzeptables Systemverhalten aufweist. Eingaben dürfen hierbei sowohl regulärer als auch steuernder Natur sein. Ein System wird als selbstmanagebar bezeichnet, wenn man für das System einen Steuerungsmechanismus (Controller) beschreiben kann, der auf Basis regulärer Eingaben ein adaptives Systemverhalten erzeugt. Als selbstmanagend wird ein um einen solchen Steuerungsmechanismus erweitertes System bezeichnet, wenn es ohne weitere externe Steuerungseingaben ein adaptives Verhalten aufweist. Ein selbstorganisierendes System verändert nach [MWJ+07] zusätzlich seine Struktur mit dem Ziel der Adaption an Veränderungen. Weiterhin verfügt es über keine zentrale Kontrollinstanz (dezentraler Aufbau), wodurch Robustheit und Skalierbarkeit des Systems unterstützt werden. Die Autoren beschreiben ihre Sicht auf selbstorganisierende Systeme detailliert in [HWM06]. Abbildung 3.12 zeigt die zwischen den genannten Systemklassen bestehenden Beziehungen.

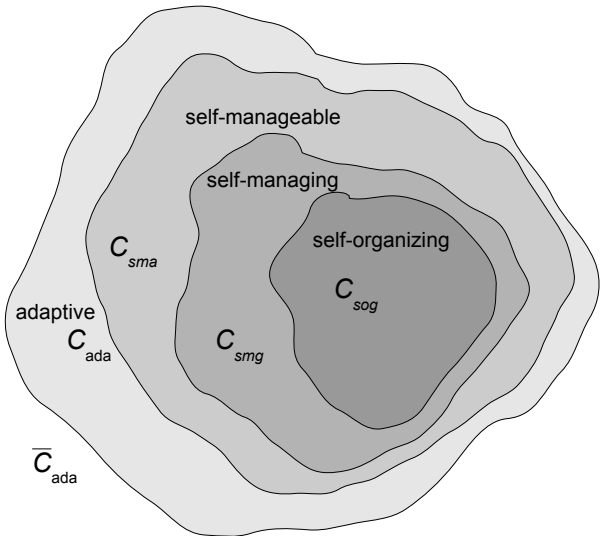

Abbildung 3.12: Klassifikation von Systemen nach [MWJ+07]

Die Autoren räumen allerdings selbst ein, dass die Klassifikation eines Systems immer aus einer bestimmten Betrachtung heraus erfolgt. Begriffe wie System und Systemkomponenten werden stets anwendungsabhängig definiert. Dies führt dazu, dass Systeme in der Regel nur in Bezug auf bestimmte betrachtete Aspekte in die oben vorgestellten Kategorien eingestuft werden können.

Eine weichere Definition von selbstorganisierenden Systemen nehmen De Wolf und Holvoet [DWH05a, DWH05b] vor: Hier wird ein selbstorganisierendes System dadurch gekennzeichnet, dass es – unabhängig von externer Kontrolle – eine innere Ordnung erzeugt.

Die Autoren betrachten selbstorganisierende Systeme im Kontext von *emergentem Systemverhalten*: Unterscheidet man innerhalb eines Systems Mikro- und Makro-Ebene, so sind die einzelnen Elemente des Systems auf der Mikro-Ebene angesiedelt. Emergentes Systemverhalten

ist nichtlinear und zeigt sich lediglich auf der Makro-Ebene des Systems. Es lässt sich durch eine Betrachtung der einzelnen Systembestandteile nicht vorhersagen. Durch Kombination von Selbstorganisation und (gewünschter) Emergenz soll eine innere Ordnung des Systems auf der Makro-Ebene hergestellt werden, ohne dass detailliert durch den Entwickler beschrieben werden muss, wie das System diese Ordnung erreichen kann. Die Autoren beschreiben aber keine allgemein gültige Vorgehensweise zum Design derartiger Systeme.

Mit dem Problem des Designs von Systemen unter Nutzung „gesteuerter Emergenz“ befassen sich auch Müller-Schloer und Sick in [MSS08]. Die Autoren schlagen vor, derartige Systeme mithilfe geschachtelter Regelkreise (mit zunehmender Abstraktion) als geschichtete Architektur zu realisieren. Dabei bleibt das System auch ohne die Existenz höherer Regelkreise lauffähig; diese dienen der Justierung genereller, durch das System zu realisierender Strategien.

3.3.5 Anwendung auf dienstorientierte Architekturen

Derzeitige Selbst-Management-Ansätze fokussieren im Allgemeinen auf das Management einzelner Komponenten und damit implizit auf Optimierungsziele, die sich auf die konkret betrachtete Komponente beschränken.

Bei SOA-Systemen, in denen unterschiedliche Dienste zusammenarbeiten, ist es jedoch wünschenswert, dass sich ein System auf der Ebene eines Workflows oder der gesamten Anwendungslandschaft optimiert. Die Optimierung einzelner Dienste birgt dabei die Gefahr, dass am gleichen Workflow beteiligte Dienste sich gegenseitig behindern. Gleichzeitig ist es aufgrund der in einer solchen Umgebung vorherrschenden Komplexität und Dynamik nicht möglich, einen kompletten Workflow als einzelne, im Rahmen eines Selbst-Management-Szenarios zu verwaltende Einheit anzusehen.

Die in Abschnitt 3.3.4 beschriebenen Ansätze zur Abgrenzung von Selbst-Management und Selbstorganisation bieten jedoch einen Ansatzpunkt zur Modellierung eines Dienstgüte-Management-Systems für SOA-Umgebungen: Legt man das von Müller-Schloer und Sick beschriebene Modell der geschachtelten Regelkreise zugrunde, so wird auf unterster Ebene ein Regelkreis innerhalb jedes SOA-Dienstes etabliert. Dieser lokale Regelkreis wird durch globale Zielvorgaben (z. B. die Korrektur des für den Dienst gültigen SLOs) beeinflusst, ist prinzipiell aber auch isoliert lauffähig. Der übergeordnete Regelkreis wird durch Selbstorganisation der beteiligten Komponenten etabliert und bezieht seine Vorgaben (ein Workflow-globales SLO) wiederum aus übergeordneten Vorgaben.

Das Konzept für eine nach diesen Prinzipien aufgebaute Architektur zum Dienstgüte-Management in SOAs wird in den Kapiteln 5 und 6 dieser Arbeit entwickelt.

4 Verwandte Arbeiten

Im Weiteren werden exemplarisch einige existierende Arbeiten und Lösungsansätze diskutiert, die die im Rahmen dieser Arbeit behandelte Thematik ergänzen. Ein Teil der hier vorgestellten Arbeiten ist unter Beteiligung bzw. Betreuung des Autors dieser Dissertation entstanden. Im Folgenden wird zunächst auf Instrumentierungsansätze eingegangen, die die Grundlage für das Monitoring von Anwendungen bilden. Im Anschluss werden existierende Arbeiten aus dem Bereich des Monitorings und des Dienstgüte-Managements von SOA-Umgebungen vorgestellt.

4.1 Monitoring von Diensten und Workflows

Im Folgenden werden beispielhaft einige Ansätze präsentiert, die die Integration eines geeigneten Monitorings in SOA-Dienste bzw. deren Middleware oder die zugrunde liegende Implementierung erleichtern. Dabei wird prinzipiell zwischen Unterstützung für das Monitoring neu zu entwickelnder und existierender Dienste unterschieden. Im Anschluss werden beispielhaft Ansätze für das Monitoring von Workflows vorgestellt.

4.1.1 Instrumentierung von Diensten

[SSK08] beschreibt einen Ansatz zur Integration einer Performance-Instrumentierung in neu zu entwerfende SOA-Dienste, der auf Methoden der modellgetriebenen Softwareentwicklung zurückgreift. Dabei wird die Unterstützung für Performance Monitoring bereits zum Modellierungszeitpunkt als integraler Design-Aspekt des zu entwickelnden Systems betrachtet. Die Auszeichnung der zu vermessenden Funktionalität erfolgt auf Ebene des UML-Modells eines Dienstes, im Verlauf des Entwicklungsprozesses werden dann mithilfe von Modelltransformationen und Codegenerierung Instrumentierungscode und eine einheitliche Monitoring-Schnittstelle erzeugt.

Durch den modellgetriebenen Entwicklungsansatz kann mit geringem Aufwand eine konsistente und vollständige Instrumentierung für neu zu entwickelnde Umgebungen realisiert werden.

Integration von Monitoring in existierende Anwendungen ist i. d. R. deutlich komplexer als das Design neuer Software unter Berücksichtigung von Monitoring-Anforderungen. Anwendungen verwenden unterschiedliche Kommunikations-Middleware, ggf. ist auch der Quellcode, der zur genauen Analyse des Anwendungsverhaltens benötigt wird, nicht verfügbar.

In [SSK09] und [SSTK08] wird ein Ansatz zur werkzeuggestützten manuellen Instrumentierung von Anwendungscode vorgestellt, mit dessen Hilfe zu vermessende Stellen im Quellcode einer Anwendung zunächst innerhalb der IDE grafisch ausgezeichnet werden können. In einem zweiten Schritt können die so gesetzten Messpunkte dann aktiviert werden, wodurch automatisiert entsprechender ARM-Instrumentierungscode in den Applikationsquelltext eingefügt wird. Auf ähnliche Weise können existierende Messpunkte verändert oder wieder entfernt werden. Mit dieser Methode kann auch die weiter oben beschriebene modellgetriebene Instrumentierung weiter verfeinert werden.

Ohne die Möglichkeit, Änderungen am Quellcode einer Anwendung vorzunehmen, sind die Möglichkeiten zur effektiven Instrumentierung stark eingeschränkt. Hier kann man entweder einen Black-Box-Ansatz für das Monitoring wählen und versuchen, den internen Zustand des Systems durch sein nach außen hin sichtbares Verhalten zu ergründen (z. B. durch Analyse von Log-Dateien und Kommunikationsverhalten, vgl. [AMW+03]), oder man versucht, auf anderen Wegen Instrumentierungscode in das System einzuschleusen (beispielsweise durch Instrumentierung der unterlagerten Kommunikations-Middleware oder Instrumentierung bestimmter Systembibliotheken, Bytecode-Manipulation, etc.).

Ein Ansatz für Middleware-Monitoring wird beispielsweise durch Pinpoint [CKF+02] realisiert. Das System fokussiert auf automatische Problemerkennung in großen, dynamischen, Internet-basierten Systemen. Pinpoint stellt eine Reihe instrumentierter Middleware-Komponenten bereit, mit deren Hilfe ein Request bei seinem Lauf durch das System beobachtet werden kann. Die Architektur erlaubt allerdings keine weitere Verfeinerung durch zusätzliche Messungen, das System verwendet einen nicht-standardisierten Messansatz und ist nicht offen für Erweiterungen.

In letzter Zeit befassen sich Arbeiten verstärkt mit einer konfigurierbaren Instrumentierung auf Middleware-Ebene: Hierbei müssen weder der Anwendungscode noch die verwendete Middleware modifiziert werden, die Instrumentierung erfolgt stattdessen durch Nutzung in der Middleware vorhandener, wohldefinierter Erweiterungsschnittstellen. Derartige Schnittstellen existieren in vielen Middleware-Architekturen in Form von Request Handlern, deren Nutzung zum Installationszeitpunkt und teilweise sogar zur Laufzeit konfiguriert werden kann. Diese ermöglichen den Zugriff auf den Inhalt einzelner Nachrichten bzw. auf Aufrufe und dazugehörige Antworten zur Laufzeit und dienen i. d. R. der transparenten Einbindung von Transaktionsbehandlung, Verschlüsselung oder Autorisierungsmechanismen. Gleichzeitig bieten sich derartige Schnittstellen zur Instrumentierung von Anwendungen an, wodurch eine direkte Manipulation des Anwendungscodes entfallen kann.

In diesem Kontext existieren beispielsweise Arbeiten zum Monitoring von CORBA-Anwendungen [NMMS99], J2EE-Anwendungen auf Basis von IBM WebSphere [IBM06b] oder Web Services [Sch06].

[DG04] beschreibt einen Ansatz zur einheitlichen Performance-Instrumentierung von Web Services auf Basis von Methoden der aspektorientierten Programmierung (AOP) und ARM. Die Autoren kombinieren den AOP-Ansatz mit einer Instrumentierung der Web Services Middleware Apache Axis[1], die dem oben beschriebenen Schema folgt. Der Ansatz unterstützt das Vermessen einzelner Methoden innerhalb der gekapselten Implementierung eines Web Services und liefert so bei Bedarf sehr feingranulare Informationen, die für Zwecke der Optimierung oder des Debuggings verwendet werden können. Durch die Nutzung von AOP-Techniken kann Monitoring-Unterstützung teilautomatisiert in die Anwendung integriert und im Entwicklungsprozess isoliert betrachtet werden.

In [STTK07] wird ein offener Ansatz zur komponentenübergreifenden Middleware-Instrumentierung beschrieben, der eine transparente Performance-Instrumentierung verteilter Anwendungen ermöglicht. Der Ansatz wurde u. a. in den Applikationsserver JBoss, den Web Container Apache Tomcat und den Apache Web Server integriert und basiert ebenfalls auf ARM. Er ermöglicht die Verfolgung und zeitliche Vermessung von Requests in und zwischen den vorgenannten Komponenten, sodass ein detailliertes Performance-Bild einer mithilfe dieser Middleware-Komponenten realisierten Anwendung entsteht. Da der Instrumentierungscode vollständig in die Middleware eingebettet ist, bedarf es keinerlei Änderungen am Quellcode der zu vermessenden Applikationen. Zur Verfeinerung der durch die Middleware erreichbaren Instrumentierung können zusätzliche Messpunkte in den Applikations-Code integriert und gemeinsam mit der Middleware-Instrumentierung ausgewertet werden.

Während noch vor einigen Jahren Anwendungsinstrumentierung weitgehend proprietär betrieben wurde, existieren heute einheitliche und teilweise sogar standardisierte Schnittstellen für unterschiedliche Anwendungsbereiche: Für strukturierte Log-Daten existiert das *Common-Base-Events*-Format (CBE) [OKS+03], und Logging APIs wie Apache Log4J und Log4cxx finden inzwischen breite Verwendung. Für die Performance-Instrumentierung in verteilten Anwendungen wird zunehmend der in Abschnitt 3.2.3.2 vorgestellte ARM-Standard eingesetzt: Beispiele hierfür sind IBM WebSphere [IBM06a] und die Eclipse *Testing and Performance Tools Platform* (TPTP) [IBM07].

4.1.2 Instrumentierung von Workflow Engines

Neben dem Erheben von Monitoring-Daten in SOA-Diensten ist für eine umfassende Überwachung von SOA-Prozessen die Beobachtung der koordinierenden Workflow Engine notwendig. Im Folgenden werden beispielhaft existierende Schnittstellen und Erweiterungsansätze für das Monitoring in unterschiedlichen WfMSs vorgestellt.

[1] Details unter `http://ws.apache.org/axis/`

Die Workflow Engine *Apache Orchestration Director Engine (ODE)*[2] bietet ein Management API zum Monitoring und zur Verwaltung von WS-BPEL-Prozessen und deren Instanzen. Neben dem aktuellen Zustand einer Prozessinstanz und den Werten der Prozessvariablen kann man zu beendeten Instanzen Informationen über Laufzeit und aufgetretene Fehlerzustände erhalten. ODE liefert keine Informationen über die aktuell in einer Prozessinstanz abgearbeitete BPEL-Aktivität oder Antwortzeiten aufgerufener Web Services. Ein Ansatz für ein detaillierteres Monitoring von ODE-Aktivitäten wird in [Utk07] beschrieben.

[ZBS09] beschreibt einen Ansatz zur Erweiterung der Workflow Engine *ActiveBPEL*[3]. Innerhalb der Architektur wird ein Ereignismonitor registriert, der auf interne Nachrichten reagiert, die das Deployment eines WS-BPEL-Prozesses, das Erzeugen oder Beenden einer Prozessinstanz oder Zustandswechsel von WS-BPEL-Aktivitäten signalisieren. Der Ansatz nutzt diese Ereignisse, um regelbasiert auf Fehlerzustände reagieren zu können. Als Beispiel für eine mögliche Reaktion auf einen Fehler wird z. B. der Austausch eines unzuverlässigen Dienstes durch einen funktional äquivalenten, zuverlässigeren Dienst genannt.

[BW06] beschreibt exemplarisch die Erweiterung der ActiveBPEL Engine (Version 2.0) um ein Laufzeit-Monitoring auf Basis von ARM. In der verwendeten Version bietet ActiveBPEL selbst keinerlei Schnittstellen für das Überwachen der Antwortzeit von Workflows oder genutzten Diensten. Der in [BW06] beschriebene Ansatz erreicht ein entsprechendes Monitoring mithilfe einer Instrumentierung des Quellcodes der BPEL Engine. Durch Einfügen von Messpunkten an bestimmten Stellen im Quellcode werden relevante Ereignisse im Lebenszyklus eines BPEL-Prozesses vermessen. Im Einzelnen sind dies Start und Terminieren einer Instanz, Abarbeitung eines Prozessschritts, Aufruf externer Web Services und ggf. der Wechsel eines Workflows in Fehlerzustände. Jedem Zeitstempel werden dabei Kontextinformationen hinzugefügt, die eine genaue Analyse des jeweiligen Prozesszustandes zulassen. Die Instrumentierung anderer WfMSs ist prinzipiell nach gleichem Muster möglich.

Die beschriebenen Instrumentierungsansätze zeigen, dass eine Integration von Monitoring-Funktionalität in WfMSs und SOA-Dienste mit begrenztem Aufwand durchgeführt werden kann. Als Resultat stehen standardisiert erhobene Monitoring-Informationen zur Verfügung (z. B. in Form von ARM-Messdaten). Im Rahmen dieser Arbeit werden diese Monitoring-Daten zur Überwachung von SOA-Diensten genutzt, die Gewinnung solcher Daten steht jedoch nicht im Vordergrund.

4.2 Dienstgüte in SOA-Umgebungen

Aufgrund der allgemeinen Zunahme kritischer Anwendungen, die nach dem SOA-Architekturmuster aufgebaut sind, widmet sich eine Reihe von Arbeiten der Betrachtung von Dienst-

[2] Details unter `http://ode.apache.org`
[3] Details unter `http://www.activebpel.org`

güte über Komponentengrenzen hinweg.

Mehrere Ansätze realisieren die Einhaltung von Dienstgüteanforderungen durch dynamische Selektion alternativer Dienste. Ein Beispiel für einen solchen Ansatz ist WSQoSX [BGR+05]. Diese Architektur unterstützt ein QoS-abhängiges Binding von Diensten, wobei unter anderem die Antwortzeit eines Dienstes als Auswahlkriterium herangezogen wird. WSQoSX wählt unter mehreren Diensten mit äquivalenter Funktionalität diejenigen aus, mit deren Beteiligung an die Architektur gestellte SLAs erfüllt werden können. Der Ansatz unterstützt keine zustandsbehafteten Dienste und beschränkt sich auf die isolierte Betrachtung der Dienstgüte einzelner Services.

In [AP07] wird ein Verfahren zur dynamischen Selektion alternativer Web Services auf Basis von globalen QoS-Kriterien vorgestellt: Aus einem Pool alternativer Dienste werden diejenigen gewählt, die sicherstellen, dass die global geforderte Dienstgüte erbracht werden kann. Dabei werden sowohl komplexe Workflow-Strukturen als auch zuvor definierte Constraints berücksichtigt, sodass auch Prozesse mit zustandsbehafteten Diensten unterstützt werden können.

In [SPJ08] wird ein Ansatz zur Optimierung der Dienstgüte von Prozessen durch redundante Einbindung unzuverlässiger Dienste vorgestellt. Die Autoren beschreiben eine Heuristik, mit deren Hilfe die Ausführung eines Prozesses unter Minimierung der genutzten Ressourcen optimiert werden kann. Zur Einbindung alternativer Dienste werden im Rahmen des mehrstufigen Auswahlprozesses semantische Beschreibungen verwendet, sodass die bereitgestellten Dienste nicht notwendigerweise syntaktisch identische Schnittstellen bereitstellen müssen.

In [MDGA08] wird ein modellgetriebener Ansatz zur Realisierung einer Monitoring-Infrastruktur für zusammengesetzte Dienste vorgestellt, die in existierende Management-Umgebungen integriert werden kann. Die Autoren konzentrieren sich bei der Erfassung von Performance-Kenngrößen auf die Ebene des WfMS, wodurch es möglich ist, eine Auswahl der zu vermessenden Operationen aus Geschäftssicht zu treffen.

Teil II

Entwurf

5 Architektur

Im Folgenden wird die Architektur eines Management-Systems zur Durchsetzung von Dienstgüteanforderungen in dienstorientierten Architekturen beschrieben. Zur komponentenübergreifenden Optimierung der Dienstgüteeigenschaften kommen Methoden aus dem Bereich des Selbst-Managements und der Selbstorganisation zum Einsatz. Insgesamt wird ein dezentraler Architekturansatz verfolgt, der das Dienstgüte-Management in Bezug auf Änderungen an Geschäftsprozessen und Diensteinfrastruktur robust machen soll.

Abschnitt 5.1 fasst zunächst die Anforderungen an ein Management-System zur Durchsetzung von Dienstgüteanforderungen in dienstorientierten Architekturen zusammen. Abschnitt 5.2 gibt einen Überblick über den architekturellen Gesamtansatz, bevor in den darauf folgenden Abschnitten genauer auf einzelne Aspekte des Ansatzes eingegangen wird.

5.1 Anforderungen und Rahmenbedingungen

Für das Dienstgüte-Management auf SOA-Ebene ergeben sich im Vergleich zur Betrachtung des SLMs einzelner, unabhängiger Multi-Tier-Client/Server-Applikationen neue Herausforderungen.

Die Anforderungen an ein Dienstgüte-Management-System für dienstorientierte Architekturen lassen sich in folgende Bereiche einteilen:

1. Funktionale Anforderungen
2. Nichtfunktionale Anforderungen
3. Integrationsanforderungen

Die folgenden Abschnitte stellen die einzelnen Anforderungen nach diesen Bereichen gegliedert vor.

5.1.1 Funktionale Anforderungen

Funktionale Anforderungen an das Dienstgüte-Management-System betreffen die Entgegennahme und Verarbeitung von Dienstgüteanforderungen sowie das Sicherstellen des Erfüllens der vereinbarten Dienstgüte.

F1 *Formale Definition von Dienstgüteanforderungen*
Das System soll Dienstgüteanforderungen in Form formal definierter SLOs interpretieren und verarbeiten können.

Zur maschinellen Verarbeitung und Interpretation müssen Dienstgüteanforderungen formal definiert werden. Wie in Abschnitt 3.2 dargestellt, werden im Bereich des SLMs Dienstgüteanforderungen typischerweise in Form von SLOs als Teil eines SLAs vereinbart, die – falls sie in geeigneter Weise codiert sind – maschinell ausgewertet werden können.

F2 *Entgegennahme von Dienstgüteanforderungen*
Das System soll Dienstgüteanforderungen sowohl auf der Workflow- als auch auf der Dienst-Ebene entgegennehmen können.

Dienstgüteanforderungen können prinzipiell für alle in einer IT-Anwendungslandschaft modellierten Komponenten formuliert werden. Im SOA-Kontext bedeutet dies, dass Anforderungen für ganze Workflows bzw. Teile derselben oder auch die Benutzung einzelner Dienste definiert werden können.

F3 *Abbildung von Dienstgüteanforderungen*
Das System soll für einen Workflow definierte SLOs in geeigneter Weise auf die beteiligten Dienste abbilden.

Falls ein SLA für einen Workflow oder Teile desselben geschlossen wird, soll das Dienstgüte-Management-System auf Basis der Struktur des Workflows entsprechende Dienstgüteanforderungen für die beteiligten Dienste ableiten. Die abgeleiteten SLOs müssen dann an die beteiligten Dienste kommuniziert werden. Dabei ist zu berücksichtigen, dass einzelne Dienste u. U. unterschiedlichen administrativen Domänen angehören.

F4 *Durchsetzung von Dienstgüteanforderungen*
Das System soll die Einhaltung der vereinbarten Dienstgüte sicherstellen.

Hierzu wird – wie in Kapitel 3 beschrieben – das Systemverhalten überwacht, analysiert und ggf. korrigiert. Dabei können keine Annahmen über die interne Struktur eines Dienstes gemacht werden, da diese durch die Dienstschnittstelle vollständig gekapselt ist.

F5 *Eskalation*
Im Fehlerfall soll das System die Vertragspartner über Verletzungen der Dienstgüte informieren.

Das System soll die Vertragsparteien über aufgetretene SLO-Verletzungen informieren. Dadurch können ggf. auf übergeordneter Ebene Maßnahmen zur Korrektur des Systemzustandes getroffen werden.

F6 *Behandlung von Konflikten*
Das System soll Konflikte bei der Durchsetzung konkurrierender SLOs erkennen und automatisiert beheben.

Durch die in einer SOA existierende, komplexe Verflechtung von Workflows und Diensten kann es bei der SLO-Vereinbarung zu Konkurrenzsituationen und Konflikten kommen. Da SLOs nach **F3** automatisiert abgeleitet werden können, ist es erforderlich, dass das System Konfliktsituationen auch automatisiert beheben kann.

F7 *Verteilte Kontrolle*
Das System soll Anwendungsumgebungen mit verteilter Kontrolle unterstützen.

Eine weitere Herausforderung besteht im Management von Umgebungen mit ausgelagerten Diensten oder Prozessen. Hierdurch können eine Vielzahl unterschiedlicher administrativer Domänen und Schnittstellen zu Dienstleistern entstehen, die in das Dienstgüte-Management einbezogen werden müssen. Die traditionell hierarchische Struktur von Enterprise Management-Systemen wie IBM Tivoli [Tiv03] oder HP OpenView [HP97] ist hingegen auf ein zentralisierbares Management der gesamten IT-Infrastruktur ausgelegt.

5.1.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen an das Dienstgüte-Management-System betreffen primär geforderte Skalierbarkeit und Flexibilität des Systems.

N1 *Orientierung an der Geschäftsarchitektur*
Das System soll sich in Kommunikation und Struktur an den durch Geschäftsprozesse vorgegebenen Beziehungen zwischen Workflows und Diensten orientieren.

Durch die Realisierung von Applikationen als durch Definition mitunter kurzlebiger Workflows realisierte Kompositionen von Diensten entstehen in einer SOA gleichzeitig hohe Flexibilität und Dynamik. Die Aufteilung von Applikationen in Dienste und Workflows erlaubt dem

Betreiber einer SOA die einfache Anpassung an sich ändernde Rahmenbedingungen. Traditionelle, statische Management-Ansätze scheitern jedoch insbesondere an der zum Management solcher Strukturen notwendigen Flexibilität [GS05].

N2 *Fokus*
Das System soll in der Lage sein, Anwendungen eines Unternehmens applikationsübergreifend zu betrachten.

Durch die weitgehende Auflösung von Anwendungsgrenzen wird – wie in Kapitel 2 dargestellt – in einer dienstorientierten Architektur bewusst der Integrationsgrad der IT-Prozesse eines Unternehmens erhöht. Als Konsequenz sollte das SOA-Dienstgüte-Management-System eine globale Perspektive zur applikationsübergreifenden Betrachtung der Anwendungen eines Unternehmens einnehmen.

N3 *Skalierbarkeit*
Die Architektur des Systems soll skalierbar gestaltet sein.

Die Anzahl der gleichzeitig innerhalb einer SOA existierenden Komponenten sowie der Parallelitätsgrad in der Ausführung können sehr hoch sein und es können komplexe Abhängigkeitsverhältnisse auftreten: Konkurrierende Workflows (oder konkurrierende Instanzen von Workflows) greifen gemeinsam auf unterschiedliche Dienste zu. Dadurch ergeben sich Anforderungen an die Skalierbarkeit des Management-Systems, damit es im Produktivbetrieb einer unternehmensweiten SOA keinen Engpass bildet.

N4 *Robustheit*
Die Architektur des Systems soll robust auf Änderungen an Diensten und Geschäftsprozessen reagieren.

Die vom System durchzusetzenden QoS-Anforderungen sind Veränderungen unterworfen. Außerdem kann nicht davon ausgegangen werden, dass die Abhängigkeiten innerhalb der SOA über einen längeren Zeitraum stabil bleiben, da sich die ablaufenden Workflows an die Organisation eines Unternehmens und seiner Abteilungen anlehnen und Unternehmen zunehmend von Flexibilität ihrer Geschäftsprozesse abhängen. Zudem werden Dienst-Implementierungen oftmals weiterentwickelt, was u. U. veränderte Ressourcen-Abhängigkeiten zur Folge hat, auch wenn sich die Schnittstellen nach außen nicht verändern. Mit der aktuell stark zunehmenden Server-Konsolidierung durch Virtualisierung verschärft sich das Problem sich zur Laufzeit verändernder Ressourcen-Abhängigkeiten noch, da unterschiedliche virtuelle Maschinen auf gemeinsamen physikalischen Ressourcen laufen können und zudem aktuelle Virtualisierungsansätze die Migration von virtuellen Maschinen zur Laufzeit unterstützen.

5.1.3 Integrationsanforderungen

Integrationsanforderungen an das Dienstgüte-Management-System betreffen sowohl die Integration mit der Implementierung existierender Dienste als auch die Integration des Aspekts Dienstgüte-Management in die SOA insgesamt.

I1 *Transparente Integration*
Das System soll möglichst transparent in existierende Architekturen integriert werden können.

Um bestehende Investitionen nicht zu gefährden, darf die Etablierung eines SOA-Dienstgüte-Managements nicht dazu führen, dass bereits existierende Systemkomponenten grundlegend überarbeitet werden müssen.

I2 *Eingeschränkte Kooperation*
Das System soll Umgebungen unterstützen, in denen nur ein Teil der Komponenten Unterstützung für das SLM bereitstellt.

Die Einführung des Management-Systems darf nicht daran scheitern, dass u. U. einige der genutzten Dienste nicht in das System einbezogen werden können, z. B. weil diese von einem Fremdanbieter verwaltet werden. Diese Forderung ergänzt **F7**.

I3 *Nutzung existierender Funktionalität*
Das System soll die Integration der ggf. vorhandenen SLM-Funktionalität existierender Dienste ermöglichen.

Die Anbindung des Systems an einzelne Dienste und deren Schnittstellen muss flexibel gestaltet werden. Ggf. vorhandene SLM-Funktionalität einzelner Dienste soll weiter genutzt werden können.

I4 *Integration mit anderen SLM-Architekturen*
Die durch das System offerierten Schnittstellen sollen durch bestehende SLM-Architekturen genutzt werden dürfen, um die Integration in Enterprise-Management-Umgebungen zu erleichtern.

Zur Integration in existierende Management-Umgebungen ist es wichtig, dass das System entsprechende Schnittstellen anbietet. Das Prinzip der losen Kopplung muss innerhalb einer SOA auch für die Interaktion von Management-Komponenten gelten.

5.2 Architekturüberblick

Die Architektur des Dienstgüte-Management-Systems muss den in Abschnitt 5.1 beschriebenen Anforderungen genügen.

Aufgrund der Anforderungen **F2** *(Entgegennahme von Dienstgüteanforderungen)* und **N1** *(Orientierung an der Geschäftsarchitektur)* wird für das Dienstgüte-Management-System eine modulare Architektur vorgesehen.

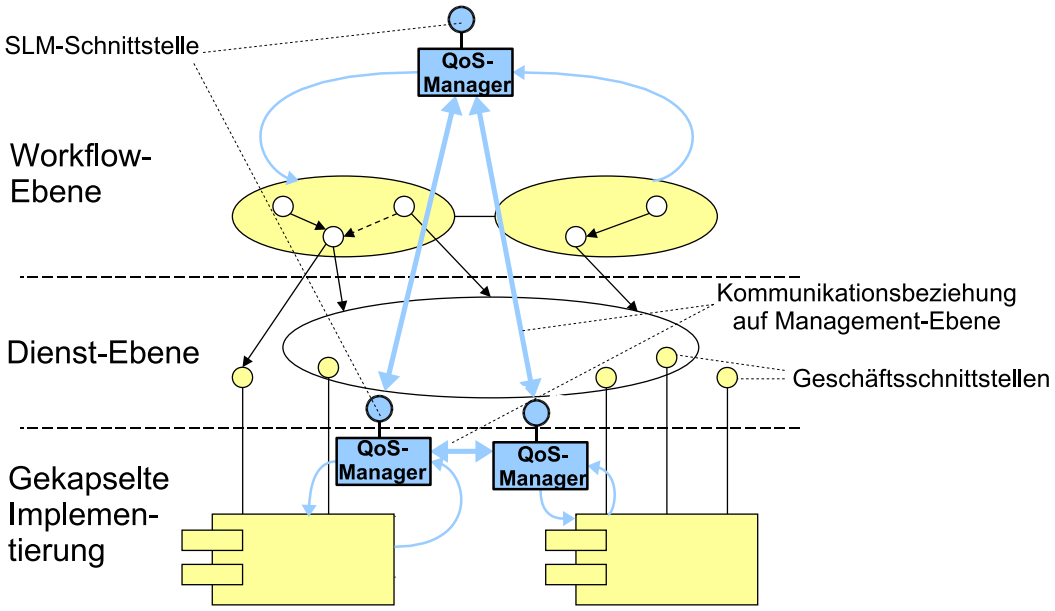

Abbildung 5.1: SLM auf Dienst- und Workflow-Ebene

Jedem Architekturelement innerhalb der SOA (Diensten und Workflows) wird – wie in Abbildung 5.1 dargestellt – logisch eine Dienstgüte-Management-Komponente zugeordnet. Diese wacht über die Einhaltung der Dienstgüteanforderungen des zugeordneten Elements und wird im Folgenden als *QoS-Manager* bezeichnet.

Um der Anforderung **F1** *(Formale Definition von Dienstgüteanforderungen)* gerecht zu werden, verfügt jeder QoS-Manager über eine Schnittstelle zur Vereinbarung von SLOs für die Nutzung des zugeordneten SOA-Architekturelements, die im Folgenden als *SLM-Schnittstelle* bezeichnet wird. QoS-Manager für Workflows und Dienste unterscheiden sich in Bezug auf die SLM-Schnittstelle nicht, sodass der Nutzer nicht unterscheiden muss, ob ein SLO mit einem Workflow oder einem Dienst vereinbart wird. Über die SLM-Schnittstelle werden Vertragspartner zur Laufzeit über Einhaltung oder Verletzung der vereinbarten SLOs informiert (Anforderungen **F5** *(Eskalation)* und **I4** *(Integration mit anderen SLM-Architekturen)*).

Zur Abbildung eines Workflow-globalen SLOs auf die beteiligten Dienste (Anforderung **F3** *(Abbildung von Dienstgüteanforderungen)*) nutzt ein QoS-Manager für Workflows die SLM-

Schnittstellen der QoS-Manager der beteiligten Dienste. Betrachtet man den einer Komponente zugeordneten QoS-Manager als Bestandteil dieser Architekturkomponente, so wird durch dieses Vorgehen die Anforderung **N1** *(Orientierung an der Geschäftsarchitektur)* gestützt, da auf Geschäftsebene Workflow-Engine und am Workflow beteiligte Dienste ebenfalls untereinander kommunizieren.

Befinden sich die durch einen Workflow genutzten Dienste in unterschiedlichen administrativen Domänen (vgl. Anforderung **F7** *(Verteilte Kontrolle)*), können SLOs nur in gegenseitigem Einvernehmen vereinbart werden. Ggf. bietet ein QoS-Manager lediglich die Zusicherung einer zuvor dauerhaft vereinbarten Dienstgüte an. Dies muss bei der Abbildung von Dienstgüteanforderungen auf die Dienste eines Workflows berücksichtigt werden. U. U. existiert für einen extern zugekauften Dienst überhaupt kein QoS-Manager, sodass kein SLO vereinbart werden kann. In einem solchen Fall kann der QoS-Manager eines Workflows zur Abschätzung des Verhaltens des beteiligten Dienstes lediglich ein zuvor gemessenes Dienstgüteverhalten (z. B. in Bezug auf Durchsatz oder Antwortzeit) zugrunde legen (Anforderung **I2** *(Eingeschränkte Kooperation)*).

In der durch Abbildung 5.1 dargestellten Situation bildet der QoS-Manager des Workflows an ihn gestellte Dienstgüteanforderungen gemäß der Struktur dieses Workflows auf die beteiligten Dienste ab und vereinbart mit diesen entsprechende SLOs über die SLM-Schnittstellen ihrer QoS-Manager. Die entsprechenden Kommunikationsbeziehungen zwischen den QoS-Managern sind in der Abbildung durch Pfeile dargestellt.

Die Integrationsanforderung **I1** *(Transparente Integration)* wird durch die Einführung einer separaten SLM-Schnittstelle berücksichtigt, da so zur Kommunikation von Dienstgüteanforderungen keine Veränderungen an den bestehenden Geschäftsschnittstellen existierender Komponenten vorgenommen werden müssen.

Die interne Architektur und Funktion eines QoS-Managers wird durch die SLM-Schnittstelle nach außen gekapselt, dies entspricht dem SOA-Prinzip der Kapselung eines Dienstes durch seine Schnittstelle. Hinsichtlich ihrer Aufgaben und ihrer internen Struktur unterscheiden sich die innerhalb einer dienstorientierten Struktur zum Einsatz kommenden QoS-Manager je nach Funktion und Anbindung an die zugeordnete Geschäftskomponente.

Um die Einhaltung der vereinbarten SLOs sicherzustellen (vgl. Anforderung **F4** *(Durchsetzung von Dienstgüteanforderungen)*), muss jeder QoS-Manager die ihm zugeordnete SOA-Komponente zur Laufzeit überwachen. Werden die vereinbarten Ziele nicht erreicht, muss kontrollierend in das System eingegriffen werden. Neben dem Erfüllen der Dienstgüteanforderungen kann dabei die Optimierung der Ressourcennutzung der Geschäftskomponenten ein weiteres Ziel sein, da ein sparsamer Ressourcenverbrauch neben finanziellen Einsparungen im Betrieb auch die Erhöhung der Kapazität des Gesamtsystems zur Folge haben kann. Abbildung 5.1 zeigt die Management-Beziehung zwischen QoS-Managern und den ihnen zugeordneten Geschäftskomponenten als Monitoring-Control-Schleife.

Auf Dienst-Ebene kann ein QoS-Manager bei drohender SLO-Verletzung u. U. automatisiert korrigierend auf die Implementierung einwirken (Selbst-Management), auf Workflow-Ebene müssen SLOs ggf. mit den beteiligten Diensten nachverhandelt werden. Alternativ kann – falls eine Auswahl äquivalenter Dienste zur Verfügung steht – ein anderer Anbieter mit besseren Dienstgütegarantien gewählt werden.

Nach Anforderung **N2** *(Fokus)* soll das SLM-System die Anwendungslandschaft über Workflow-Grenzen hinweg betrachten, wodurch sich Optimierungsmöglichkeiten in Bezug auf die globale Ressourcennutzung ergeben, da beispielsweise Workflows global priorisiert werden oder Dienste bei der Nutzung geteilter Ressourcen kooperieren können. Hierfür müssen sich die beteiligten QoS-Manager ggf. über geplante Ressourcennutzung austauschen und eine Möglichkeit zur Bearbeitung von Anfragen in unterschiedlichen Dienstgüteklassen schaffen. In Abbildung 5.1 ist dies als Kommunikationsbeziehung zwischen den QoS-Managern auf Dienst-Ebene dargestellt.

Zur Behandlung von Konflikten (Anforderung **F6** *(Behandlung von Konflikten)*) bei der Bearbeitung von Workflows und Diensten nutzt das System die Möglichkeit zur Priorisierung von Anfragen nach der Wichtigkeit des über einen Workflow realisierten Geschäftsprozesses. Bei der Spezifikation von SLAs ordnet der Benutzer der Einhaltung der Dienstgütekriterien des betroffenen Workflows einen monetären Wert zu, der im Vergleich mit der Wertigkeit anderer Workflows dessen Priorität im Gesamtkontext der Anwendungslandschaft eines Unternehmens widerspiegelt.

In Bezug auf Änderungen an Diensten und Workflows muss sich die Architektur robust verhalten (Anforderung **N4** *(Robustheit)*). Robustheit bedeutet in diesem Kontext Minimieren des Einflusses auftretender Änderungen auf das Verhalten des Gesamtsystems. In der SLM-Architektur wird dieses Ziel durch konsequente Modularisierung und Dezentralisierung der Kontrolle verfolgt. Ein einem Workflow zugeordneter QoS-Manager vereinbart zwar initial SLOs mit den QoS-Managern der beteiligten Dienste, er greift aber erst dann in das Verhalten des Systems ein, wenn das insgesamt zu erreichende Dienstgüteziel in Gefahr gerät. Durch dieses Verhalten soll eine Überreaktion des Systems auf Änderungen im Verhalten einzelner Dienste verhindert werden.

Zur optimalen Nutzung der vorhandenen Ressourcen gehört auch, dass das System nicht versucht, unnötig eng gefasste SLOs durchzusetzen, sondern lediglich anstrebt, die für die global gestellten Anforderungen notwendige Dienstgüte sicherzustellen. Hierzu kommen Mechanismen der Selbstorganisation zum Einsatz: QoS-Manager einzelner Dienste kommunizieren direkt miteinander, um eine Veränderung der von ihnen zu erbringenden Dienstgütekriterien auszuhandeln, ohne dass dies Auswirkungen auf die Dienstgüte des übergeordneten Workflows hat. So können z. B. innerhalb eines Workflows sequenziell aufgerufene Dienste die an ihre Antwortzeit gestellten Anforderungen lockern und straffen, ohne dass die global einzuhaltende Antwortzeit sich verändert.

Dezentralisierte Kontrolle, Modularisierung, Selbstorganisation und Selbst-Management sind die wesentlichen Säulen, die zur Skalierbarkeit der Architektur (vgl. Anforderung **N3** *(Skalierbarkeit)*) beitragen. Gleichzeitig minimiert die konsequente Ausrichtung der SLM-Architektur an der auf Geschäftsebene etablierten Struktur der SOA den Administrationsaufwand für das SLM-System.

5.3 Zuordnung von QoS-Managern zu Geschäftskomponenten

Jeder Architekturkomponente einer SOA, d. h. allen Dienst-Implementierungen und jeder WfMS-Instanz, wird zur Realisierung der Dienstgüte-Management-Funktionalität technisch ein QoS-Manager zugeordnet. Dienst-Implementierungen und WfMS-Instanzen werden im Folgenden zusammenfassend als *Geschäftskomponenten* bezeichnet. Die Zuordnung von QoS-Managern zu Geschäftskomponenten ist in Abbildung 5.2 dargestellt.

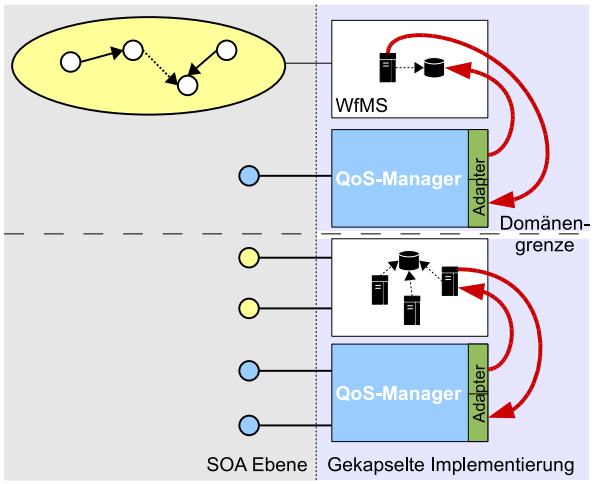

Abbildung 5.2: Zuordnung von QoS-Managern zu Geschäftskomponenten

QoS-Manager stellen eigenständige SOA-Dienste dar, die Services zum Dienstgüte-Management der ihnen zugeordneten Geschäftskomponenten über eine SLM-Schnittstelle offerieren. Sie entsprechen damit selbst der in Abschnitt 2.4.1 getroffenen Definition für SOA-Dienste (vgl. Definition 2.2).

5.3.1 Kommunikation zwischen QoS-Manager und Geschäftskomponente

Logisch erfolgt aus SOA-Sicht eine 1:1-Zuordnung zwischen einem QoS-Manager und einer Dienstschnittstelle seiner Geschäftskomponente, technisch ist eine QoS-Manager-Instanz jedoch der Implementierung eines Dienstes zugeordnet, da hier Monitoring bzw. regelnde Eingriffe erfolgen müssen. Offeriert eine Geschäftskomponente mehrere Dienste (oder Workflows), so stellt der jeweilige QoS-Manager für jeden dieser Dienste eine eigene SLM-Schnittstelle bereit.

Auf Implementierungsebene befindet sich ein QoS-Manager jeweils in der administrativen Domäne der ihm zugeordneten Geschäftskomponente. Damit wird sichergestellt, dass die Kommunikation zwischen dem QoS-Manager und der Implementierung der Geschäftskomponente nicht durch administrative Barrieren erschwert wird. Weiterhin wird so erreicht, dass der Betreiber einer Geschäftskomponente zugleich den zugehörigen QoS-Manager kontrolliert und somit Einfluss auf die durch den QoS-Manager zu treffenden Entscheidungen ausüben kann.

Die Überwachungs- und Steuerungsaufgaben des QoS-Managers machen eine enge Kopplung notwendig, die häufig die Kommunikation über nicht-öffentliche Schnittstellen voraussetzt. Die Kommunikation zwischen QoS-Manager und Geschäftskomponente ist damit nicht auf die als Dienst in der SOA offerierten Schnittstellen beschränkt, d. h., zur Kommunikation können auch in der SOA nicht sichtbare Schnittstellen der Implementierung benutzt werden.

Da sich die Implementierungen von Geschäftskomponenten – und damit auch die verfügbaren Kommunikationsschnittstellen – erheblich unterscheiden können, erfolgt die Kommunikation zwischen QoS-Manager und Implementierung der Geschäftskomponente mithilfe von Protokolladaptern. Dadurch können die QoS-Manager-Funktionalität und Kommunikationsaspekte entkoppelt betrachtet werden.

5.3.2 Integration auf Basis eines SOA-Komponentenmodells

Wie in Abbildung 5.2 dargestellt, besteht aus SOA-Sicht zunächst keine sichtbare Verbindung zwischen der Geschäftsschnittstelle eines Dienstes und der zugehörigen, durch den der Geschäftskomponente zugeordneten QoS-Manager bereitgestellten SLM-Schnittstelle. Eine solche Verbindung kann auf unterschiedliche Weise realisiert werden: Sie kann implizit erfolgen, z. B. durch Vergabe von URIs, die bestimmten Namenskonventionen folgen, oder explizit durch eine mithilfe eines geeigneten SOA-Komponentenmodells modellierte Verknüpfung ausgedrückt werden. Der letztgenannte Ansatz ist hierbei vorzuziehen, da nur durch eine explizite Modellierung von Strukturen sichergestellt werden kann, dass auch Komponenten Dritter in geeigneter Weise mit der Architektur interagieren können (vgl. Anforderung **I4** *(Offenheit)*).

In Abschnitt 2.6 wurde die Service Component Architecture als Beispiel für ein SOA-Komponentenmodell vorgestellt. Im Folgenden wird die explizite Modellierung der Verknüpfung von Geschäftsschnittstelle und SLM-Schnittstelle einer Komponente auf Basis von SCA beschrieben.

Für die Abbildung der SLM-Architektur auf SCA wird angenommen, dass die zu verwaltenden Geschäftskomponenten SCA-konform definiert sind und so im Modell adäquat durch eine SCA-Komponente bzw. ein SCA Composite repräsentiert werden können.

Abbildung 5.3 zeigt, wie ein SCA-konformer QoS-Manager als SCA-Komponente modelliert wird. Er kann so problemlos in eine SCA-basierte SOA eingebunden werden. Nach außen stellt der SCA-basierte QoS-Manager seine SLM-Schnittstelle als Service `M` zur Verfügung, für die Dienstnutzung statisch vereinbarte SLAs werden als SCA Properties modelliert.

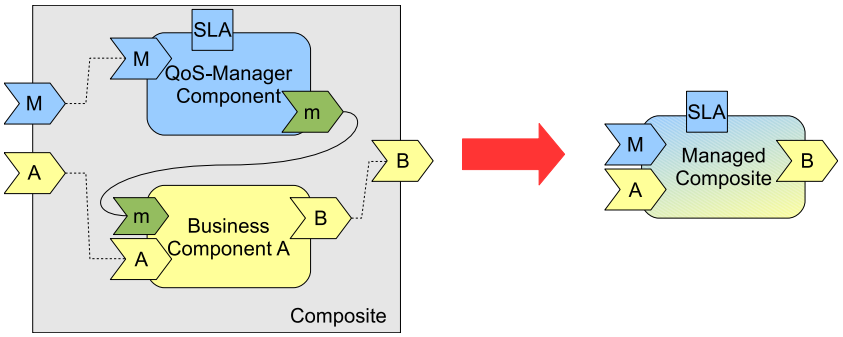

Abbildung 5.3: Einbettung eines QoS-Managers in ein SCA Composite

Die Kommunikation des QoS-Managers mit der Implementierung der zu verwaltenden Geschäftskomponente kann in SCA ebenfalls explizit modelliert werden: die Geschäftskomponente offeriert einen Management Service `m`, der durch den QoS-Manager referenziert wird (vgl. Abbildung 5.3). In der Regel ist eine Modellierung in diesem Detaillierungsgrad jedoch nicht notwendig und häufig auch nicht erwünscht da der Zugriff auf die Management- und Monitoring-Schnittstellen der zu verwaltenden Geschäftskomponenten normalerweise auf wenige Kommunikationspartner aus der lokalen administrativen Domäne eingeschränkt ist.

Um den QoS-Manager dauerhaft an eine SCA-Komponente zu binden und gleichzeitig dessen Existenz für die Referenzen anderer Komponenten transparent zu halten, werden die zu verwaltende Geschäftskomponente und ihr QoS-Manager in ein SCA Composite eingebettet, das sowohl den QoS-Management-Service `M` als auch den durch die Geschäftskomponente erbrachten Service `A` nach außen propagiert. Innerhalb der SOA werden alle Referenzen auf die verwaltete Komponente durch das neu erzeugte Composite substituiert. In dieser Arbeit wird ein solcher Verbund aus Manager und Komponente als *Managed Composite* bezeichnet.

Im Folgenden werden SLM-Schnittstelle und funktionale Struktur von QoS-Managern näher beschrieben.

5.4 Funktionale Struktur von QoS-Managern

Die durch einen QoS-Manager zu erbringende Management-Funktionalität hängt, wie in Abschnitt 5.2 skizziert, primär davon ab, ob der QoS-Manager einem SOA-Dienst oder einer Workflow-Engine zugeordnet ist. Unabhängig von der Zuordnung zu einer bestimmten Geschäftskomponente kann jedoch eine einheitliche SLM-Schnittstelle für alle QoS-Manager definiert werden.

Die in der SLM-Architektur zum Einsatz kommenden QoS-Manager für Workflows und für Dienste werden im Rahmen der Umsetzung der Architektur technisch auf ein gemeinsames modulares Manager-Framework abgebildet. Die Architektur des Frameworks gewährleistet dabei die problemlose Integration von Schnittstellen zur Anbindung an ein WfMS bzw. an die Monitoring- und Management-Schnittstellen unterschiedlicher Dienst-Implementierungen.

Das Framework wird im Detail in Abschnitt 7.1 vorgestellt. Dort wird auch beispielhaft die Realisierung eines QoS-Managers für eine Workflow Engine und einen konkreten Dienst besprochen.

Im Weiteren wird zunächst die in der Architektur definierte SLM-Schnittstelle vorgestellt, bevor genauer auf die funktionale Struktur der unterschiedlichen QoS-Manager-Typen eingegangen wird.

5.4.1 Die SLM-Schnittstelle

Die SLM-Schnittstelle eines QoS-Managers dient der Kommunikation mit anderen Diensten innerhalb der SOA. Sie bildet damit die in Abbildung 5.2 skizzierte Dienst-Schnittstelle des QoS-Managers auf SOA-Ebene.

5.4.1.1 Vorgaben für die zu erbringende Funktionalität

Aus den in Abschnitt 5.1 skizzierten Anforderungen **F2** *(Entgegennahme von Dienstgüteanforderungen)* und **F5** *(Eskalation)* ergeben sich konkrete Vorgaben für die durch die SLM-Schnittstelle bereitzustellende Funktionalität:

SLM-I1 *Bekanntgabe von Rahmenbedingungen für die SLA-Vereinbarung*
Der QoS-Manager ermöglicht einem Kommunikationspartner, vorab die Rahmenbedingungen für die Nutzung der Geschäftsfunktionalität in Erfahrung zu bringen. Hierzu gehören sowohl vom QoS-Manager akzeptierte Wertebereiche für einzelne SLA-Parameter als auch Kosten für die Nutzung des Dienstes.

SLM-I2 *Vereinbarung von SLAs*
Über die SLM-Schnittstelle werden mit einem Kommunikationspartner – basierend auf den zuvor bekannt gegebenen Rahmenbedingungen – konkrete Dienstgütevereinbarungen zur Nutzung der Geschäftsfunktionalität geschlossen. Die Schnittstelle ermöglicht dem QoS-Manager das Akzeptieren oder Ablehnen von durch den Kommunikationspartner gemachten Vorschlägen.

SLM-I3 *Statusabfrage*
Der Kommunikationspartner hat jederzeit die Möglichkeit, sich über den Status der Einhaltung eines geschlossenen SLAs und deren SLOs zu informieren. Die SLM-Schnittstelle offeriert eine entsprechende Funktionalität.

SLM-I4 *Verwaltung des Lebenszykluses von SLAs*
Neben der in **SLM-I3** *(Statusabfrage)* beschriebenen Möglichkeit zur Statusabfrage bietet die SLM-Schnittstelle beiden Kommunikationspartnern die Möglichkeit, Einfluss auf den Lebenszyklus einer getroffenen Vereinbarung zu nehmen und z. B. ein SLA vorzeitig zu kündigen. Dadurch eventuell anfallende Vertragsstrafen können bereits vorab im Rahmen der SLA-Aushandlung vereinbart werden.

SLM-I5 *Eskalation von Problemen*
Beim Auftreten von für den QoS-Manager nicht lokal lösbaren Problemen, die Auswirkungen auf das Erfüllen der zuvor gegebenen Dienstgütegarantien haben, werden die Vertragspartner aktiv durch den QoS-Manager benachrichtigt.

Innerhalb der Architektur wird eine einheitliche SLM-Schnittstelle durch alle QoS-Manager angeboten, die die oben beschriebenen Vorgaben erfüllt. Die SLM-Schnittstelle eines QoS-Managers kann mithilfe eines entsprechenden Werkzeugs zur SLA-Vereinbarung durch einen Administrator genutzt werden, sie wird gleichzeitig auch durch andere QoS-Manager verwendet und kann auch durch andere Managementsoftware genutzt werden.

QoS-Manager, die mit anderen QoS-Managern zum Zwecke der Vereinbarung von SLAs kommunizieren, implementieren zusätzlich die entsprechende Client-Schnittstelle.

5.4.1.2 Nutzung von WS-Agreement

Die in Abschnitt 3.2.3.1 beschriebene WS-Agreement-Architektur definiert ein Protokoll zur maschinellen Vereinbarung von SLAs, das die Vorgaben **SLM-I1** bis **SLM-I5** erfüllt:

Die Anforderung **SLM-I1** zur Bekanntgabe von Rahmenbedingungen für die Vereinbarung eines SLAs kann in WS-Agreement auf das Konzept der Agreement Templates abgebildet werden. Der Anbieter eines Dienstes kann parallel mehrere Templates offerieren, die als Grundlage für ein zu vereinbarendes SLA genutzt werden können. Auf diese Weise kann die Nutzung eines Dienstes z. B. parallel in unterschiedlichen Dienstgüteklassen zu entsprechenden Preisen angeboten werden. Der Agreement Initiator holt sich eine Liste verfügbarer Agreement

Templates und kann bei der Abgabe seines Angebots das Template referenzieren, das diesem zugrunde liegt.

Die in Anforderung **SLM-I2** geforderte Funktionalität zur Aushandlung bindender SLAs ist Kernbestandteil des WS-Agreement-Protokolls. Nach Prüfung des durch einen Agreement Initiator erstellten Agreement Offers kann der QoS-Manager entscheiden, ob eine entsprechende Vereinbarung eingegangen werden soll oder nicht. Die möglichen Zustandsübergänge eines Agreements sind in Abbildung 3.6 dargestellt.

Die Anforderungen **SLM-I3** *(Statusabfrage)* und **SLM-I4** *(Verwaltung des Lebenszykluses von SLAs)* können ebenfalls durch WS-Agreement abgedeckt werden, da in der Spezifikation sowohl Funktionen zur detaillierten Statusabfrage als auch zum vorzeitigen Terminieren eines SLAs vorgesehen sind.

Eine aktive Ereignis-Eskalation (vgl. Anforderung **SLM-I5**) kann in WS-Agreement unter Nutzung der WS-Notification-Spezifikation [OAS06b] erfolgen, deren Verwendung in Kombination mit WS-Agreement durch das Global Grid Forum ausdrücklich empfohlen wird.

Weitere Argumente für den Einsatz von WS-Agreement im Rahmen der SLM-Architektur sind die Interoperabilität mit anderen Lösungen, die zur SLA-Aushandlung ebenfalls auf das standardisierte WS-Agreement setzen (und damit eine leichtere Integration in bestehende Management-Systeme) sowie die Nutzung des Web-Services-Protokoll-Stacks durch WS-Agreement. Dadurch bleibt in einer mithilfe von WS-BPEL und Web Services realisierten SOA auf Implementierungsebene eine einheitliche Protokolllandschaft erhalten. Letzteres ist insbesondere in heterogenen Umgebungen relevant, die sich über administrative Grenzen hinweg erstrecken, weil hier der Einsatz zusätzlicher Transportprotokolle in der Regel die Rekonfiguration von Firewall-Systemen etc. notwendig machen würde.

5.4.2 QoS-Manager für Workflows

5.4.2.1 Überblick

Ein einer WfMS-Instanz zugeordneter QoS-Manager ist für die Überwachung und Durchsetzung von SLAs verantwortlich, die mithilfe der in Abschnitt 5.4.1 beschriebenen SLM-Schnittstelle für die durch das WfMS auszuführenden Workflows vereinbart werden.

Abbildung 5.4 zeigt die logischen Funktionsabläufe in einem QoS-Manager für Workflows. Neben der Überwachung der Einhaltung bereits geschlossener SLAs (*Prozessüberwachung*) ist die Zuteilung individueller SLOs zu einzelnen an einem Workflow beteiligten Diensten die zweite Hauptaufgabe des QoS-Managers (*SLO-Zuteilung und -Aushandlung*).

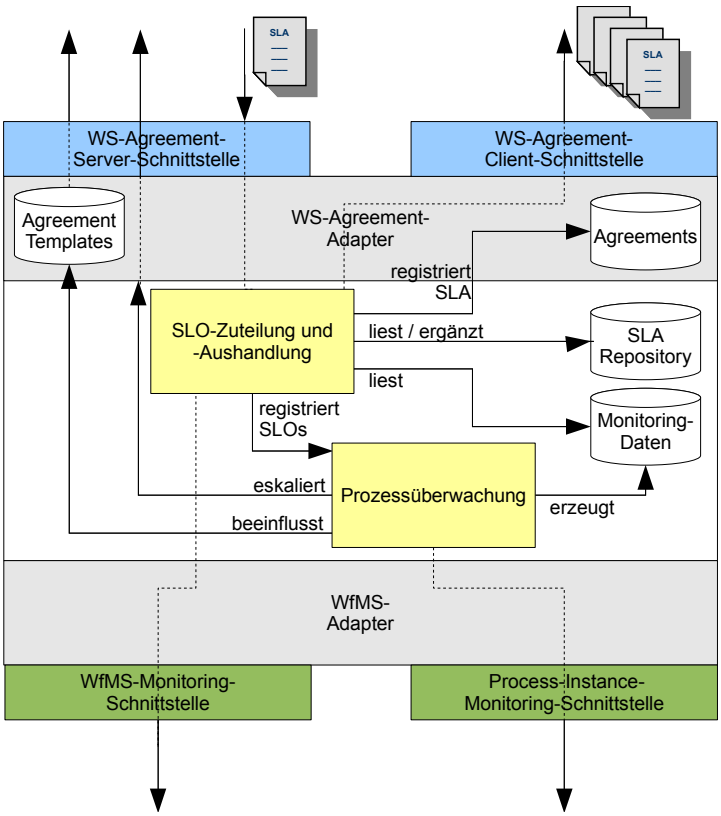

Abbildung 5.4: Funktionsabläufe in einem QoS-Manager für Workflows

Der QoS-Manager verfügt über Adapter, die die Anbindung an externe Ressourcen kapseln. Auf Seiten der SLM-Schnittstelle existiert ein *WS-Agreement-Adapter*, der die Kommunikation mit externen Partnern über das WS-Agreement-Protokoll abwickelt und geeignete Transformationen zwischen intern verwendeten Datenformaten und WS-Agreement vornimmt. Gleichzeitig kapselt der Adapter auch die WS-Agreement-Client-Schnittstelle, über die mit den SLM-Schnittstellen anderer QoS-Manager kommuniziert werden kann.

Weiterhin verfügt der QoS-Manager über einen *WfMS-Adapter*, der die Spezifika der Anbindung an ein konkretes WfMS verbirgt. Mithilfe des WfMS-Adapters werden zum einen das Sammeln von Performance-Monitoring-Daten zur Laufzeit und zum anderen der Zugriff auf Konfigurationsinformationen des WfMSs (beispielsweise zum Auslesen der aktiven Prozesse) realisiert.

Der QoS-Manager verfügt über eine Reihe von Repositories zur Ablage von Informationen: Im WS-Agreement-Adapter werden verfügbare SLA Templates und geschlossene SLAs verwaltet, zusätzlich existiert ein Repository mit Informationen zu nicht veränderlichen SLAs einzelner Dienste. In einem weiteren Repository werden Monitoring-Daten abgelegt, die im WfMS erhoben wurden. Die historischen Monitoring-Daten werden – genauso wie die In-

formationen über unveränderliche SLAs – bei der Definition von SLOs für einzelne Dienste herangezogen.

Funktional können die SLM-Aufgaben des QoS-Managers wie folgt unterteilt werden:

- initiale SLO-Zuteilung und -Aushandlung
- Prozessüberwachung

Diese werden im Folgenden genauer betrachtet.

5.4.2.2 Initiale SLO-Zuteilung und -Aushandlung

Die initiale SLO-Zuteilung und -Aushandlung wird immer dann aktiv, wenn an den QoS-Manager ein neues SLA herangetragen wird. Im Rahmen der SLO-Zuteilung werden mit den an einem WS-BPEL-Prozess beteiligten Diensten individuelle SLOs ausgehandelt, die in der Summe die Erfüllung des an den Prozess gestellten SLAs sicherstellen. Nach dieser initialen Phase der Aushandlung wird eine Komponente mit der Überwachung des für einen Prozess vereinbarten SLAs beauftragt.

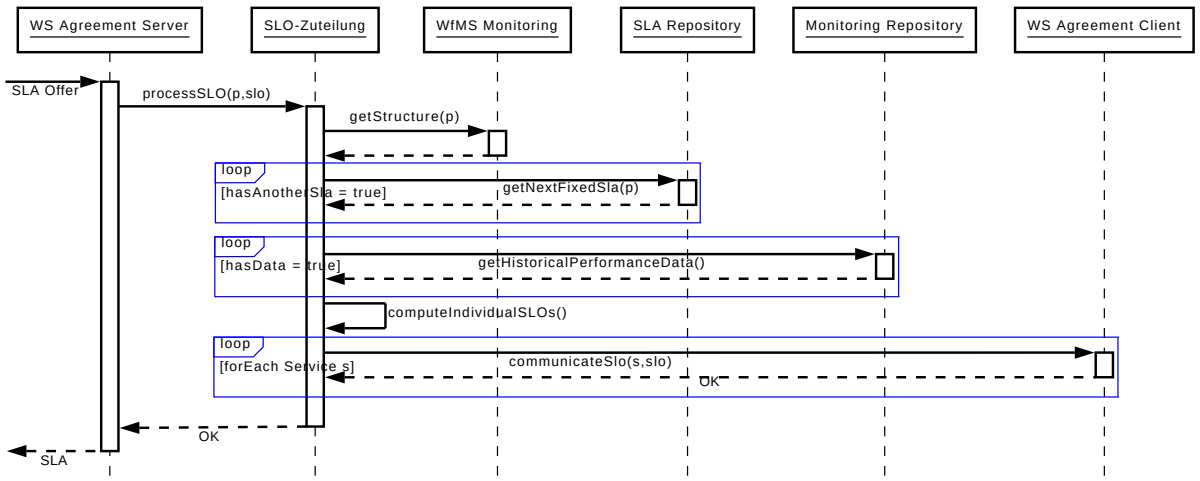

Abbildung 5.5: SLA-Annahme und SLO-Zuteilung in einem QoS-Manager für Workflows

Abbildung 5.5 zeigt den prinzipiellen Ablauf der Aushandlung individueller SLOs für die an einem Workflow beteiligten Dienste. Zunächst wird dem QoS-Manager über seine SLM-Schnittstelle das Agreement Offer-Dokument eines Kommunikationspartners zugestellt. Innerhalb des WS-Agreement-Adapters wird das Angebot zunächst auf seine Konformität zu den im Repository abgelegten SLA-Templates überprüft und bei Erfolg an die SLO-Zuteilungsfunktion weitergeleitet.

Die initiale Definition von SLOs für einzelne Dienste basiert auf einer Analyse der Struktur des WS-BPEL-Prozesses, in die unveränderliche SLAs und historische Monitoring-Informationen mit eingehen. Zunächst wird der durch das SLA referenzierte WS-BPEL-Prozess über den WfMS-Adapter eingelesen und analysiert. Ist im WfMS kein entsprechender Prozess bekannt, so wird das Agreement Offer abgewiesen. Im Anschluss werden nicht veränderliche SLAs und Monitoring-Daten der Dienste angefordert, die durch den WS-BPEL-Prozess referenziert werden. Auf dieser Basis werden dann SLOs für die einzelnen Dienste bestimmt. Der für die Zerlegung und initiale SLO-Aushandlung entwickelte Algorithmus ist im Detail in Abschnitt 6.7 beschrieben.

Im Anschluss werden über die WS-Agreement-Client-Schnittstelle entsprechende SLAs mit den QoS-Managern der beteiligten Dienste geschlossen. Gelingt die Aushandlung der SLAs nicht oder erlauben die Rahmenbedingungen die Vereinbarung eines SLAs mit den geforderten Dienstgüteanforderungen nicht, weist der QoS-Manager das Agreement Offer ab. Ansonsten wird die Prozessüberwachungskomponente über das Agreement Offer informiert und das SLA akzeptiert.

5.4.2.3 Prozessüberwachung

Das Prozessüberwachungsmodul (vgl. Abbildung 5.4) ist für die Überwachung der zuvor für WS-BPEL-Prozesse vereinbarten SLAs verantwortlich. Für jeden in einem WfMS registrierten Workflow, dem ein SLA zugeordnet ist, wird ein separates Prozessüberwachungsmodul instanziiert. Damit können gleichzeitig SLAs für unterschiedliche Workflows überwacht werden.

Bei der Prozessüberwachung werden die Ausführungsschritte der einzelnen Workflow-Instanzen überwacht: Für jeden Dienstaufruf wird die Antwortzeit aus Sicht des WfMS ermittelt und für jede Workflow-Instanz wird die Gesamtantwortzeit gemessen. Weiterhin wird festgehalten, ob die Workflow-Abarbeitung erfolgreich oder mit einem Fehler abgeschlossen wurde. Die erfassten Messwerte werden im Repository für Monitoring-Daten abgelegt, zusätzlich werden grundlegende statistische Größen über einen konfigurierbaren Zeitraum erhoben. Im Einzelnen sind dies Minimum, Maximum, Mittelwert und Standardabweichung der Bedienzeit.

Im Rahmen des Monitorings stellt das Prozessüberwachungsmodul ggf. Verstöße gegen das global für einen Prozess vereinbarte SLO fest. Tritt ein SLO-Verstoß auf, so benachrichtigt das Prozessüberwachungsmodul über den WS-Agreement-Adapter den zuständigen Vertragspartner. Gleichzeitig erhält die Prozessüberwachung ihrerseits ggf. Eskalationsmeldungen einzelner beteiligter Dienste. Diese werden ebenfalls im Repository für Monitoring-Daten protokolliert.

SLO-Verstöße einzelner Dienste werden für die Überwachung des globalen SLAs ignoriert, solange diese sich nicht kritisch auf die für den Gesamtprozess garantierte Dienstgüte auswirken. Überschreitungen der Antwortzeit durch einen Dienst können u. U. im Verlauf des

Workflows durch die schnellere Abarbeitung eines anderen Dienstes oder durch Wahl eines schnelleren Pfades im Workflow ausgeglichen werden. Durch diese vorsichtige Reaktion auf das Überschreiten der Antwortzeit durch einzelne Dienste werden Überreaktionen des SLM-Systems vermieden.

Basierend auf den erhobenen Monitoring-Daten kann die Prozessüberwachung regelmäßig die durch den QoS-Manager zur Verfügung gestellten Agreement Templates aktualisieren und die Rahmenbedingungen für zukünftige SLAs definieren. Alternativ können diese Templates manuell durch einen Administrator vorgegeben werden.

5.4.2.4 Verhalten in einer mehrschichtigen SOA

Die bisher beschriebene logische Sicht eines QoS-Managers für Workflows beschreibt lediglich die Funktionalität, die der QoS-Manager für die Verwaltung der Dienstgüte eines Workflows benötigt. Wird ein QoS-Manager in einer SOA-Umgebung eingesetzt, in der eine Hierarchie von Workflows etabliert ist, so müssen zusätzlich zu der bisher vorgestellten Funktionalität auch die im Folgenden für QoS-Manager für Dienste vorgestellten Kommunikationsfunktionen implementiert werden.

5.4.3 QoS-Manager für Dienste

5.4.3.1 Überblick

Ein einer Dienst-Implementierung zugeordneter QoS-Manager ist für die Durchsetzung und Überwachung von SLAs verantwortlich, die mithilfe der in Abschnitt 5.4.1 beschriebenen SLM-Schnittstelle für den Dienst vereinbart werden.

Abbildung 5.6 zeigt die logischen Funktionsabläufe in einem QoS-Manager für Dienste. Im Rahmen der *initialen SLO-Aushandlung* geht ein QoS-Manager mit seinen Kommunikationspartnern temporäre Dienstgütevereinbarungen ein. Die Rahmenbedingungen für akzeptable Dienstgütevereinbarungen ergeben sich aus den in einem Repository abgelegten *Agreement Templates*. Jedes neu abgeschlossene SLA wird zur Laufzeit an die Dienstüberwachungs-Komponente kommuniziert und im SLA Repository des QoS-Managers abgelegt.

Hauptaufgabe des QoS-Managers ist das Sicherstellen der Einhaltung zuvor vereinbarter SLAs durch die funktionale Komponente *Dienstüberwachung*. Die Dienstüberwachung interagiert zur Durchsetzung der SLOs direkt mit einem komponentenspezifischen *Management Controller*, der über *Monitoring*- und *Control-Schnittstellen* auf die zu verwaltende Dienst-Implementierung zugreift. Die Dienstüberwachung ist damit direkt an den SLM-Regelkreis der Dienst-Implementierung angebunden. Eine *Strategie-Optimierungskomponente* überwacht wiederum die durch diesen Regelkreis erzielte Dienstgüte und interagiert mit anderen QoS-

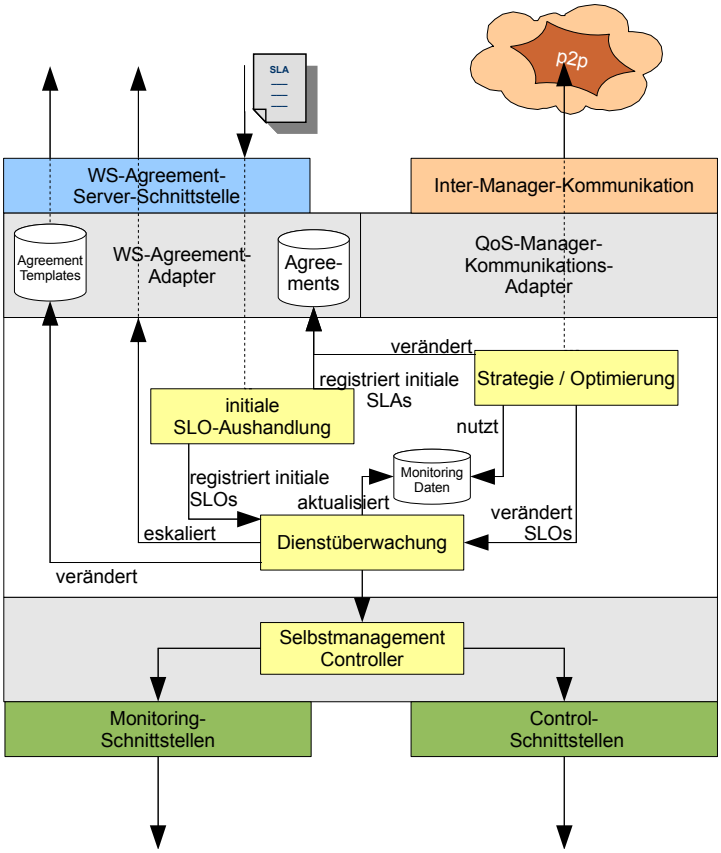

Abbildung 5.6: Funktionsabläufe in einem QoS-Manager für Dienste

Manager-Strategiekomponenten zur komponentenübergreifenden Optimierung des Dienstgüteverhaltens der Architektur.

Analog zum QoS-Manager für Workflows verfügt der einer Dienst-Implementierung zugeordnete QoS-Manager über Adapter, die die Anbindung an externe Ressourcen kapseln. Auf Seiten der SLM-Schnittstelle existiert ein *WS-Agreement Adapter*, der die Kommunikation mit externen Partnern über das WS-Agreement-Protokoll abwickelt und geeignete Transformationen zwischen intern verwendeten Datenformaten und WS-Agreement vornimmt. Über einen weiteren Adapter kann die Strategie-Optimierungskomponente des QoS-Managers direkt mit anderen QoS-Managern zum Zwecke der dezentralen Optimierung des Dienstgüteverhaltens kommunizieren.

5.4.3.2 Beeinflussung des Dienstgüteverhaltens von SOA-Komponenten

Zur Optimierung des Dienstgüteverhaltens von SOA-Komponenten existieren unterschiedliche Ansatzpunkte: Einerseits kann mit dem Ziel der Verbesserung der Dienstgüte in geeigneter

Weise auf die Dienst-Implementierung und durch sie genutzte Ressourcen eingewirkt werden, andererseits können die Rahmenbedingungen für die Erbringung der Dienstgüte durch Kommunikation und Kooperation mit anderen Komponenten verbessert werden.

Diese Ansatzpunkte werden im Folgenden genauer betrachtet.

Lokale Überwachung und Durchsetzung von Dienstgüteanforderungen

Das Einwirken auf Dienst-Implementierung und Ressourcen ist der üblicherweise zur Realisierung eines SLM-Systems gewählte Ansatz; der für die Durchsetzung der Management-Ziele etablierte Regelkreis optimiert das Dienstgüteverhalten einer Komponente nach lokalen Kriterien. Der Eingriff in das verwaltete System kann dabei auf unterschiedliche Weise erfolgen:

- Rekonfiguration der Dienst-Implementierung
- Rekonfiguration zur Verfügung stehender Ressourcen
- Verfügbarmachen zusätzlicher Ressourcen

Die Management-Komponente muss bei der Rekonfiguration nicht zwischen Dienst-Implementierung und Ressourcen unterscheiden – aus Managementsicht bilden beide gemeinsam das zu verwaltende System, das dem Management geeignete Konfigurationsschnittstellen offeriert. Die Möglichkeiten zur Rekonfiguration von Ressourcen sind in der Regel sehr limitiert, sodass der Manager bei auftretenden Engpässen oftmals versucht, zusätzliche Ressourcen verfügbar zu machen. Heute ergeben sich allerdings durch die Zunahme virtualisierter Umgebungen erweiterte Möglichkeiten zur Rekonfiguration auf Ressourcen-Ebene. Hier können beispielsweise zur Laufzeit Änderungen an Speichergröße oder (virtueller) CPU-Anzahl und -Anteilen (Shares) vorgenommen werden.

Die Beeinflussung der Konfiguration des zu verwaltenden Systems geschieht im QoS-Manager durch den implementierungsabhängigen Selbst-Management-Controller, dessen Zielvorgaben von der Prozessüberwachungskomponente getroffen werden. Im Folgenden wird beispielhaft gezeigt, auf welche Weise der QoS-Manager auf die durch eine Komponente erbrachte Dienstgüte Einfluss nehmen kann.

> **Beispiel 5.1:** [DSK05] beschreibt das Dienstgüte-Management eines Clusters von J2EE-Applikationsservern mithilfe eines Selbst-Management-Controllers. Überschreitet die auf dem Cluster laufende Applikation eine definierte maximale Antwortzeit, startet der Selbst-Management-Controller zusätzliche Instanzen des Applikationsservers auf separaten physikalischen Maschinen, bei geringer Last werden ungenutzte Instanzen beendet. Der Kontrollalgorithmus ist als Zustandsautomat realisiert, die Lastverteilung erfolgt über einen vorgeschalteten Load Balancer.

Dem in Beispiel 5.1 beschriebenen Kontrollalgorithmus liegen zwei Annahmen zugrunde, die gleichzeitig die Grenzen des Ansatzes aufzeigen: Zum einen wird davon ausgegangen, dass ein Zusammenhang zwischen der Antwortzeit der Applikation und der Bereitstellung von Ressourcen existiert, sodass das Verfügbarmachen zusätzlicher Ressourcen die Antwortzeit des Systems senken kann. Zum anderen setzt die Regelung voraus, dass die in das Cluster eingebundenen Ressourcen ausschließlich durch die verwaltete Applikation genutzt werden und nicht mit anderen, unbekannten Diensten geteilt werden müssen.

Zusammenhänge zwischen Antwortzeiten und Bereitstellung bestimmter Ressourcen (z. B. CPU, RAM, Netzwerk, ...) sind abhängig von der Art der betrachteten Anwendung. Zudem erfordern Überwachung und Einflussnahme auf das zu verwaltende System die Nutzung komponentenspezifischer Kenn- und Stellgrößen. Daher ist die individuelle Anpassung des Selbst-Management-Controllers an die zu verwaltende Komponente Voraussetzung für ein erfolgreiches lokales Selbst-Management.

Die Annahme, dass keine gemeinsame Nutzung von Ressourcen durch unterschiedliche Komponenten stattfindet, gilt für aktuelle dynamische Architekturen nur sehr eingeschränkt, da in einer SOA der Begriff der Ressource von der reinen Hardware auf gemeinsam genutzte Geschäftskomponenten ausgedehnt werden muss. Zur Beschreibung von Abhängigkeiten in einer SOA und die konkurrierende Nutzung von Ressourcen können Abhängigkeitsgraphen (vgl. [Deb05]) eingesetzt werden. Zur Optimierung der in einer solchen Architektur zu erbringenden Dienstgüte ist die lokale Betrachtung isolierter Komponenten somit nicht ausreichend. Im Folgenden werden deshalb im QoS-Manager ergänzend zum Einsatz kommende Ansätze zur komponentenübergreifenden Optimierung vorgestellt.

Kommunikation mit anderen QoS-Managern zur Optimierung der Dienstgüte

Zur übergreifenden Optimierung des Dienstgüteverhaltens von SOA Workflows kommunizieren die Strategie-Optimierungskomponenten der in einer SOA aktiven QoS-Manager direkt miteinander:

- QoS-Manager innerhalb des gleichen Workflows kooperieren zur Einhaltung des für den Workflow vereinbarten SLOs.
- QoS-Manager, deren Geschäftskomponenten gemeinsame Ressourcen nutzen, kommunizieren mit dem Ziel der Lösung von Konflikten.

Als Ergebnis der Kommunikation kooperierender QoS-Manager modifiziert die Strategie-Optimierungskomponente des QoS-Managers die in den abgeschlossenen SLAs enthaltenen SLOs. Diese dienen wiederum als Grundlage für zukünftige Entscheidungen der Dienstüberwachungskomponente.

Konflikte werden durch Veränderungen in der Ressourcenzuordnung gelöst, die sich an der für einzelne Workflows festgelegten Priorität orientieren.

Abbildung 5.7 zeigt den logischen Aufbau der Kommunikationsschnittstelle eines QoS-Managers. Abhängig von den zu erreichenden Zielen und Kommunikationspartnern werden unterschiedliche Kommunikations-Plugins zum Nachrichtenaustausch mit anderen Komponenten verwendet. Die Kommunikations-Plugins bieten dem QoS-Manager eine abstrakte Schnittstelle zur Kommunikation mit anderen Komponenten, sie sind aber gleichzeitig in die unterlagerte Kommunikationsinfrastruktur eingebettet.

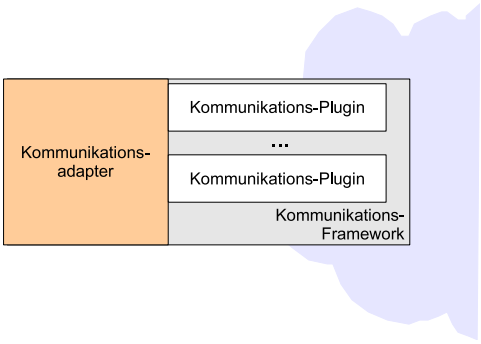

Abbildung 5.7: Kommunikationsschnittstelle eines QoS-Managers

Die Entwicklung geeigneter Verfahren zur selbstorganisierten Kommunikation von QoS-Managern bildet einen zentralen Aspekt dieser Arbeit und ist detailliert in Kapitel 6 beschrieben. Die dort vorgestellten Verfahren werden auf entsprechende Kommunikations-Plugins abgebildet.

5.4.3.3 Behandlung konkurrierender Dienstgüteanforderungen

In der bisherigen Darstellung der Architektur wurde der Behandlung konkurrierender Dienstgüteanforderungen keine Beachtung geschenkt.

Konkurrierende Dienstgüteanforderungen können sich in der Praxis durch die parallele Einbindung eines Dienstes in unterschiedliche Workflows ergeben. Sie müssen durch den für die unterlagerte Dienstimplementierung zuständigen QoS-Manager geeignet behandelt werden. Für eine solche Behandlung existieren unterschiedliche Ansätze:

- Management von Komponenten mit QoS-Unterstützung
- Management von Komponenten ohne QoS-Unterstützung

Unterstützt die unterlagerte Dienst-Implementierung explizit unterschiedliche Dienstgüteklassen, so kann ein QoS-Manager gestellte Dienstgüteanforderungen direkt auf diese abbilden. Ein Beispiel für einen solchen Ansatz, der QoS-Unterstützung für Dienste implementiert, die auf Basis einer entsprechend erweiterten Web-Services-Plattform realisiert wurden, findet sich in [DLP03].

Unterstützt der verwaltete Dienst keine Dienstgüteklassen, kann dies durch einen der Implementierung vorgeschalteten Proxy-Dienst realisiert werden, der im Folgenden als *QoS-Proxy* bezeichnet wird. Die Einbindung eines solchen QoS-Proxys in die Architektur ist in Abbildung 5.8 beispielhaft für einen einfachen Dienst skizziert, das Verfahren kann aber unverändert auch für komplexe Workflows angewandt werden.

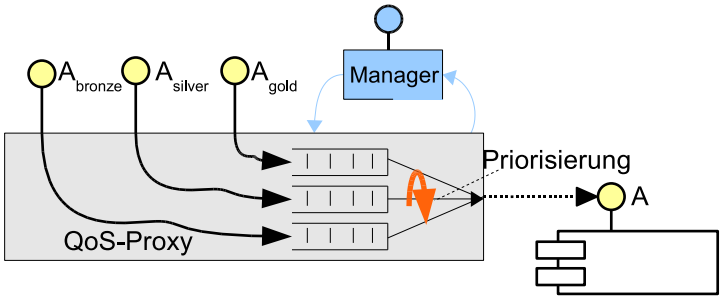

Abbildung 5.8: Über einen QoS-Proxy werden Zugriffe auf einen Dienst geregelt

Der Proxy ist, wie in Abbildung 5.8 dargestellt, als Dienst realisiert, der für jede zu erbringende Dienstgüteklasse c (in der Abbildung mit $c \in \{bronze, silver, gold\}$) eine eigene Schnittstelle A_c bereitstellt, die inhaltlich der Schnittstelle A des nachgelagerten Dienstes entspricht. Anfragen, die an diesen Schnittstellen eingehen, werden zunächst in eine Prioritätswarteschlange eingereiht und, dann entsprechend den durch den Manager gesetzten QoS-Vorgaben, an den eigentlich aufgerufenen Dienst weitergeleitet. Zur Priorisierung und Regelung der Dienstgüte von Anfragen müssen dabei an die Art der Dienstgütedimension angepasste Algorithmen eingesetzt werden. Für den aufrufenden Dienst selbst ist die Existenz des Proxys transparent. Voraussetzung für die erfolgreiche Umsetzung eines solchen QoS-Proxy-Ansatzes ist die Möglichkeit, zur Laufzeit Referenzen auf Dienste aktualisieren zu können, beispielsweise auf Basis einer geeigneten Dienste-Registry.

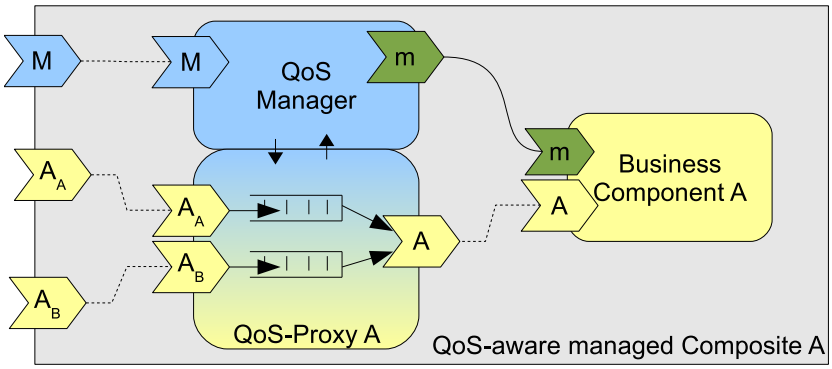

Abbildung 5.9: Realisierung eines QoS-Proxys innerhalb eines SCA Composites

Für die Verwaltung von Workflows kann das Proxy-Modell ebenfalls angewandt werden. Hierbei wird der durch das WfMS bereitgestellten Dienst-Schnittstelle eines Workflows eine entsprechende Proxy-Instanz vorgeschaltet. Alternativ kann aber auch direkt für jede Dienstgüteklasse eine Kopie des Workflows erzeugt und instanziiert werden.

Ein QoS-Proxy kann – wie in Abbildung 5.9 dargestellt – mithilfe von SCA realisiert werden. Hierbei wird für jede implementierte Dienstgüteklasse ein separater SCA Service bereitgestellt. Dienstaufrufe an einen Service werden zunächst in eine private Warteschlange innerhalb der Proxy-Komponente eingereiht. Entsprechend den vereinbarten Dienstgütekriterien werden die einzelnen Aufrufe dann aus den Warteschlangen entnommen und an den eigentlichen Dienst weitergeleitet.

5.5 Zusammenspiel der Management-Komponenten

Im Folgenden wird beispielhaft das Zusammenspiel der bisher präsentierten Architekturkomponenten für das Dienstgüte-Management von SOA-Komponenten vorgestellt. Es wird ein Szenario diskutiert, bei dem unterschiedliche Workflows parallel den gleichen Dienst nutzen.

In Abschnitt 5.5.1 werden die Abhängigkeiten zwischen Workflows und Diensten technologieunabhängig dargestellt. In Abschnitt 5.5.2 wird das Beispiel dann auf SCA-Komponenten abgebildet und um die in diesem Kapitel vorgestellten Management-Komponenten ergänzt.

5.5.1 Innerhalb der SOA modellierte Abhängigkeiten

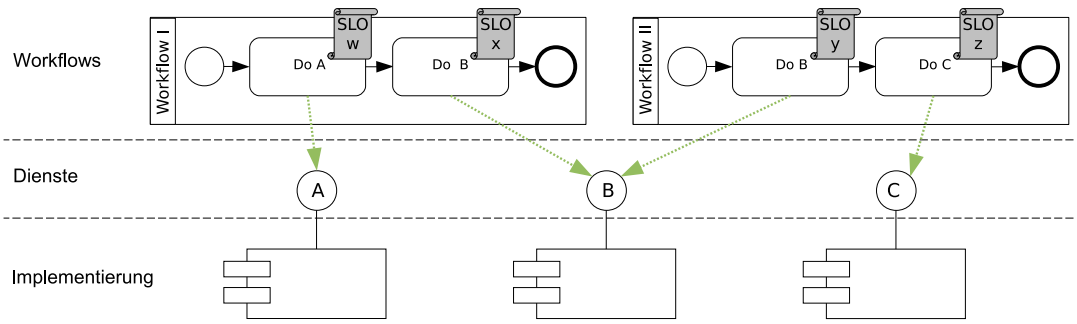

Abbildung 5.10: Gleichzeitige Nutzung eines Dienstes durch mehrere Workflows

In Abbildung 5.10 ist das Beispielszenario aus Anwendersicht dargestellt. Die Abbildung strukturiert die SOA in unterschiedliche Ebenen (vgl. Abschnitt 2.4). Auf der Geschäftsprozessebene sind zwei Workflows in BPMN-Notation abgebildet. Workflow I führt zunächst Aktivität A (in der Abbildung als `Do A` bezeichnet) und im Anschluss daran Aktivität B aus.

Workflow II führt nacheinander die Aktivitäten B und C aus. Den einzelnen Aktivitäten der Workflows sind individuelle Antwortzeit-SLOs zugeordnet. Jede Workflow-Aktivität steht für den Aufruf eines gleichnamigen Dienstes, was im Bild durch Verweise auf die darunterliegende Dienst-Ebene dargestellt ist. Auf der Dienst-Ebene der SOA existieren damit die drei durch die Workflows I und II referenzierten Dienste A, B und C. Diese sind auf der Implementierungsebene ohne weitere Abhängigkeiten voneinander realisiert.

Aus der Perspektive des Dienstgüte-Managements sind insbesondere die durch die Workflows formulierten disjunkten Anforderungen an den Dienst B von Interesse.

5.5.2 Darstellung der Abhängigkeiten mithilfe von SCA

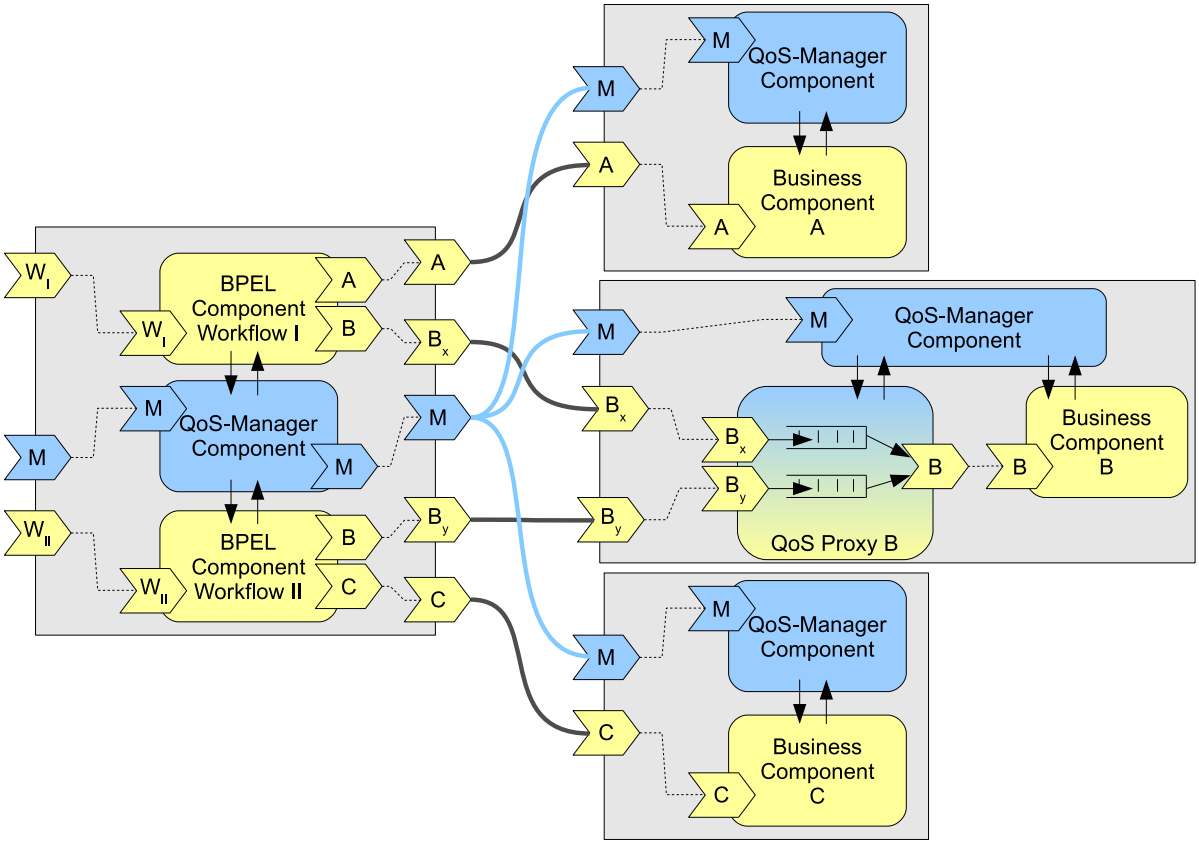

Abbildung 5.11: Abbildung des Szenarios auf die Service Component Architecture

Abbildung 5.11 zeigt die Abbildung des in Abschnitt 5.5.1 beschriebenen Szenarios auf einzelne SCA-Komponenten. Im Folgenden wird zunächst auf die Modellierung der in der Abbildung links dargestellten Workflows eingegangen. Im Anschluss wird die Modellierung der durch diese referenzierten Dienste besprochen.

5.5.2.1 Modellierung von Workflows

Im Beispiel wird angenommen, dass Workflow I und Workflow II durch die gleiche WfMS-Instanz bereitgestellt werden, sodass ein einzelner Workflow-QoS-Manager (vgl. Abschnitt 5.4.2) die Instanzen beider Workflows überwachen kann. Die Workflows und der dem WfMS zugeordnete QoS-Manager können als SCA Composite modelliert werden. Das Composite definiert eine Reihe von SCA Services und SCA References:

- Jeder Workflow definiert einen Service, über den er aufgerufen werden kann. Diese sind in der Abbildung als W_I und W_{II} bezeichnet.
- Die durch die Aktivitäten eines Workflows referenzierten Dienste sind als SCA References modelliert. Zur besseren Unterscheidbarkeit ist in der Abbildung für die References auf den Dienst B die jeweils geforderte Dienstgüte als Index angegeben.
- Ein Management Service `M` erlaubt das Aushandeln von SLA-Vereinbarungen mit dem QoS-Manager des WfMSs.
- Eine Referenz auf die Management Services der beteiligten Dienste definiert die Kommunikationsschnittstellen zur Aushandlung der SLOs zur Dienstnutzung.

Aus Gründen der Übersichtlichkeit werden die Management References des Composites in Abbildung 5.11 gruppiert dargestellt.

5.5.2.2 Modellierung von Diensten

Die Dienste A, B und C sind ebenfalls als SCA-Komponenten modelliert. Jeder Dienst-Implementierung ist ein QoS-Manager für Dienste (vgl. Abschnitt 5.4.3) zugeordnet. QoS-Manager und Dienst werden zu einem SCA Composite gruppiert. Der Implementierung des Dienstes B ist die in Abschnitt 5.4.3.3 vorgestellte Proxy-Komponente vorgeschaltet. Dadurch wird der Dienst B in die Lage versetzt, parallel unterschiedliche Dienstgüteanforderungen zu erfüllen.

Prinzipiell kann auch bei der Implementierung der Dienste A und C eine Proxy-Komponente zum Einsatz kommen. Im Beispiel werden diese Dienste jedoch nur einmal referenziert, sodass ihr Proxy jeweils nur eine einzige Dienstgüteklasse bereitstellen müsste.

Das einen Dienst repräsentierende SCA Composite definiert eine Reihe von SCA Services:

- Für die Geschäftsfunktionalität eines Dienstes wird in jeder nachgefragten Dienstgüteklasse ein SCA Service zur Verfügung gestellt.
- Zur Aushandlung von SLOs (und damit von Anforderungen für einzelne Dienstgüteklassen) wird zusätzlich ein Management Service `M` bereitgestellt.

Abhängigkeiten von anderen Diensten existieren im Beispiel nicht, diese würden aber beispielsweise bei der Modellierung von zusammengesetzten Diensten in Form von SCA References beschrieben.

5.5.2.3 Parallele Betrachtung unterschiedlicher Dienstgüteanforderungen

Parallele Nutzung eines SOA-Dienstes durch unterschiedliche Workflows stellt in einer SOA-Umgebung den Regelfall dar, sodass ein QoS-Manager gleichzeitig mit unterschiedlichen Dienstgüteanforderungen umgehen können muss. Durch Einsatz eines QoS-Proxys lassen sich – wie in Abschnitt 5.4.3.3 gezeigt – die unterschiedlichen Dienstgüteanforderungen bereitgestellten Dienstgüteklassen zuordnen. Als Resultat können die durch einen Workflow an den Dienst gestellten Dienstgüteanforderungen isoliert von anderen Dienstgüteklassen betrachtet werden.

Das nun folgende Kapitel 6 kann sich daher auf die Optimierung der durch das System erbrachten Dienstgüte in Bezug auf einen einzelnen Workflow und damit auf die Betrachtung einer einzelnen Dienstgüteklasse beschränken.

6 Selbstorganisation von QoS-Managern

6.1 Überblick

In diesem Kapitel werden mehrere Ansätze zur Optimierung des Dienstgüteverhaltens einer SOA zur Laufzeit vorgestellt: Durch die in einer solchen Umgebung mögliche konsistente Gesamtsicht auf die Anwendungslandschaft eines Unternehmens ergeben sich erhebliche Potenziale, da so Optimierungen nicht nur auf Dienste- oder Applikationsebene vorgenommen, sondern applikationsübergreifend verwirklicht werden können.

Ziel der im Folgenden vorgestellten Verfahren ist eine Verbesserung des globalen Dienstgüteverhaltens, die auf direkter Kommunikation und Kooperation einzelner QoS-Manager beruht. Diese ist nicht an durch Geschäftsprozesse vorgegebene Kommunikationsbeziehungen gebunden, sondern unterstützt den direkten Informationsaustausch von QoS-Managern, die zur Erreichung eines gemeinsamen Zieles kooperieren oder innerhalb der Architektur um Ressourcen konkurrieren. Dadurch wird die Grundlage für eine selbstorganisierende Optimierung des Dienstgüte-Managements gelegt, die die Skalierbarkeitsanforderung **N3** *(Skalierbarkeit)* erfüllt.

Allgemeine Voraussetzungen und Rahmenbedingungen für die im Weiteren vorgestellten Verfahren werden in Abschnitt 6.2 beschrieben.

Grundlage für eine dezentrale Optimierung des Dienstgüteverhaltens einer SOA bildet für jeden beteiligten QoS-Manager seine lokale Einschätzung der derzeit erreichten Dienstgüte. Abschnitt 6.3 beschreibt hier eine allgemeine Vorgehensweise dieser Einschätzung.

In Abschnitt 6.4 wird für die Dienstgütedimension Antwortzeit beschrieben, wie QoS-Manager der an einem Workflow beteiligten Dienste zur Einhaltung des für einen Workflow vereinbarten Antwortzeit-SLOs kooperieren: Innerhalb eines Workflows können Antwortzeit-Anteile zwischen Diensten übertragen werden, ohne dass das Gesamt-SLO dadurch beeinflusst wird. Die zur Koordination der Kommunikation verwendeten Verfahren werden in Abschnitt 6.5 beschrieben.

Globale, Workflow-übergreifende Optimierungen können innerhalb einer SOA im Bereich der Ressourcennutzung durchgeführt werden, da durch die einheitliche Modellierung der Anwendungslandschaft die Nutzung vorhandener Ressourcen über Anwendungsgrenzen hinweg betrachtet werden kann. Diese Optimierung kann durch Privilegierung von bestimmten Workflow-Typen bei der Ressourcennutzung erfolgen oder beispielsweise durch Verlagerung weniger wichtiger Aktivitäten auf leistungsschwächere Ressourcen. Abschnitt 6.6 beschreibt eine solche Vorgehensweise unter Nutzung von Ressourcen-Virtualisierung.

Voraussetzung für die vorgestellten Optimierungsverfahren ist eine initiale Verteilung des einem Workflow global zugeordneten SLOs auf die beteiligten Dienste. Abschnitt 6.7 skizziert einen Ansatz für die Verteilung von Antwortzeit-SLOs, der sich an durch die Geschäftsebene vorgegebenen Kommunikationsstrukturen orientiert.

6.2 Rahmenbedingungen für die Definition individueller SLOs

Wie in Abschnitt 2.2 dargestellt, können in einer dienstorientierten Architektur Komponenten aus unterschiedlichen administrativen Domänen kooperieren, woraus sich Einschränkungen für die Vereinbarung von SLAs zwischen einzelnen Komponenten ergeben können. Die für die Architektur formulierte Integrationsanforderung **I2** besagt, dass ein QoS-Manager auch mit Situationen umgehen können muss, in denen einige Dienste die Vereinbarung geeigneter SLAs verweigern. Im Einzelnen werden die folgenden Situationen unterschieden:

a) Die Nutzung eines Dienstes ist an ein vorab dauerhaft vereinbartes SLA gekoppelt.

b) Eine SLA-Vereinbarung ist mangels Unterstützung für QoS Management nicht möglich.

c) Ein Dienst akzeptiert das gewünschte SLA nicht,

   1) weil die gewählten Dienstgüteparameter nicht unterstützt werden.
   2) weil die gestellten Dienstgüteanforderungen außerhalb des aus Sicht des Dienstes akzeptablen Rahmens liegen.

Im Fall a) muss das dauerhaft vereinbarte SLA als unveränderlicher Bestandteil in die Berechnung der individuellen SLOs eingehen. Im Fall b) kann der QoS-Manager keine Anforderungen an den Dienst formulieren. Er kann evtl. Annahmen über das Dienstgüteverhalten des Dienstes treffen (z. B. aufgrund vorliegender historischer Daten) und kann diese Annahmen wie im Fall a) mit in die Verteilung einbeziehen. Im Fall c) muss unterschieden werden, ob der Dienst die gestellten Anforderungen prinzipiell nicht unterstützt (z. B. weil sein QoS-Manager keine maximale Antwortzeit oder keinen Mindestdurchsatz garantieren kann) oder ob die gestellten Anforderungen außerhalb dessen liegen, was der QoS-Manager des Dienstes zu akzeptieren bereit ist. Im ersten Fall muss ein solcher Dienst analog zu b) behandelt werden,

im zweiten Fall werden unterstützte Werte gewählt und der Dienst im Folgenden analog zu a) behandelt.

Ergebnis der Berechnungen zur Definition individueller SLOs kann jedoch sein, dass die Einhaltung der an den Workflow gestellten Dienstgüteanforderungen nicht garantiert werden kann. In einem solchen Fall errechnet der QoS-Manager des Workflows den maximal garantierbaren Wert und lehnt die gestellten QoS-Anforderungen ab. Er verhält sich damit selbst analog zum zweiten unter c) geschilderten Fall.

6.3 Bewertung der eigenen Managementsituation

6.3.1 Überblick

Grundlage für die selbst organisierte, dezentrale Optimierung einer SOA ist stets die lokale Bewertung des Verhaltens einer Komponente in Bezug auf die vereinbarten Dienstgüteziele durch den ihr zugeordneten QoS-Manager. Zur Bewertung des Grades der SLO-Erfüllung ermittelt ein QoS-Manager fortlaufend die erbrachte Dienstgüte qos seines (im Workflow durch eine `Invoke`-Aktivität $a \in A$ repräsentierten) Dienstes und berechnet dann die Vergleichsgröße Δslo wie folgt:

$$\Delta slo(a) = \mathrm{slo}(a) - \mathrm{qos}(a) \tag{6.1}$$

Basierend auf dem ermittelten Wert Δslo bewertet ein QoS-Manager die Erfüllung der ihm zugeordneten SLOs. Abbildung 6.1 zeigt eine Bewertungsskala für QoS-Werte, die der QoS-Manager zu Rate ziehen kann. Das durch den QoS-Manager eingegangene SLO wird als Zielwert der Skala interpretiert, durch Subtraktion von Δslo vom Zielwert wird der Ist-Zustand dargestellt. Zusätzlich können explizit Schwellwerte für obere und untere Grenzen definiert werden, die in der Grafik als l_i bzw. h_i dargestellt sind.

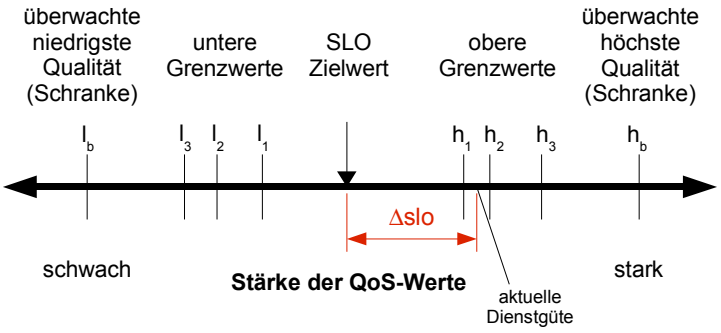

Abbildung 6.1: Schranken und Grenzwerte für QoS nach [Sti96]

Abhängig vom erreichten Ist-Wert und damit dem Über- oder Unterschreiten zuvor bestimmter Grenzwerte oder dem Erreichen eines Schwellwerts für Δslo greift der QoS-Manager dann ggf. korrigierend in das System ein um die lokale Managementsituation zu optimieren oder tritt mit anderen QoS-Managern in Kontakt.

6.3.2 Bewertung von Antwortzeitverhalten

Die Definition geeigneter Schwellwerte und korrigierender Aktionen ist stets abhängig von der betrachteten Dienstgütedimension. Im Weiteren konzentriert sich diese Arbeit auf die Dienstgütedimension Antwortzeit (vgl. Abschnitt 3.2.2.1). In den folgenden Abschnitten werden Verfahren zur Kooperation von QoS-Managern vorgestellt, die die Optimierung von Antwortzeit und Ressourcennutzung innerhalb eines Workflows anstreben.

Ein Beispiel für eine mögliche Optimierung der lokalen Managementsituation ist die Übertragung von Antwortzeit-Anteilen einer anderen Komponente im gleichen Workflow auf die eigene Komponente. Tabelle 6.1 zeigt beispielhaft, wie die lokale Bewertung bei Verwendung der Schwellwerte aus Abbildung 6.1 durchgeführt werden kann.

ZUSTAND VON a	BEWERTUNG	AKTIONEN
$\mathrm{qos}(a) < l_3$	Die Komponente erfüllt die an sie gestellten Anforderungen nicht.	Eskalation: z. B. Benachrichtigung eines Administrators
$l_3 \leq \mathrm{qos}(a) < l_2$		Versuch der Lockerung der Anforderungen
$l_2 \leq \mathrm{qos}(a) < l_1$		Zuordnung weiterer Ressourcen
$l_1 \leq \mathrm{qos}(a) < h_1$	Die Komponente erfüllt die an sie gestellten Anforderungen.	—
$h_1 \leq \mathrm{qos}(a) < h_2$	Die Komponente übererfüllt die an sie gestellten Anforderungen.	Reduktion der Ressourcennutzung
$h_2 \leq \mathrm{qos}(a) < h_3$		Versuch der Straffung der Anforderungen

Tabelle 6.1: Bewertung der Managementsituation und mögliche Aktionen

Grundlage für die in Tabelle 6.1 durchgeführte Bewertung ist, wie in Abschnitt 6.3.1 beschrieben, die Ermittlung von $\mathrm{qos}(a)$ und die Bestimmung vom Antwortzeit-SLO $\mathrm{sloResponse}(a)$ abhängiger Schwellwerte (vgl. Abschnitt 3.2.2.2). Bei Fokussierung auf die Dienstgütedimension Antwortzeit kann $\mathrm{qos}(a)$ mit der aktuellen Antwortzeit $t_{response}(a)$ des Dienstes gleichgesetzt werden.

Lokal durch Messungen ermitteln kann ein QoS-Manager jedoch nur die Bedienzeit $t_{work}(a)$, die zur Ermittlung von $t_{response}(a)$ notwendigen Übertragungszeiten sind ihm unbekannt (vgl. Formel 3.1).

Für die weiteren Betrachtungen wird in dieser Arbeit für die Summe der Übertragungszeiten t_{invoke} und t_{reply} ein konstanter Wert c angenommen. Damit kann entsprechend Formel 3.6 aus dem Antwortzeit-SLO $\mathrm{sloResponse}(a)$ ein Bedienzeit-SLO $\mathrm{sloWork}(a)$ errechnet werden, sodass intern die Bewertung der lokalen Managementsituation auf Basis von $\mathrm{sloWork}(a)$ und $t_{work}(a)$ durchgeführt werden kann.

Vereinfachend wird im Folgenden weiterhin von Antwortzeit-SLOs gesprochen. Intern führt jede an einem Workflow beteiligte Komponente jedoch die beschriebenen Umrechnungen auf Bedienzeiten und entsprechende SLOs durch.

6.4 Übertragung von Antwortzeit-SLO-Anteilen

6.4.1 Überblick

Analog zur Bewertung der Managementsituation einer einzelnen Komponente kann eine Bewertung für einen gesamten Workflow W oder Teile desselben vorgenommen werden. Falls für W die Bedingung $\Delta slo(\mathsf{W}) \geq 0$ gilt, können SLO-Verletzungen einzelner Workflow-Abschnitte durch Übertragung von Antwortzeit-SLO-Anteilen anderer Abschnitte vollständig kompensiert werden. Eine Übertragung von Antwortzeit-SLO-Anteilen von Diensten mit zu lockerem SLO zu Diensten mit zu straffem SLO erlaubt dem einzelnen Dienst die Minimierung der für die Erbringung der geforderten Dienstgüte aufzuwendenden Ressourcen. Hieraus resultieren i. d. R. Kosteneinsparungen und die Möglichkeit der Erhöhung des Ressourceneinsatzes zum Ausgleich von Lastspitzen.

Die Vorgehensweise bei der Übertragung von Antwortzeit-SLO-Anteilen ergibt sich aus der Struktur des betrachteten Workflows. Daher wird im Folgenden zunächst eine geeignete Notation für die Struktur von in Form von WS-BPEL-Prozessen vorliegenden Workflows beschrieben. Im Anschluss daran wird die Übertragung von Antwortzeit-SLO-Anteilen zunächst für eine einfache Workflow-Sequenz diskutiert, danach wird das Vorgehen für komplexere Workflow-Konstrukte besprochen.

6.4.2 Beschreibung von WS-BPEL-Prozessen

6.4.2.1 Überblick

Zur Definition von Antwortzeit-SLOs für Einzelaktivitäten eines BPEL-Workflows und der Beschreibung von Kooperationsmechanismen zur Verbesserung der Dienstgüte eines Workflows eignet sich die textuelle XML-Darstellung von WS-BPEL-Prozessen nicht. Im Folgenden wird zunächst eine grafische Notation vorgestellt, die dann durch Vereinfachungen und Transformationen im Hinblick auf die Zuordnung von Antwortzeit-SLOs zu einzelnen Aktivitäten optimiert wird – die finale Repräsentierung eines WS-BPEL-Prozesses ist nur noch in Bezug auf Antwortzeitaspekte äquivalent zum eigentlichen Prozess.

6.4.2.2 Überführung in eine Baumdarstellung

Aufgrund des geschachtelten Sprachaufbaus von WS-BPEL-Dokumenten ist es möglich, jede BPEL-Prozess-Beschreibung W in einen Baum $\mathcal{W}$ zu überführen, in dem die im Prozess definierten BPEL-Aktivitäten $A = \{a_1 \dots a_n\}$ als Knoten und Beziehungen zwischen den Aktivitäten als Kanten $K = \{k_1 \dots k_m\}$ modelliert werden:

$$\mathcal{W} = (A, K) \tag{6.2}$$

mit $K \subseteq A \times A$. Eine Kante $(a_i, a_j) \in K$ zwischen den Aktivitäten a_i und a_j wird im Weiteren notiert durch $a_i a_j$.

Die Menge $\Gamma^+(x)$ der *Nachfolger* einer Aktivität x ergibt sich aus $y \in \Gamma^+(x) :\Leftrightarrow xy \in K$. Die Menge $\Gamma^-(x)$ der *Vorgänger* einer Aktivität x ist definiert als $y \in \Gamma^-(x) :\Leftrightarrow yx \in K$. Die Anzahl der Nachfolger einer Aktivität ist damit $d^+(a) :=|\Gamma^+(a)|$, die Anzahl der Vorgänger ergibt sich als $d^-(a) :=|\Gamma^-(a)|$.

Die Menge A der Aktivitäten einer WS-BPEL-Prozessbeschreibung W setzt sich aus Basisaktivitäten (B, vgl. Tabelle 2.1) und strukturierten Aktivitäten (S, vgl. Tabelle 2.2) zusammen: $A := S \cup B$. Hierbei gilt:

$$\text{Für alle } a \in S : d^-(a) \leq 1; d^+(a) \geq 0$$
$$\text{Für alle } a \in B : d^-(a) = 1; d^+(a) = 0$$

Die Wurzel des Baumes wird durch das BPEL `Process`-Element gebildet, als Blätter fungieren die Basic Activities. Die Vorgänger-Nachfolger-Beziehung im Baum drückt aus, dass die Nachfolger-Aktivität strukturell innerhalb der Vorgänger-Aktivität angeordnet ist (Beispiel: Eine `Invoke`-Aktivität a_i ist Teil einer Sequenz a_j, falls gilt: $a_j a_i \in K$).

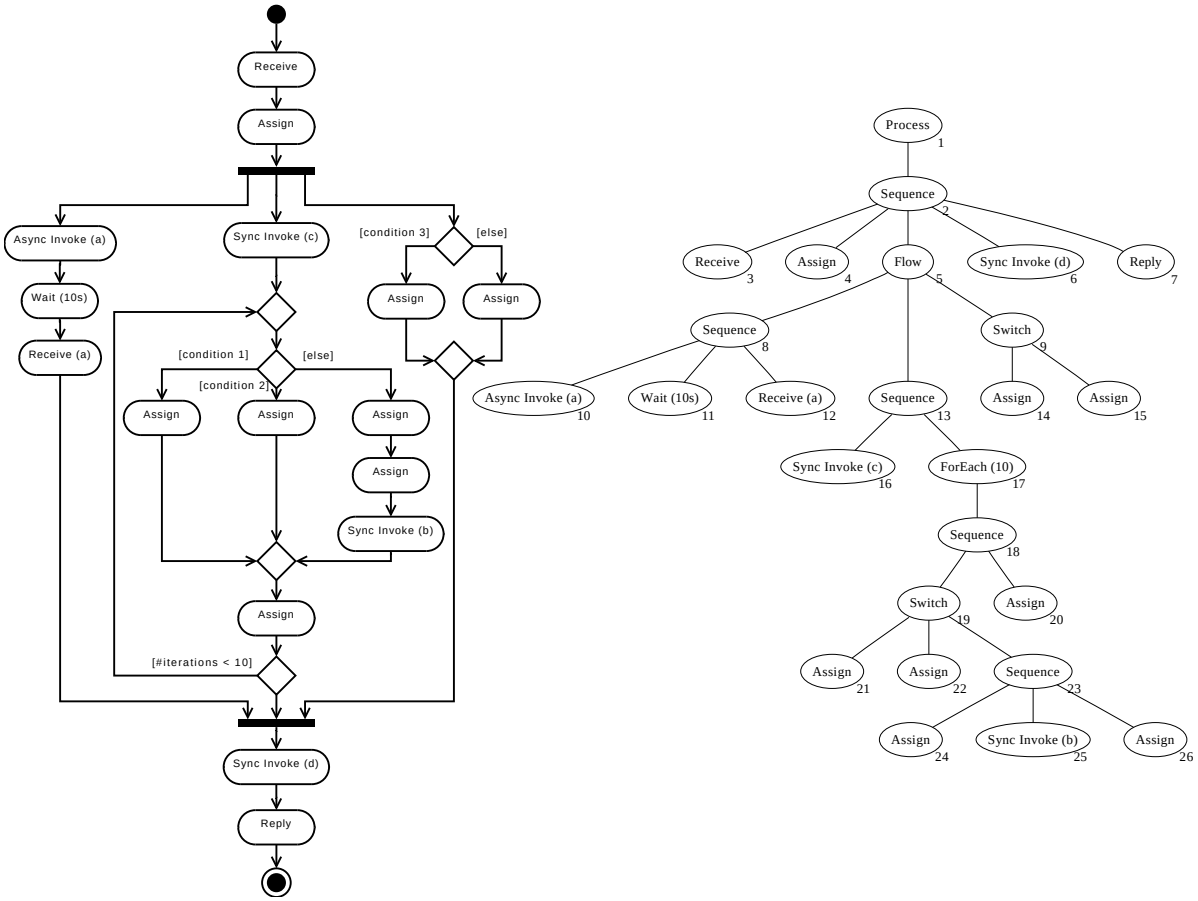

Abbildung 6.2: Visualisierung eines WS-BPEL-Prozesses als UML-Aktivitätsdiagramm; Abbildung des gleichen Prozesses als Baum

Beispiel 6.1: In Abbildung 6.2 ist links ein WS-BPEL-Prozess in Form eines UML-Aktivitätsdiagramms dargestellt. Zur Verbesserung der Lesbarkeit sind die einzelnen Aktivitäten des Prozesses ohne weitere Argumente dargestellt. Auf der rechten Seite der Abbildung ist der gleiche Prozess als Baum dargestellt. Im Gegensatz zur Darstellung als UML-Aktivitätsdiagramm werden in der Baumdarstellung strukturierende WS-BPEL-Sprachelemente (z. B. Sequenzen) explizit als Knoten modelliert.

6.4.2.3 Vereinfachung der Baumdarstellung

Für die Betrachtung maximaler Antwortzeit sind nur diejenigen WS-BPEL-Aktivitäten relevant, die zur Laufzeit eines Prozesses nennenswert zu dessen Antwortzeitverhalten beitragen.

Dies sind – wie in Abschnitt 3.2.2.1 skizziert – im Wesentlichen Aufrufe der am Prozess beteiligten Dienste sowie Aktivitäten, die Wartezeiten im Prozessablauf definieren.

Algorithmus 6.1 Entfernen nicht benötigter Basisaktivitäten

Input: $\mathcal{W}$ `//Beschreibung eines WS-BPEL-Prozesses (vgl. Formel 6.2)`
B `//Menge der Basisaktivitäten des Prozesses` $\mathcal{W}$

1: **for all** $a \in B$ **do**
2: **if** typeOf(a) $\notin \{Invoke, Receive, Wait\}$ **then**
3: remove(a,$\mathcal{W}$);
4: **end if**
5: **end for**

Daher kann die Baumdarstellung des Prozesses vereinfacht werden, indem für die Antwortzeit nicht relevante Aktivitäten eliminiert werden: Hierzu gehören zunächst die `Receive`- und `Reply`-Aktivitäten, die Instanziierung und Beenden der Prozessinstanz modellieren. Im Anschluss können – wie in Algorithmus 6.1 gezeigt – weitere Basisaktivitäten eliminiert werden: Sämtliche Basisaktivitäten ohne externe Kommunikation (mit Ausnahme von `Wait`) sind in Bezug auf ihre Ausführungsdauer zu vernachlässigen. Somit können aus dem den Prozess repräsentierenden Baum, mit Ausnahme der `Invoke`-, `Receive`- und `Wait`-Aktivitäten, alle Basisaktivitäten entfernt werden.

Algorithmus 6.2 Entfernen nicht benötigter strukturierter Aktivitäten

Input: $\mathcal{W}$ `//Beschreibung eines WS-BPEL-Prozesses (vgl. Formel 6.2)`
S `//Menge der strukturierten Aktivitäten des Prozesses` $\mathcal{W}$

1: isOptimized $\Leftarrow$ **false**;
2: **while** (isOptimized = **false**) **do**
3: isOptimized $\Leftarrow$ **true**;
4: **for all** $a \in S$ **do**
5: **if** $d^+(a) = 0$ **then**
6: remove(a,$\mathcal{W}$);
7: isOptimized $\Leftarrow$ **false**;
8: **else if** ((typeOf(a) $\in \{Scope, Sequence\}$) and ($d^+(a)$ = 1) and ($d^-(a) \neq 0$)) **then**
9: $p \Leftarrow \Gamma^-(a)$;
10: **for all** $x \in \Gamma^+(a)$ **do**
11: link(x,p);
12: **end for**
13: remove(a,$\mathcal{W}$);
14: isOptimized $\Leftarrow$ **false**;
15: **end if**
16: **end for**
17: **end while**

In einem weiteren Schritt werden alle überflüssigen strukturierten Aktivitäten entfernt. Im Einzelnen wird hier wie in Algorithmus 6.2 beschrieben vorgegangen:

- Strukturierte Aktivitäten ohne Nachfolger-Elemente werden entfernt (Zeile 5).
- Für `Scope`- und `Sequence`-Aktivitäten mit nur einem Nachfolger werden die Nachfolger direkt mit dem Vorgänger verknüpft (Zeile 11). Im Anschluss werden diese ebenfalls aus dem Prozess entfernt (Zeile 13).

Der Baum wird dabei so lange durchlaufen, bis keine weiteren Modifikationen mehr vorgenommen werden können und somit `isOptimized = true` gilt (Zeile 2).

Beispiel 6.2: Abbildung 6.3 greift den WS-BPEL-Prozess aus Beispiel 6.1 auf. Teil (a) der Abbildung stellt die für die Definition individueller SLAs nicht benötigten Knoten und Kanten gepunktet dar. Die Auswahl dieser Knoten ergibt sich durch Anwendung der Algorithmen 6.1 und 6.2. Teil (b) zeigt den resultierenden reduzierten Baum.

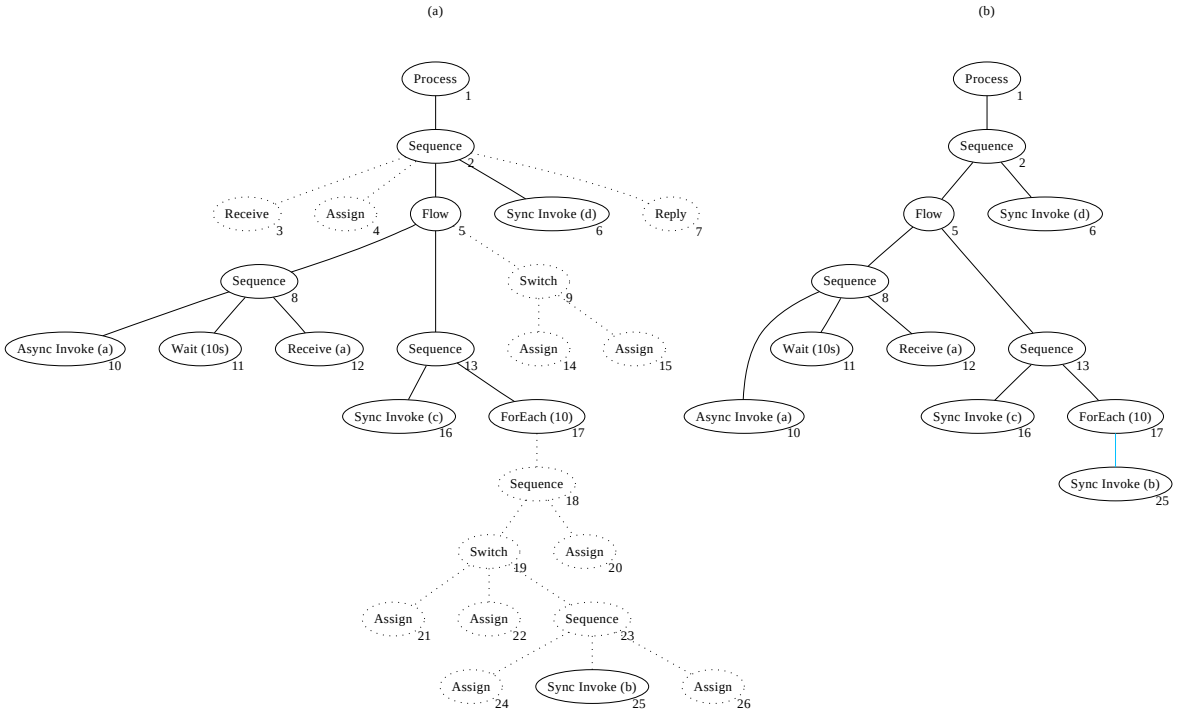

Abbildung 6.3: Vereinfachung der Baumdarstellung des Prozesses aus Abbildung 6.2 mithilfe der Algorithmen 6.1 und 6.2

6.4.2.4 Transformation zur Unterstützung asynchroner Aktivitäten

Beinhaltet ein WS-BPEL-Prozess asynchrone Dienstaufrufe, werden weitere Modifikationen vorgenommen:

- Asynchrone Aufrufe an externe Dienste, bei denen im Verlauf des Prozesses nicht auf eine Antwort gewartet wird, werden aus dem Baum entfernt. Dies bezieht sich im Wesentlichen auf asynchrone `Invoke`-Aktivitäten, zu denen keine entsprechende `Reply`-Aktivität im Prozess existiert.
- Für asynchrone Aktivitäten, bei denen im Prozessverlauf auf eine Antwort gewartet wird, wird eine zusätzliche Flow-Aktivität in den Baum eingefügt, mit deren Hilfe die Abarbeitung des asynchronen Aufrufs und das Warten auf eine Antwort explizit als parallel zu den Aktivitäten modelliert wird, die zwischen Absetzen des Aufrufs und Warten auf die Antwort ausgeführt werden. Mithilfe dieser Kapselung können `Invoke` und `Reply` durch ein synchrones `Invoke` ersetzt werden. Durch das Hinzufügen neuer Aktivitäten wird u. U. das erneute Optimieren des Baumes mithilfe von Algorithmus 6.2 nötig.

Das entsprechende Vorgehen wird in Beispiel 6.3 verdeutlicht.

Beispiel 6.3: In Abbildung 6.4 sind die zur Behandlung asynchroner Dienstaufrufe vorgenommenen Modifikationen der Baumdarstellung des WS-BPEL-Prozesses aus Abbildung 6.3 dargestellt. Teil (a) der Abbildung zeigt den relevanten Ausschnitt des Baumes in seinem ursprünglichen Zustand.

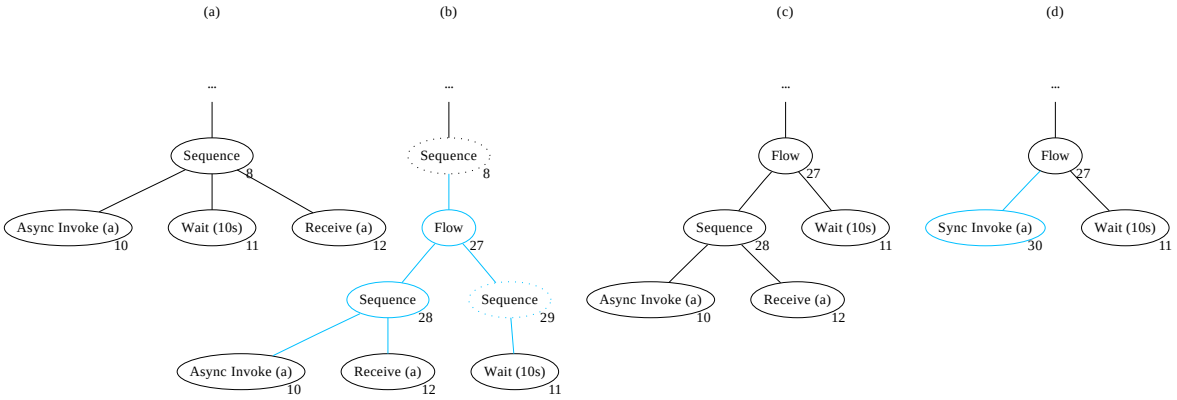

Abbildung 6.4: Baum-Transformation zur Isolierung asynchroner Aktivitäten eines WS-BPEL-Prozesses

Zur Isolation des asynchronen Aufrufs werden (in Teil (b)) eine `Flow`-Aktivität und zwei `Sequence`-Aktivitäten in den Baum eingefügt, die hervorgehoben dargestellt sind. Die `Sequence`-Aktivitäten kapseln den asynchronen Aufruf bzw.

die zwischen `Invoke` und `Reply` ausgeführten Aktionen. Für die SLO-Zuordnung nicht relevante Structured Activities sind wieder gepunktet gekennzeichnet.

In Teil (c) der Abbildung wurden im Rahmen einer erneuten Optimierung diese unnötigen Elemente entfernt (vgl. Algorithmus 6.2). Teil (d) der Abbildung zeigt schließlich den finalen Zustand des Baumes, der isolierte asynchrone Aufruf kann nun bei der SLO-Zuordnung als synchroner Aufruf interpretiert werden. Abbildung 6.5 zeigt eine vollständige Baumdarstellung des auf diese Weise transformierten WS-BPEL-Prozesses.

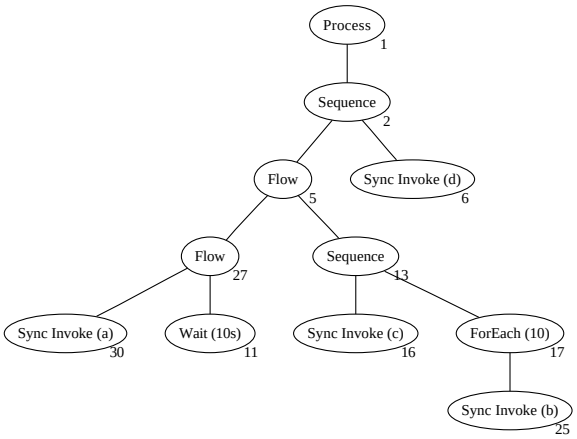

Abbildung 6.5: Vollständig transformierter und optimierter WS-BPEL-Prozess

6.4.2.5 Bestimmung der möglichen Pfade eines Prozesses

Antwortzeit-SLO-Anteile können lediglich innerhalb eines Workflow-Ausführungspfades verschoben werden, da das für den Gesamt-Workflow vereinbarte SLO unverändert bleiben soll. Daher werden im Folgenden zunächst die möglichen Ausführungspfade eines Workflows identifiziert.

Definition 6.1: AUSFÜHRUNGSPFAD EINES WORKFLOWS
Ein Ausführungspfad P eines Workflows $\mathcal{W}$ ist definiert als eine Menge von Aktivitäten $W_P \subseteq A$, die durch eine Menge von Kanten $L_P \subseteq K$ miteinander verbunden sind und so einen Pfad des Workflows vom definierten Startpunkt bis zu einem beliebigen Endpunkt bilden.

Zur Bestimmung aller möglichen Ausführungspfade eines Prozesses werden aus dem final reduzierten Baum alle Teilbäume erzeugt, die einen vollständigen Pfad durch den Baum beschreiben. Verzweigungen, die in unterschiedlichen Ausführungspfaden resultieren, werden

dabei entweder durch Parallelausführungs-Aktivitäten (`flow`) oder durch Fallunterscheidungen (`switch`) ausgedrückt. Die Gleichsetzung von Parallelausführungen und Fallunterscheidungen zur Aufspaltung unterschiedlicher Ausführungspfade gilt nur für die Betrachtung der Antwortzeit eines Prozesses: Die durch das Workflow-SLO geforderte Antwortzeit ist über alle Workflow-Pfade zu gewährleisten – dies gilt sowohl für „echt unterschiedliche“ Pfade, die durch eine Fallunterscheidung gekennzeichnet werden, als auch für die einzelnen Ausführungsstränge eines parallelen Abschnitts.

Zur Vereinfachung der resultierenden, jeweils einen Ausführungspfad repräsentierenden Bäume wird wiederum Algorithmus 6.2 angewandt.

Beispiel 6.4: In Beispiel 6.3 wurde die Vorgehensweise bei der Transformation der Baumdarstellung eines WS-BPEL-Prozesses zur Definition individueller Antwortzeit-SLOs für die am Prozess beteiligten Dienste beschrieben. Ergebnis dieser Transformation ist der in Abbildung 6.5 dargestellte Baum. Dieser Baum enthält zwei `Flow`-Aktivitäten und damit mehrere mögliche Pfade. Nach Erzeugung eines Teilbaumes für jeden Pfad des Prozesses und der Vereinfachung der resultierenden Bäume mithilfe von Algorithmus 6.2 erhält man die in Abbildung 6.6 dargestellten Bäume.

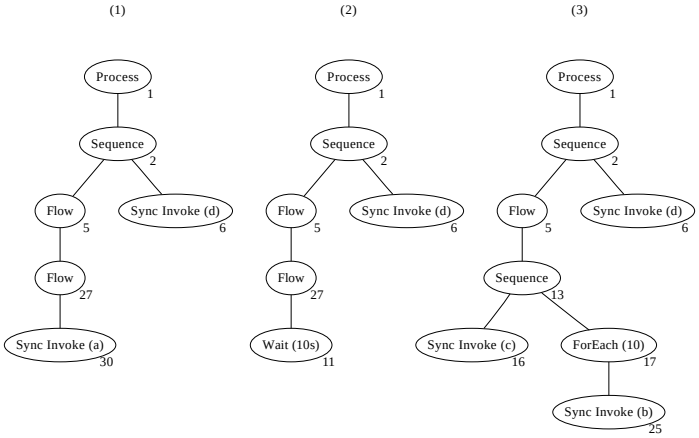

Abbildung 6.6: Die möglichen Ausführungspfade des Prozesses aus Abbildung 6.5

Im final transformierten und reduzierten Baum $\mathrm{tree}(P)$ jedes Ausführungspfades P bilden Aufrufe von Diensten eine Untermenge der Basisaktivitäten:

$$I_P := \{x \in W_P | x.type = \texttt{Invoke}\} \tag{6.3}$$

Die Menge I_P bildet damit die Gesamtheit der Aktivitäten eines Prozesspfades, für die zur Laufzeit Antwortzeit-SLOs definiert werden. Hierbei werden Dienste danach unterschieden,

ob das vereinbarte Antwortzeit-SLO zur Laufzeit als statisch oder veränderbar angesehen wird. Statische SLOs bestehen z. B. in Situationen, in denen für die Nutzung eines externen Dienstes feste Verträge ausgehandelt wurden oder kein QoS-Manager zur dynamischen SLO-Aushandlung verfügbar ist.

Für alle `Invoke`-Aktivitäten liefert die Funktion $\mathrm{slo}(x)$ das jeweils vereinbarte Antwortzeit-SLO zurück:

$$\text{Für alle } i \in I_P : \mathrm{slo}(i) := i.slo \tag{6.4}$$

Invoke-Aktivitäten mit unveränderlichem Antwortzeit-SLO sind entsprechend über ein Attribut $i.fix = \texttt{true}$ gekennzeichnet, für Aktivitäten mit verhandelbarem Antwortzeit-SLO gilt $i.fix = \texttt{false}$.

Die Menge Z_P der `Wait`-Aktivitäten eines Ausführungspfades P bildet ebenfalls eine Untermenge der Basisaktivitäten im final transformierten und reduzierten Baum $\mathrm{tree}(P)$:

$$Z_P := \{x \in W_P | x.type = \texttt{Wait}\} \tag{6.5}$$

Für alle `Wait`-Aktivitäten liefert die Funktion $\mathrm{slo}(x)$ die durch die jeweilige Aktivität verursachte Wartezeit zurück:

$$\text{Für alle } z \in Z_P : \mathrm{slo}(z) := z.duration \tag{6.6}$$

Die Menge der Basisaktivitäten B_P eines Pfades P des transformierten und reduzierten Baum ist definiert als die Vereinigungsmenge der `Wait`- und `Invoke`-Aktivitäten des Pfades:

$$B_P := Z_P \cup I_P \tag{6.7}$$

6.4.3 Berechnung von Antwortzeit-SLOs für strukturierte Aktivitäten

Die Berechnung von Antwortzeit-SLOs für strukturierte Aktivitäten eines Workflows erfolgt auf Basis der Aggregation von Antwortzeit-SLOs, die für Aktivitäten eines Workflow-Teilpfades (vgl. Definition 6.2) definiert wurden.

Definition 6.2: TEILPFAD EINES WORKFLOWS
Als Teilpfad T eines Workflows $\mathcal{W}$ wird eine über eine Menge von Kanten $L_T \in K$ verbundene Untermenge der einen Workflow-Ausführungspfad P (vgl. Definition 6.1) kennzeichnenden Aktivitäten $W_T \subseteq A$ bezeichnet.

Im Weiteren wird der Begriff Teilpfad ausschließlich für solche Pfade verwendet, die in der grafischen Notation einen vollständigen Teilbaum des Workflows definieren.

Die Berechnung von Antwortzeit-SLOs für Basisaktivitäten eines Workflows erfolgt wie durch Gleichung 3.3 beschrieben.

Antwortzeit-SLOs für strukturierte Aktivitäten werden wie folgt bestimmt: Für eine Sequenz x ergibt sich das Gesamt-SLO aus der Summe der Antwortzeit-SLOs der Menge der Kindelemente $a_i \in \Gamma^+(x)$ der Sequenz:

$$\mathrm{slo}(x) = \sum_{i=0}^{n} \mathrm{slo}(a_i) \tag{6.8}$$

Für eine Schleife x ergibt sich das Antwortzeit-SLO als Multiplikation des Antwortzeit-SLOs des Schleifenkörpers ($a_i \in \Gamma^+(x)$) mit der maximalen Anzahl n_x der Schleifendurchläufe:

$$\mathrm{slo}(x) = n_x \times \sum_{i=0}^{m} \mathrm{slo}(a_i) \tag{6.9}$$

Das Antwortzeit-SLO von Fallunterscheidungen und Parallelausführungen wird als Maximum der Antwortzeit-SLOs der untergeordneten Teilpfade $a_i \in \Gamma^+(x)$ definiert:

$$\mathrm{slo}(x) = \max_{a_i \in \Gamma^+(x)} \{\mathrm{slo}(a_i)\} \tag{6.10}$$

Mithilfe der Gleichungen 6.8 bis 6.10 können rekursiv Antwortzeit-SLOs für beliebige Teilpfade eines Workflows ermittelt werden. Das auf diese Weise ermittelte Antwortzeit-SLO für den Wurzelknoten entspricht dem Gesamt-Antwortzeit-SLO des Workflows.

6.4.4 Sicherstellen der Einhaltung des Antwortzeit-SLOs eines Workflows

Zur Überprüfung der Einhaltung des Gesamt-Antwortzeit-SLOs eines Workflows muss die Antwortzeit des Workflows zur Laufzeit überwacht werden. Dies ist die Aufgabe des in Abschnitt 5.4 vorgestellten QoS-Managers, der einem Workflow direkt zugeordnet wird. Dabei ist es unerheblich, ob jeder einzelne an der Ausführung beteiligte Dienst in der Lage ist, das ihm gestellte Antwortzeit-SLO zu erfüllen: Notwendige Bedingung zur Erfüllung des Gesamt-Antwortzeit-SLOs ist, dass die Summe der Antwortzeiten aller Workflow-Pfade maximal der im Antwortzeit-SLO definierten Zeitspanne entspricht. Aus Sicht eines einzelnen Dienstes bzw. des ihm zugeordneten QoS-Managers gibt es keine Möglichkeit für eine derartige Überprüfung des Gesamt-Antwortzeit-SLOs eines Workflows. Diesem sind sowohl die Struktur des aufrufenden Workflows als auch dessen Gesamt-Antwortzeit-SLO unbekannt; er kennt lediglich sein eigenes Antwortzeit-SLO und sein aktuelles Dienstgüteverhalten Δslo.

Zur Erfüllung des Gesamt-Antwortzeit-SLOs hinreichend ist, wenn für alle Einzelaktivitäten $\Delta slo(a_i) \geq 0$ gilt. Diese Bedingung ist für den QoS-Manager eines Dienstes auf Basis lokal vorliegender Informationen verifizierbar. Macht man das Einhalten dieser Bedingung in einer dezentralen SOA-Umgebung zum Ziel für jeden einzelnen QoS-Manager, so kann auf diese Weise die Einhaltung des Gesamt-Antwortzeit-SLOs eines Workflows verfolgt werden.

Kann ein QoS-Manager das lokal gesteckte Ziel nicht erreichen, versucht er, dieses Ziel durch Übertragung von Antwortzeit-SLO-Anteilen anderer am Workflow beteiligter Komponenten auf sich selbst zu *lockern*. Im Gegenzug kann ein QoS-Manager, der sein eigenes Ziel übertrifft, dieses durch Übertragung an andere QoS-Manager *straffen*. Essenziell bei der Übertragung von Antwortzeit-SLO-Anteilen zwischen QoS-Managern ist, dass dadurch das Gesamt-Antwortzeit-SLO des Workflows (vgl. Abschnitt 6.4.3) nicht verändert wird. Um dies sicherzustellen, dürfen Antwortzeit-SLO-Anteile nicht zwischen beliebigen QoS-Managern eines Workflows transferiert werden, Mechanismen zur Übertragung von Antwortzeit-SLO-Anteilen müssen vielmehr die Struktur eines Workflows berücksichtigen.

Im Folgenden wird ein Verfahren zur Übertragung von Antwortzeit-SLO-Anteilen zwischen QoS-Managern vorgestellt, das den oben beschriebenen Anforderungen genügt. Das Verfahren definiert zur Unterstützung der Kriterien Dezentralität und Skalierbarkeit folgende Ziele:

- *Kein globales Wissen*: Zur Antwortzeit-SLO-Übertragung soll kein Dienst globales Wissen über Workflow-Strukturen benötigen.
- *Lokalität*: Die Übertragung benötigter Antwortzeit-SLO-Anteile soll möglichst aus der durch den jeweiligen Workflow-Pfad gegebenen lokalen Umgebung eines QoS-Managers erfolgen.

In den folgenden Abschnitten wird zunächst auf das Übertragen von Antwortzeit-SLO-Anteilen innerhalb einer Workflow-Sequenz eingegangen, im Anschluss wird das Verfahren dann

auf andere Workflow-Strukturen erweitert. In einer SOA werden Dienste regelmäßig parallel durch mehrere Workflows genutzt. In Abschnitt 5.4.3.3 wurde gezeigt auf welche Weise ein QoS-Manager parallel Dienstgüteanforderungen unterschiedlicher Workflows erfüllen kann. Das im Folgenden beschriebene Verfahren zur Übertragung von Antwortzeit-SLO-Anteilen konzentriert sich auf die Betrachtung eines einzelnen Workflows, kann aber in Kombination mit den in Abschnitt 5.4.3.3 beschriebenen Techniken auch parallel für mehrere Workflows angewandt werden.

6.4.5 Übertragung von Antwortzeit-SLO-Anteilen in sequenziellen Strukturen

Innerhalb einer Workflow-Sequenz x werden alle Aktivitäten $a_i \in \Gamma^+(x)$ betrachtet. Ziel ist, durch Übertragen von Antwortzeit-SLO-Anteilen zu erreichen, dass für alle $a_i \in \Gamma^+(x)$ gilt: $\Delta slo(a_i) \geq 0$.

Im Folgenden werden zwei Arten der Übertragung von Antwortzeit-SLO-Anteilen unterschieden: Gilt für die Sequenz $\sum_{i=0}^{n} \Delta slo(a_i) \geq 0$, so kann durch Übertragen von Antwortzeit-SLO-Anteilen innerhalb der Sequenz ein Ausgleich erzielt werden. Dieser Fall wird in Abschnitt 6.4.5.1 behandelt. Gilt $\sum_{i=0}^{n} \Delta slo(a_i) < 0$, so ist ein Antwortzeit-SLO-Ausgleich durch Übertragen von SLO-Anteilen innerhalb der lokalen Sequenz x nicht erreichbar, der Versuch eines Ausgleichs muss auf übergeordnete Workflow-Strukturen ausgedehnt werden. Das Vorgehen wird in Abschnitt 6.4.5.2 beschrieben.

6.4.5.1 Lokales Übertragen von Antwortzeit-SLO-Anteilen

Durch lokales Übertragen von Antwortzeit-SLO-Anteilen kann eine Lösung gefunden werden, falls in der Sequenz Aktivitäten existieren, die durch Straffung der ihnen zugeordneten Antwortzeit-SLOs frei werdende Antwortzeit-SLO-Anteile an andere Aktivitäten innerhalb derselben Sequenz abtreten können, sodass im Anschluss alle Aktivitäten ihre jeweiligen Antwortzeit-SLOs einhalten können.

> **Beispiel 6.5:** Abbildung 6.7 zeigt beispielhaft das lokale Übertragen von Antwortzeit-SLO-Anteilen innerhalb der durch den Workflow-Pfad (1) aus Abbildung 6.6 definierten Sequenz x. In der Abbildung ist der numerische Wert von Δslo für die einzelnen Aktivitäten vor Durchführung der Übertragung angegeben: Eine der dargestellten `Invoke`-Aktivitäten verfügt über $\Delta slo = 3$, die andere `Invoke`-Aktivität verfügt über $\Delta slo = -2$. Im Beispiel kann eine lokale Lösung erreicht werden, da sich für die gesamte Sequenz $\Delta slo(x) = 1$ ergibt. Die Lösung besteht hier im Übertragen von 2 Antwortzeit-SLO-Anteilen von `Invoke(a)` nach `Invoke(d)`, was in der Grafik durch einen gepunkteten Pfeil visualisiert wird.

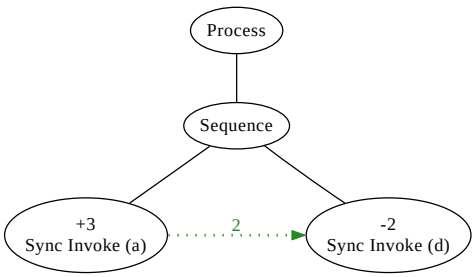

Abbildung 6.7: Lokales Verschieben von Antwortzeit-SLO-Anteilen zwischen Elementen einer Sequenz

6.4.5.2 Nicht-lokales Übertragen von Antwortzeit-SLO-Anteilen

Ist ein Übertragen von Antwortzeit-SLO-Anteilen innerhalb der lokalen Sequenz x nicht möglich oder nicht ausreichend, erfolgt der Ausgleich rekursiv bis zur Wurzel des Workflow-Pfades. Hierzu propagiert die die Sequenz repräsentierende strukturierte Aktivität den Wert von $\Delta slo(x)$ in die übergeordnete Workflow-Struktur. Diese Propagation muss rekursiv durchgeführt werden, bis ein Ausgleich möglich ist. Kann ein Ausgleich erreicht werden, wird der benötigte Antwortzeit-SLO-Anteil an die entsprechenden Komponenten der Sequenz weitergereicht.

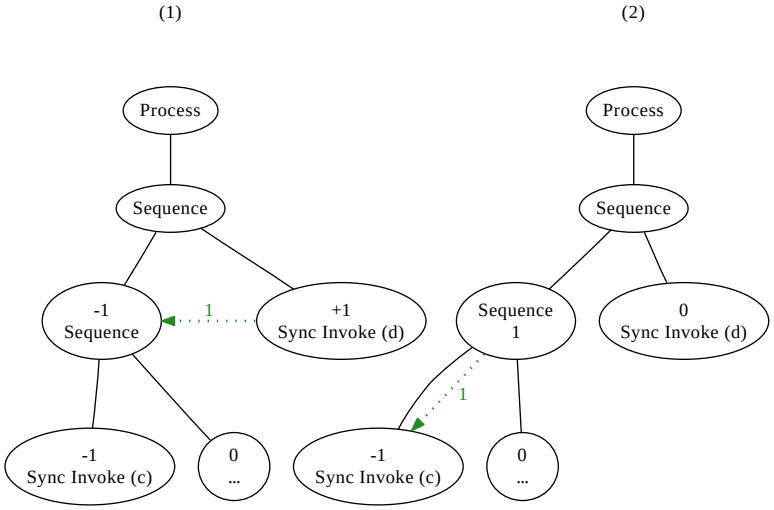

Abbildung 6.8: Verschieben von Antwortzeit-SLO-Anteilen über die Grenzen einer Sequenz hinweg

Beispiel 6.6: Abbildung 6.8 zeigt beispielhaft das Übertragen von Antwortzeit-SLO-Anteilen über die Grenzen einer Sequenz x hinweg als mehrschrittigen Prozess. Im ersten Schritt (vgl. Teil (1) der Abbildung) beträgt $\Delta slo(x) = -1$ –

ausgelöst durch das $\Delta slo = -1$ der abgebildeten Aktivität `Invoke(c)`. Da keine lokale Lösung möglich ist, wird $\Delta slo(x)$ in die nächste übergeordnete Struktur propagiert. Dies ist in der Abbildung durch den Wert „-1“ über der `Sequence`-Aktivität dargestellt. Hier kann ein Ausgleich erfolgen, da für die Aktivität `Invoke(d)` Δslo den Wert 1 ergibt. Nach Übertragung dieses Wertes an die Sequenz kann diese ihren untergeordneten Aktivitäten den erhaltenen Wert 1 offerieren (im Bild als Schritt (2) dargestellt). Ab hier verläuft die Übertragung von Antwortzeit-SLO-Anteilen wie zuvor für den lokalen Fall beschrieben.

6.4.6 Übertragung von Antwortzeit-SLO-Anteilen über Schleifengrenzen

Ist die übergeordnete Workflow-Aktivität eine Schleife x, dann müssen Antwortzeit-SLO-Anteile bei der Übertragung über eine Schleifengrenze hinweg mit der maximalen Anzahl n_x der Schleifendurchläufe multipliziert bzw. durch diese dividiert werden.

Überträgt eine Aktivität a innerhalb der Schleife überschüssige Antwortzeit-SLO-Anteile nach außen, wird die maximal zu transferierende Menge mit $n_x \times \Delta slo(a)$ berechnet. Benötigt eine Aktivität a innerhalb der Schleife zusätzliche Antwortzeit-SLO-Anteile von einer Aktivität b außerhalb der Schleife, so erhält sie maximal $\frac{\Delta slo(b)}{n_x}$ Anteile.

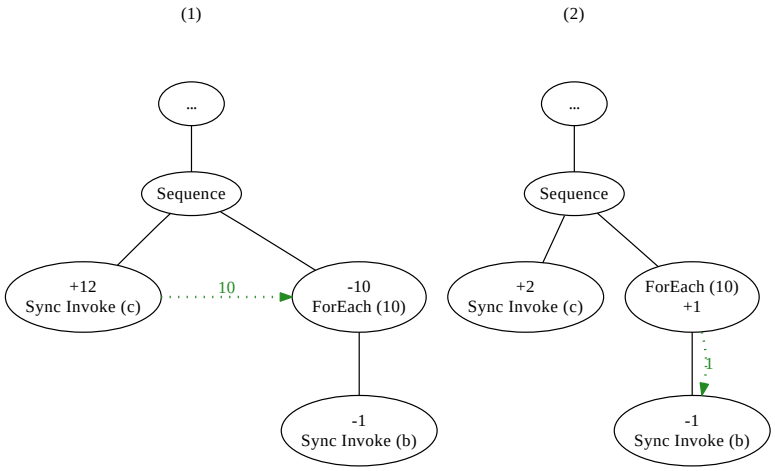

Abbildung 6.9: Verschieben von Antwortzeit-SLO-Anteilen über die Grenzen einer Schleife hinweg

Analog zur Vorgehensweise beim nicht-lokalen Übertragen von Antwortzeit-SLO-Anteilen (vgl. Abschnitt 6.4.5.2) wird beim Transferieren von Antwortzeit-SLO-Anteilen über Schleifengrenzen hinweg in mehreren Schritten vorgegangen.

Beispiel 6.7: Abbildung 6.9 zeigt das Übertragen von Antwortzeit-SLO-Anteilen über die Grenzen einer Schleife x hinweg. Da Δslo der in der Schleife enthaltenen `Invoke`-Aktivität -1 beträgt und die Schleife für $n_x = 10$ Durchläufe ausgelegt ist, ergibt $\Delta slo(x) = n_x \times (-1) = 10 \times (-1) = -10$. Die Schleife propagiert nun diesen Δslo-Wert in die übergeordnete Sequenz. In dieser Sequenz existiert eine `Invoke`-Aktivität mit einem Δslo von $+12$. Diese überträgt der Schleifen-Aktivität die benötigten 10 Antwortzeit-SLO-Anteile (dargestellt durch den gepunkteten Pfeil). Die Schleifen-Aktivität dividiert diese durch die Anzahl der Schleifendurchläufe und leitet sie an die in der Schleife enthaltene `Invoke`-Aktivität weiter (vgl. Teil (2) der Abbildung). Die Übertragung von Antwortzeit-SLO-Anteilen aus einer Schleife heraus erfolgt auf umgekehrtem Weg.

6.4.7 Übertragen von Antwortzeit-SLO-Anteilen über Grenzen von Parallelausführungen und Fallunterscheidungen

Das Transferieren von Antwortzeit-SLO-Anteilen über Grenzen von Parallelausführungen und Fallunterscheidungen hinweg funktioniert prinzipiell genauso wie das Übertragen von Anteilen über Sequenzgrenzen. Allerdings muss hier sichergestellt werden, dass Veränderungen an einem Antwortzeit-SLO die den benachbarten Pfaden zugewiesenen Antwortzeit-SLOs nicht beeinträchtigen.

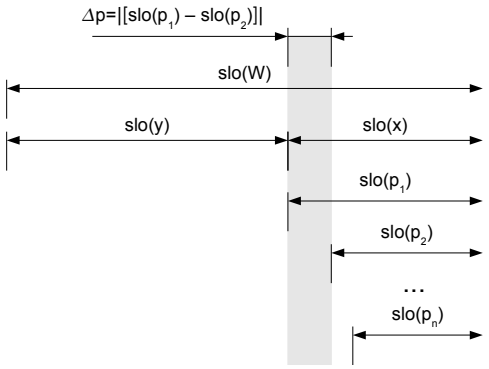

Abbildung 6.10: Zusammenhang zwischen Antwortzeit-SLOs einzelner Ausführungspfade

Abbildung 6.10 zeigt den Zusammenhang zwischen Antwortzeit-SLOs, die für alternative Teilpfade p_1 bis p_n einer Parallelausführung oder Fallunterscheidung x definiert sind. In der Abbildung bezeichnet $\mathrm{slo}(\mathcal{W})$ das Gesamt-Antwortzeit-SLO des Workflows, $\mathrm{slo}(p_1)$ und $\mathrm{slo}(p_2)$ die Antwortzeit-SLOs, die für die längsten Teilpfade p_1 und p_2 der Parallelausführung x vereinbart wurden. Als $\mathrm{slo}(y)$ wird der Anteil des Gesamt-Antwortzeit-SLOs bezeichnet, der außerhalb der Parallelausführung oder Fallunterscheidung vergeben wurde. Die Differenz

zwischen dem maximalen SLO $\mathrm{slo}(p_1)$ und dem zweitgrößten SLO $\mathrm{slo}(p_2)$ ist in der Abbildung als Δp bezeichnet.

Allgemein wird der Wert Δp für eine Parallelausführung oder Fallunterscheidung x wie folgt bestimmt:

Zunächst wird mit X_{max} die Menge derjenigen Teilpfade von x bestimmt, deren Antwortzeit-SLO maximal ist (also gleich dem Antwortzeit-SLO von x, vgl. Gleichung 6.10).

$$X_{max} := \{p \in \Gamma^+(x) | (\mathrm{slo}(p) = \mathrm{slo}(x))\} \tag{6.11}$$

Es wird aus der Menge X_{max} ein beliebiges Element p' ausgewählt.

$$\text{sei } p' \in X_{max} \tag{6.12}$$

Die Menge X_{rest} wird definiert als die Menge aller Teilpfade von x, deren SLO nicht maximal ist, die also nicht in der Menge X_{max} enthalten sind.

$$X_{rest} := \Gamma^+(x) \backslash X_{max} \tag{6.13}$$

Aus der Menge der Elemente in X_{rest} wird nun ein beliebiges Element p'' mit maximalem SLO ausgewählt.

$$\text{sei } p'' \in \{p \in \Gamma^+(X_{rest}) | (\mathrm{slo}(p) = \mathrm{slo}(X_{rest}))\} \tag{6.14}$$

p' und p'' bezeichnen damit zwei Teilpfade von x, denen das maximal in einem Teilpfad von x vorkommende SLO sowie das nächstkleinere SLO zugeordnet sind. Aus der Differenz der SLOs dieser Teilpfade kann nun der Wert Δp berechnet werden.

$$\Delta p := \mathrm{slo}(p') - \mathrm{slo}(p'') \tag{6.15}$$

Allgemein gelten für Parallelausführungen und Fallunterscheidungen folgende Regeln bei der Übertragung von Antwortzeit-SLOs:

R1 Beim Transfer von Antwortzeit-SLO-Anteilen von p' an übergeordnete Strukturen können maximal Δp Anteile übertragen werden, ohne die Antwortzeit-SLOs anderer Pfade zu verletzen.

R2 Beim Transfer von Antwortzeit-SLO-Anteilen aus übergeordneten Strukturen nach p' können maximal $\mathrm{slo}(y)$ Anteile übertragen werden.

R3 p'' kann sein Antwortzeit-SLO um maximal Δp Anteile erweitern, ohne dass Antwortzeit-SLOs anderer Teile des Workflows verletzt werden. Die SLO-Erweiterung bedarf keiner Übertragung von Antwortzeit-SLO-Anteilen aus übergeordneten Strukturen.

R4 Gilt $|X_{max}| > 1$, so können Antwortzeit-SLO-Anteile nur dann an übergeordnete Strukturen übertragen werden, wenn die zu übertragende Menge an Antwortzeit-SLO-Anteilen allen Elementen der Menge X_{max} abgezogen werden kann. Die maximal übertragbare Menge von Antwortzeit-SLO-Anteilen beträgt dabei Δp (vgl. **R1**).

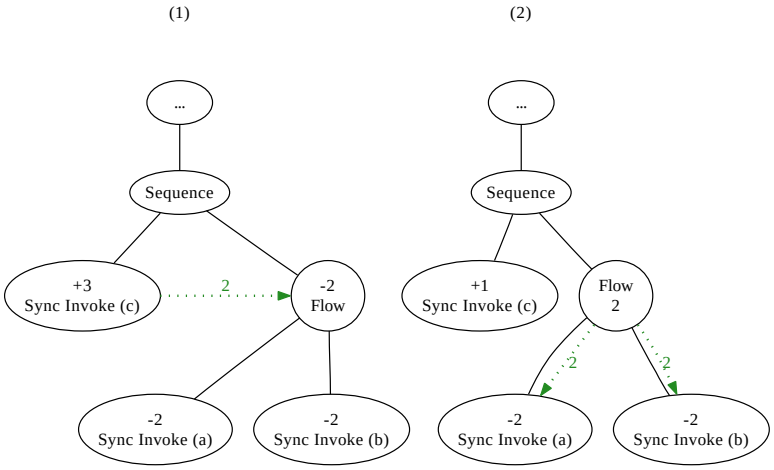

Abbildung 6.11: Verschieben von Antwortzeit-SLO-Anteilen über die Grenzen einer Parallelausführung

Beispiel 6.8: In Abbildung 6.11 ist eine Situation dargestellt, in der eine Parallelausführung zwei Teilpfade mit jeweils einer `Invoke`-Aktivität beinhaltet. Für die Parallelausführung gilt $\Delta p = 0$. Beide `Invoke`-Aktivitäten weisen ein Δslo von -2 auf. Dieses Δslo wird in Teil (1) der Abbildung durch die `Flow`-Aktivität in die übergeordnete Sequenz propagiert. Dort kann das Δslo durch eine Invoke-Aktivität, die über ein Δslo von $+3$ verfügt, mittels Transfer von 2 Antwortzeit-SLO-Anteilen ausgeglichen werden. In Teil (2) der Abbildung werden die benötigten 2 Antwortzeit-SLO-Anteile dann von der `Flow`-Aktivität an die einzelnen Pfade übermittelt.

Durch Kombination der vorgestellten Vorgehensweisen zur Übertragung von Antwortzeit-SLO-Anteilen für die unterschiedlichen in einem Workflow enthaltenen Struktur-Elemente (Sequenz, Schleife, Parallelausführungen, Fallunterscheidungen) kann – soweit möglich – ein SLO-Ausgleich auch in komplex strukturierten Workflows erreicht werden.

6.5 Betrachtete Koordinationsverfahren

6.5.1 Überblick

Um die in Abschnitt 6.4.5 beschriebenen Optimierungen ohne zentrale Kontrolle der einzelnen Architekturkomponenten durchführen zu können, werden geeignete Verfahren zur Strukturierung der Kommunikation der beteiligten QoS-Manager benötigt. Die Charakteristika des verwendeten Koordinationsverfahrens bestimmen dabei maßgeblich den Erfolg der Optimierung. Beispiele für Koordinationsverfahren zur Ressourcenverteilung oder -übertragung sind Auktionen und Basar-Protokolle [WWWM01] sowie Protokolle zur sozialen Interaktion [GWBS03]. Aus der Perspektive der Spieltheorie werden in [Jun08] Auktionen als Koordinationsverfahren im SLM-Kontext diskutiert. Im Folgenden werden unterschiedliche Auktionsmechanismen vorgestellt und auf ihre Eignung als Koordinationsmechanismus zur Übertragung von Antwortzeit-SLO-Anteilen innerhalb von Workflow-Pfaden untersucht.

6.5.2 Auktionen

6.5.2.1 Überblick

Teilnehmer einer Auktion sind prinzipiell ein Auktionator (der entweder Agent eines Verkäufers oder der Verkäufer selbst sein kann) und ein oder mehrere Bieter, wobei einige Auktionsverfahren nur mit mehr als einem Bieter korrekt funktionieren. Jeder Teilnehmer einer Auktion verfolgt ein eigenes Ziel; der Auktionator versucht, den Preis für die Ware zu maximieren, Bieter hingegen versuchen, den Preis so gering wie möglich zu halten. Am Ende einer erfolgreichen Auktion wechselt die versteigerte Ware in den Besitz des Bieters, der als Gewinner aus der Auktion hervorgegangen ist [Kri02] gegen Zahlung eines vereinbarten Preises.

In der Praxis findet man unterschiedliche Varianten von Auktionsverfahren, vier Grundtypen werden im Folgenden vorgestellt:

- Bei der klassischen *Englischen Auktion* werden ausschließlich *öffentliche Gebote* abgegeben; der Preis steigt mit jedem Gebot. Optional benennt der Auktionator einen Mindestpreis. Gehen innerhalb einer definierten Zeitspanne keine weiteren Gebote mehr ein, erhält der letzte Bieter den Zuschlag für die Ware.
- Die *Holländische Auktion* startet bei einem deutlich überhöhten Preis, der schrittweise in regelmäßigen Abständen weiter gesenkt wird. Hier erhält das erste Gebot den Zuschlag zum aktuellen Preis.
- Bei einer *Verdeckten Ausschreibung* darf jeder Bieter ein einziges, *verdecktes Gebot* abgeben. Das für den Auktionator beste Angebot erhält den Zuschlag (beim Verkauf von Ware das Höchste, beim Kauf von Ware das Niedrigste).

- Eine *Vickrey-Auktion* [Vic61] verläuft fast analog zu einer verdeckten Ausschreibung, allerdings erhält der Bieter mit dem höchsten Gebot den Zuschlag zum Preis des zweithöchsten Gebots.

In Weiteren werden diese Verfahren in Bezug auf die folgenden Kriterien verglichen:

- *Gebotsabgabe*
  Man unterscheidet Auktionen, die mit öffentlichen, für alle Teilnehmer sichtbaren Geboten arbeiten, von Auktionen, bei denen mit verdeckten Geboten gearbeitet wird.
- *Preisermittlung*
  Auktionsverfahren unterscheiden sich wesentlich durch die Art der Ermittlung des Gewinners und damit in der Vorgehensweise zur Ermittlung des Verkaufspreises.
- *Terminierungskriterien*
  Terminierungskriterien legen fest, wann eine Auktion endet. Abhängig vom Auktionstyp kann dies zu einem festen Zeitpunkt, nach dem Verstreichen einer festgelegten Zeitspanne ohne Auftreten eines Ereignisses oder beim Auftreten eines bestimmten Ereignisses sein. Das Betrachten von Terminierungskriterien bildet die Grundlage für das Abschätzen der Dauer einer Auktion.
- *Kommunikationsaufwand*
  Auktionsverfahren unterscheiden sich in der Anzahl der Nachrichten, die zur Durchführung zwischen den Auktionsteilnehmern ausgetauscht werden müssen. Abhängig vom unterlagerten Kommunikationsprotokoll werden Nachrichten an alle Teilnehmer (z. B. Ankündigung einer Auktion, öffentliche Gebote, ...) auf Broadcast/Multicast-Mechanismen abgebildet oder mithilfe einzeln adressierter Nachrichten realisiert. Die Betrachtung des Kommunikationsaufwands eines Auktionsverfahrens dient der Bestimmung der Last für die unterlagerte Kommunikationsinfrastruktur.

Grundannahme für die Durchführung einer Auktion ist, dass jeder Auktionsteilnehmer lediglich über eingeschränkte Informationen – seine lokale Sicht auf das Gesamtgeschehen – verfügt und dass in einer solchen Situation ein fairer Preis für das zu veräußernde Gut festgelegt werden soll.

6.5.2.2 Gebotsabgabe und Preisermittlung

Bei der Gebotsabgabe verhalten sich die einzelnen Auktionsteilnehmer strategisch, d. h. eigene Aktivitäten erfolgen basierend auf der Abschätzung des Verhaltens der anderen Teilnehmer. Im Gegensatz zu verdeckten Auktionen kann bei offenen Auktionen der einzelne Teilnehmer seine Biet-Strategie direkt auf das sichtbare Verhalten der anderen Teilnehmer abstimmen.

Im Folgenden wird die Auktion eines unteilbaren Gutes mit $N = \{1, \ldots, m\}$ Teilnehmern betrachtet. Jeder Teilnehmer i trifft privat seine eigene Wertabschätzung w_i des zu versteigernden Gutes. Ziel jedes Auktionsteilnehmers ist die Maximierung des eigenen Ertrags e_i:

Im Fall eines Auktionsgewinns ergibt sich dieser aus der Differenz der Wertabschätzung w_i und des zu zahlenden Preises p: $e_i = w_i - p$. Wird die Auktion durch einen anderen Teilnehmer gewonnen, gilt $e_i = 0$. Um einen negativen Ertrag zu vermeiden, wird kein rational agierender Teilnehmer bereit sein, einen Preis $p > w_i$ zu zahlen.

Bei der Holländischen Auktion und der verdeckten Ausschreibung kann ein Bieter i die gleiche Strategie verfolgen (vgl. [Mol97]): Zunächst schätzt er mit den ihm zur Verfügung stehenden Informationen die Wertabschätzungen der anderen Auktionsteilnehmer. Unter der Annahme, dass die eigene Wertabschätzung w_i am größten ist, bestimmt der Bieter nun w' als maximale Wertabschätzung seiner Konkurrenten. Gibt i ein Gebot in der Höhe von w' ab und gewinnt damit die Auktion, so maximiert er seinen Ertrag mit $e_i = w_i - w'$.

Bei der Vickrey-Auktion ist es für den einzelnen Bieter i sinnvoll, ein Gebot b_i in Höhe seiner eigenen Wertabschätzung w_i abzugeben: Zum Beleg betrachtet man das höchste zu erwartende Gebot der Konkurrenz als b'. Für den Fall, dass $b' < b_i$ ist, gewinnt i die Auktion und zahlt b' als Preis. Sein positiver Ertrag beläuft sich damit auf $e_i = w_i - b'$. Eine Verringerung von b_i würde den Ertrag nicht steigern, da dieser unabhängig von der Höhe des eigenen Gebots ist – sie würde allerdings das Risiko erhöhen, bei der Versteigerung nicht zum Zug zu kommen (falls $b_i < b' < w_i$). Für den Fall $b' > w_i$ verliert i die Auktion, da er ansonsten einen negativen Ertrag e_i erwirtschaften würde. Damit ist die optimale Bietstrategie bei der Vickrey-Auktion $b_i = w_i$ und somit unabhängig von Annahmen über die Wertabschätzungen der anderen Bieter.

Bei einer Englischen Auktion bildet ebenfalls die eigene Wertabschätzung w_i die Grenze für das Maximalgebot. Hier scheidet der vorletzte Bieter also bei einem Gebot in Höhe seiner eigenen Wertabschätzung w' aus. Der verbleibende Bieter erwirbt das Gut zu einem minimal höheren Betrag als w'.

[Vic61] und [Mye81] zeigen, dass – unter der Bedingung, dass alle Bieter die oben beschriebene rationale Strategie verfolgen – der Erlös der Auktion für alle oben vorgestellten Auktionstypen gleich ist (*Revenue-Equivalence-Theorem*). Damit unterscheiden sich die Verfahren für die anzuwendende Bieterstrategie lediglich in den zu treffenden Annahmen und Abschätzungen.

6.5.2.3 Terminierung

Ausschreibung und Vickrey-Auktion terminieren zu einem vom Auktionator vorab festgelegten Zeitpunkt. Zum Terminierungszeitpunkt evaluiert der Auktionator die eingegangenen Gebote und wählt den Gewinner aus. Später eingehende Gebote finden keine Berücksichtigung.

Die Holländische Auktion terminiert nach Eingang eines Gebotes, spätestes aber dann, wenn ein vom Auktionator zuvor festgelegter Mindestpreis erreicht ist. Damit kann die maximale

Dauer der Holländischen Auktion aus der maximalen Anzahl der Gebotsreduktionsschritte r und der Zeitspanne Δt zwischen zwei Schritten ermittelt werden:

$$t_{max} = \Delta t \times r$$

Die Englische Auktion hat kein im Voraus bestimmbares zeitliches Ende. Im Prinzip ist es möglich, dass Bieter einander bei ausreichendem Kapital beliebig lange überbieten. Oft wird versucht, lange Auktionslaufzeiten und kleinschrittiges Überbieten durch Festlegung eines Minimalpreises und von Mindeststufen zur Erhöhung eines Gebots zu vermeiden.

6.5.2.4 Kommunikationsaufwand

Für die Betrachtung des Kommunikationsaufwands wird im Folgenden angenommen, dass das Versenden einer Broadcast-Nachricht durch Abbildung auf n Einzelnachrichten realisiert wird (Worst-Case-Betrachtung).

Betrachtet man die Anzahl der zur Durchführung einer Auktion notwendigen Nachrichten für die unterschiedlichen Ansätze, so kann man erhebliche Unterschiede feststellen. Zunächst ergibt sich bei einer Auktion mit $n + 1$ Teilnehmern (n Bieter, 1 Auktionator) ein für alle Protokolle gleicher Grundaufwand von $2n$ Nachrichten zur Bekanntgabe der Auktion und zur Bekanntgabe des Ausgangs der Auktion.

Für die Englische Auktion entsteht pro Gebot ein Aufwand von n Nachrichten zur Benachrichtigung aller Teilnehmer über das abgegebene öffentliche Gebot, insgesamt werden für G Gebotsrunden also $(2 + G) \times n$ Nachrichten verschickt.

Bei der Holländischen Auktion sind zwei unterschiedliche Vorgehensweisen denkbar: In Variante (1) versendet der Auktionator regelmäßig den aktuellen Preis an alle Teilnehmer, was mit n Nachrichten pro Runde zu Buche schlägt. Variante (2) arbeitet mit synchronen Uhren und funktioniert nur bei kurzer Nachrichtenübertragungszeit richtig: Der Auktionator teilt zu Beginn der Auktion mit, in welchem Zeitabstand der Gebotsbetrag um welchen Wert sinkt, danach werden seitens des Auktionators keine weiteren Nachrichten mehr verschickt. Jeder Bieter rechnet den jeweilig zu zahlenden Betrag mit und gibt zur entsprechenden Zeit sein Gebot ab. Die Abgabe eines Gebots erfordert bei der Holländischen Auktion eine einzelne Nachricht, sodass bei Variante (1) die Gesamtzahl der zu versendenden Nachrichten wesentlich von der Anzahl der Runden ohne Gebot R abhängt. Insgesamt ergibt sich ein Aufwand von $(R + 2) \times n + 1$, wobei davon auszugehen ist, dass in der Praxis $R \gg 1$ gilt, da mit einem deutlich überhöhten Startpreis begonnen wird. Bei Variante (2) wird unabhängig von der Anzahl der Runden eine feste Anzahl von Nachrichten versandt: $2n + 1$.

Die Anzahl der zu versendenden Nachrichten für Ausschreibung und Vickrey-Auktion sind gleich: Zur Abgabe der verdeckten Gebote von n Bietern sind n Nachrichten notwendig, insgesamt also $3n$ Nachrichten. Damit sind Ausschreibung und Vickrey-Auktion genauso effizient wie die Englische und die Holländische Auktion in Variante (1) im optimalen Fall.

6.5.2.5 Fazit

Da sich der Auktionserlös für die vorgestellten Verfahren bei Verwendung einer rationalen Strategie nicht unterscheidet, kann die Auswahl eines geeigneten Verfahrens anhand der Kriterien Kommunikationsaufwand, Terminierung und Gebotsabgabe erfolgen.

Bezogen auf den verursachten Kommunikationsaufwand erscheinen die zweite Variante der Holländischen Auktion, die Ausschreibung und die Vickrey-Auktion als geeignete Verfahren. Die zweite Variante der Holländischen Auktion verursacht den geringsten Kommunikationsaufwand, stellt dafür aber Anforderungen bezüglich der Uhrensynchronisation. Ausschreibung und Vickrey-Auktion kommen – wie in Abschnitt 6.5.2.4 gezeigt – auf jeden Fall mit der Nachrichtenanzahl aus, die bei Englischer und der ersten Variante der Holländischen Auktion minimal für einen Verkauf benötigt wird.

Im Hinblick auf die Reaktionsgeschwindigkeit des Systems erscheint es ratsam, Verfahren mit deterministischer Laufzeit zu wählen, was wiederum Holländische Auktion, Ausschreibung und Vickrey-Auktion begünstigt. Hier hängt die Reaktionsgeschwindigkeit des Systems ausschließlich vom – in gewissen Grenzen – frei wählbaren Terminierungszeitpunkt ab.

Die Art der Gebotsabgabe bestimmt wesentlich die durch einen Bieter lokal zu verfolgende rationale Strategie. Vergleicht man Holländische Auktion, Ausschreibung und Vickrey-Auktion in diesem Aspekt, so erweist sich die Abhängigkeit der Gebotshöhe von der eigenen Wertabschätzung des zu erwerbendes Gutes als Vorteil der Vickrey-Auktion. Diese ermöglicht damit die Umsetzung einer Bietstrategie, die lediglich auf der Auswertung lokaler Informationen beruht.

Aus den oben genannten Gründen wird die Vickrey-Auktion als Koordinationsverfahren für kooperierende QoS-Manager gewählt und entsprechend [AC04] erweitert. Im Weiteren wird beschrieben, wie das Verfahren innerhalb der in Kapitel 5 beschriebenen Architektur eingesetzt wird.

6.5.3 Anwendung eines Auktionsverfahrens auf das Übertragen von Antwortzeit-SLO-Anteilen

6.5.3.1 Zuordnung von Geldbeträgen

Für diese Arbeit werden in Bezug auf die Strategie einzelner QoS-Manager die folgenden Annahmen getroffen:

- Jeder QoS-Manager strebt danach, das ihm zur Verfügung stehende Kapital zu vermehren.

- Die Wertabschätzung eines zu ersteigernden Gutes erfolgt unabhängig von der Höhe des verfügbaren Kapitals.

Voraussetzung für die Koordination von QoS-Managern mithilfe von Auktionen ist, dass ein System zur Zuordnung von Geldbeträgen definiert wird, durch das die einzelnen QoS-Manager ihr Verhandlungskapital beziehen.

Zunächst muss grundsätzlich geklärt werden, ob QoS-Manager „Schulden“ machen dürfen, d. h. ob sie im Rahmen eines Ersteigerungsprozesses mehr Kapital einsetzen dürfen als ihnen zur Verfügung steht. Unter der Annahme, dass sich QoS-Manager verschulden dürfen, kann ein sehr einfaches System definiert werden, das jedoch über keinerlei Möglichkeiten zur Beeinflussung des Systemverhaltens verfügt:

QoS-Managern wird kein Kapital zugeordnet. Sie ersteigern und versteigern Antwortzeit-SLO-Anteile allein aufgrund ihrer eigenen Wertabschätzung. Während des Systembetriebs sammeln einzelne QoS-Manager dadurch u. U. hohe „Schulden“ an, andere verfügen über einen positiven Kapitalstock. Für den Betreiber der SOA lassen sich aus der resultierenden Kapitalverteilung Schlüsse zur zukünftigen Dimensionierung von Ressourcen ziehen. Ein solcher, rein beobachtender Ansatz vernachlässigt allerdings die Möglichkeiten zur Steuerung des Systemverhaltens, die prinzipiell durch ein auktionsgetriebenes Koordinationsverfahren gegeben sind. Daher wird im Folgenden eine zweite Variante vorgestellt, bei der einzelnen QoS-Managern das Ansammeln von „Schulden“ verwehrt ist, wodurch QoS-Manager ohne ausreichende Kapitalisierung in der Höhe ihrer Gebotsabgabe begrenzt werden.

Um die Gesamt-Dienstgüte einer SOA zu optimieren, müssen vor allem die Dienste und Workflows ihre Dienstgüteziele erreichen können, denen auf Geschäftsebene eine hohe Wichtigkeit zugeordnet wird. Wie in Abschnitt 5.2 beschrieben, werden einzelnen Workflows global Prioritäten zugeordnet. Die Priorität eines Workflows ist den beteiligten Diensten bekannt, die Wichtigkeit eines Dienstes für das Gesamtsystem ergibt sich aus den Prioritäten der Workflows, an denen der Dienst beteiligt ist.

Gleichzeitig werden durch die Nutzung eines Dienstes und die damit verbundenen Ressourcen Kosten verursacht, die z. B. mithilfe eines Kostenmodells (vgl. [SDKH05b, SDKH05a]) ermittelt werden können. Es liegt daher nahe, über ein geeignetes Modell eine Bezahlung von Diensten pro ausgeführtem Auftrag (und entsprechend der erbrachten Dienstgüte) zu realisieren. Mithilfe von Utility Functions ([LRJ04]) kann ein solches „Bezahlmodell“ global innerhalb der SOA realisiert werden. Durch die Zuordnung von Kapital entsprechend der Wichtigkeit eines Dienstes für das Gesamtsystem kann die auktionsbasierte Koordination von QoS-Managern auf einfache Weise an Geschäftsziele gekoppelt werden.

6.5.3.2 Abläufe bei Auktionator und Bietern

Mit dem in Abschnitt 6.3 vorgestellten Verfahren bestimmt ein einem Dienst zugeordneter QoS-Manager seine lokale Situation und entscheidet, ob er überschüssige Antwortzeit-SLO-Anteile an andere QoS-Manager übertragen kann oder ob zusätzliche Antwortzeit-SLO-Anteile benötigt. Abhängig von dieser Entscheidung tritt der QoS-Manager nun entweder als Anbieter auf und initiiert eine Auktion, oder er betätigt sich in Auktionen anderer QoS-Manager als Bieter. Jeder QoS-Manager verfügt sowohl über Auktionator- als auch über Bieterfunktionalität. Im Weiteren wird daher ein QoS-Manager in Anbieter-Rolle vereinfachend als Auktionator bezeichnet.

Abbildung 6.12 zeigt die Interaktion von Bieter und Auktionator bei einer Vickrey-Auktion.

Auktionator Der Auktionator kündigt potenziellen Bietern die Auktion zunächst über eine Nachricht an. Bestandteile dieser Nachricht sind eine Beschreibung des zu veräußernden Antwortzeit-SLO-Anteils und der Zeitpunkt, zu dem Gebote beim Auktionator eingegangen sein müssen.

Nach Absenden der Auktionsankündigung wartet der Auktionator auf eingehende Gebote bzw. auf den Ablauf der Bietfrist für die Auktion. Geht ein Gebot ein, wird dieses zunächst auf Gültigkeit überprüft (ein ungültiges Gebot könnte beispielsweise einen Preis beinhalten, der geringer als die eigene Wertabschätzung des zu verkaufenden Antwortzeit-SLO-Anteils ist, oder sich auf eine bereits abgelaufene Auktion beziehen). Ist das Gebot gültig, wird es registriert. Der Auktionator wartet im Anschluss wieder auf den Eingang weiterer Gebote oder den Ablauf der Bietfrist.

Ist die Bietfrist abgelaufen, wird die Auktion für beendet erklärt. Sind keine Gebote registriert worden, kann der Antwortzeit-SLO-Anteil nicht verkauft werden, und die Auktion ist gescheitert. Als Reaktion auf eine gescheiterte Auktion kann der Auktionator die eigene Wertabschätzung überarbeiten und ggf. einen weiteren Auktionsversuch starten.

Wurden gültige Gebote registriert, werden Gewinner und zu zahlender Preis ermittelt und eine Gewinnbenachrichtigung versandt. Nach Eingang der Zahlungsbestätigung des Gewinners wird der Antwortzeit-SLO-Transfer bestätigt und der verkaufte Antwortzeit-SLO-Anteil vom eigenen Antwortzeit-SLO abgezogen.

Bieter Empfängt ein Bieter eine Auktionsankündigung, analysiert er zunächst, ob seine lokale Managementsituation einen Bedarf an zusätzlichen Antwortzeit-SLO-Anteilen erkennen lässt. Ist kein Antwortzeit-SLO-Bedarf vorhanden, kehrt der Bieter in seinen Ausgangszustand zurück und wartet auf weitere Auktionsankündigungen. Benötigt er zusätzliche Antwortzeit-SLO-Anteile, analysiert er das eingegangene Auktionsangebot inhaltlich. Passt das Angebot zum eigenen Bedarf, bewertet der Bieter den zu erwerbenden Antwortzeit-SLO-Anteil und

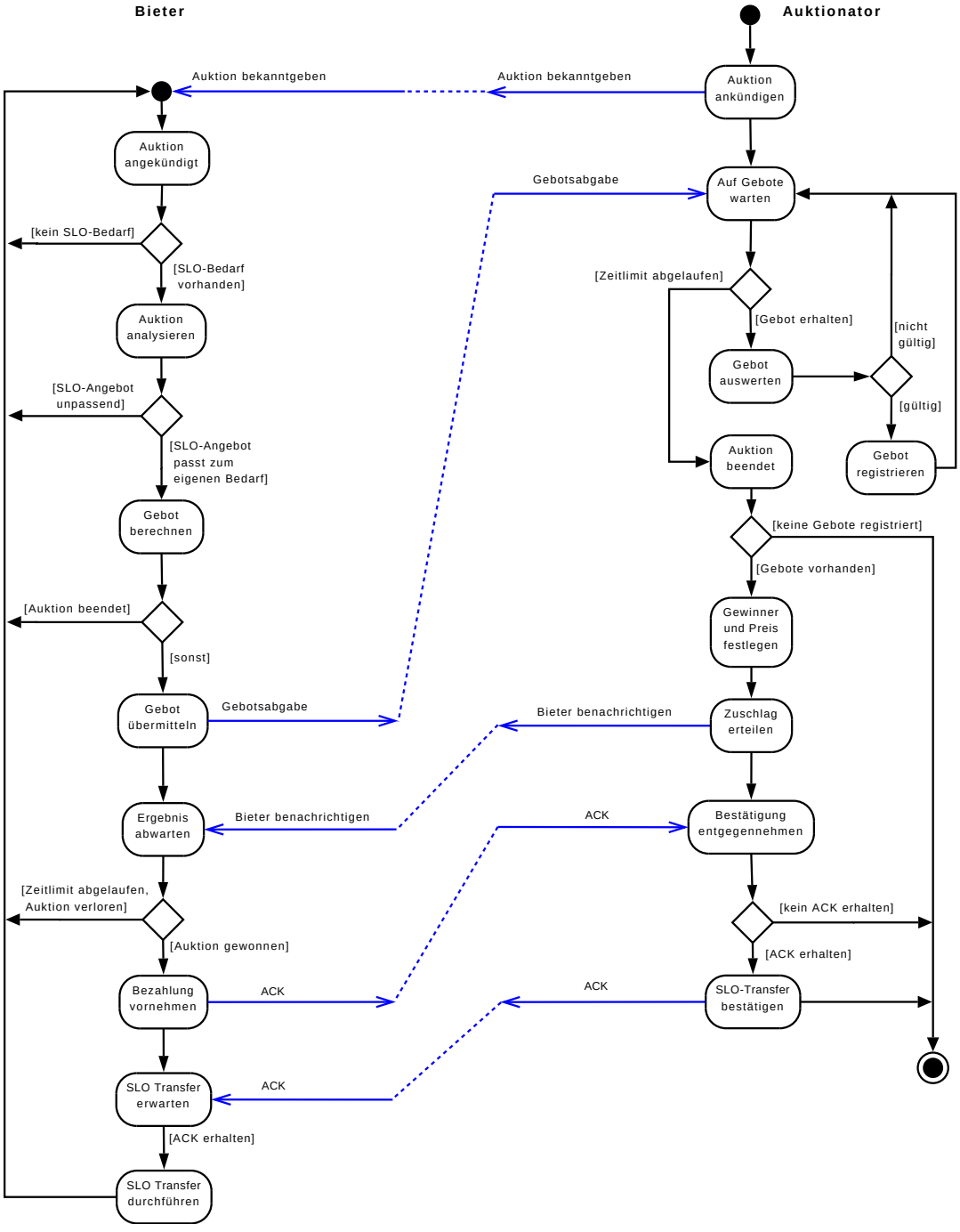

Abbildung 6.12: Abläufe bei Bieter und Auktionator einer Vickrey-Auktion

bestimmt so die Höhe des eigenen Gebotes. Ist die Gebotsfrist zu diesem Zeitpunkt noch nicht abgelaufen, übermittelt der Bieter sein Gebot an den Auktionator.

Im Anschluss wartet der Bieter auf Eingang der Nachricht über den Ausgang der Auktion. Erhält er eine Gewinnnachricht vom Auktionator mit Angabe des zu zahlenden Preises, so bestätigt er diese und transferiert damit die geforderte Summe zum Auktionator. Anschließend wartet der Bieter auf eine Nachricht, die den Eingang des Geldes beim Auktionator bestätigt und den Antwortzeit-SLO-Transfer quittiert. Nun kann der Bieter sein eigenes Antwortzeit-SLO um den ersteigerten Wert erweitern. Hat ein Bieter die Auktion verloren, wartet er auf weitere Auktionsankündigungen.

6.5.4 Unterlagerte Kommunikationsstrukturen

6.5.4.1 Logische Kommunikation

Zur Übertragung von Antwortzeit-SLO-Anteilen finden Auktionen zwischen Auktionatoren und Bietern in den durch einen Workflow-Pfad abgesteckten Grenzen statt. Zur Umsetzung der in Abschnitt 6.4.5 beschriebenen Übertragung von Antwortzeit-SLOs zwischen QoS-Managern mithilfe der Vickrey-Auktion ist es notwendig, Auktionator- und Bieterrollen im System zuzuordnen und Kommunikationsbeziehungen entsprechend der zuvor identifizierten Workflow-Pfade (vgl. Abschnitt 6.4.2.5) zu strukturieren.

Als Anbieter und Interessenten für Antwortzeit-SLO-Anteile können in einer Auktion ausschließlich die QoS-Manager agieren, die einem durch eine Workflow-Basisaktivität repräsentierten Dienst zugeordnet sind. Im Folgenden werden die Kommunikationsstrukturen für die einzelnen in Abschnitt 6.4.5 vorgestellten Situationen diskutiert.

Auktion innerhalb einer einfachen Workflow-Sequenz Im einfachsten Fall besteht ein Workflow-Pfad lediglich aus einer Sequenz von Basisaktivitäten. In diesem Fall agieren die QoS-Manager der durch die Basisaktivitäten repräsentierten Dienste als Anbieter von und Interessenten für Antwortzeit-SLO-Anteile. Ein QoS-Manager kann über die in Abschnitt 5.4.3 vorgestellte Schnittstelle zur Inter-Manager-Kommunikation direkt mit den anderen beteiligten QoS-Managern kommunizieren. Der Ablauf einer solchen Auktion wird im Folgenden beispielhaft diskutiert.

> **Beispiel 6.9:** In der in Beispiel 6.5 dargestellten Sequenz würden QoS-Manager direkt miteinander kommunizieren, wobei einer die Auktionator-Rolle A und die anderen die Rolle der Bieter B_n annehmen würden. Dieses Verhalten ist in Abbildung 6.13 dargestellt. A gibt die Auktion den Bietern bekannt. Als Reaktion darauf gibt B_1 ein Gebot der Höhe y ab. Der potenzielle Bieter B_2 ist nicht interessiert und gibt kein Angebot ab. Nach Ablauf der Bietfrist evaluiert A die

eingegangenen Angebote und erteilt dem Höchstbietenden B_1 den Zuschlag zum Preis des zuvor festgelegten Mindestpreises, weil nur ein Gebot eingegangen ist.

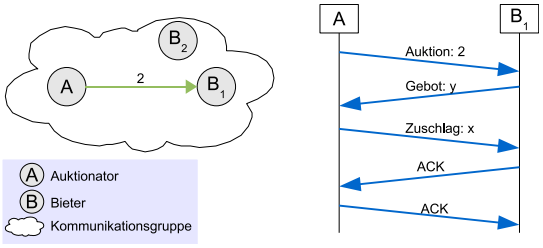

Abbildung 6.13: Ablauf einer Auktion innerhalb einer Sequenz

Auktionen über Grenzen von Workflow-Strukturen Normalerweise setzt sich ein Workflow-Pfad nicht nur – wie zuvor dargestellt – aus einer einfachen Sequenz zusammen, sondern wird durch eine komplexere Struktur realisiert. In Abschnitt 6.4.5.2 wird für die Übertragung von Antwortzeit-SLO-Anteilen über Sequenz-Grenzen hinweg ein zweischrittiges Verfahren vorgestellt. Innerhalb einer strukturierten Aktivität (z. B. Sequenz oder Schleife) kommunizieren die Partner direkt miteinander, die Kommunikation mit übergeordneten Strukturen wird durch einen Koordinator übernommen.

Bei der Abbildung dieses Verfahrens auf eine Vickrey-Auktion kommunizieren Auktionator und Bieter innerhalb einer strukturierten Aktivität des Workflow-Pfades direkt. Dies entspricht dem im vorherigen Abschnitt diskutierten Vorgehen. Zur Verknüpfung der einzelnen Strukturen wird eine Proxy-Rolle eingeführt, die innerhalb einer Struktur in einer Bieter-Rolle und innerhalb der anderen Struktur in der Auktionator-Rolle auftritt. Auf diese Weise leitet der Proxy die Nachrichten des Auktionators an zusätzliche potenzielle Bieter weiter und gibt deren Angebote an den Auktionator zurück. Beispiel 6.10 beschreibt das Vorgehen für zwei ineinander geschachtelte Workflow-Sequenzen.

Beispiel 6.10: Abbildung 6.14 zeigt die Abbildung der in Beispiel 6.6 beschriebenen Vorgehensweise auf eine Vickrey-Auktion.

Zunächst kommuniziert der Auktionator A_1 die anstehende Versteigerung eines Antwortzeit-SLO-Anteils an die für ihn sichtbaren Bieter B_1 und B_2. Die Rolle des Bieters B_2 wird durch den Proxy P erfüllt, der die Auktionsankündigung in der Rolle A_2 an die für ihn sichtbaren Bieter B_3 und B_4 weiterleitet. Geben B_3 oder B_4 Gebote ab, richten sich diese an A_2, der aus ihrer Sicht den Auktionator repräsentiert. Eingehende Gebote werden durch den Proxy dann an den eigentlichen Auktionator A_1 weitergeleitet. Gewinnt einer der Bieter B_3 oder B_4 die Auktion, so ergeht der Zuschlag zunächst an den Proxy, der die Nachricht dann an

den Gewinner weiterleitet und auch die weitere Kommunikation zwischen Auktionator und Bieter unterstützt.

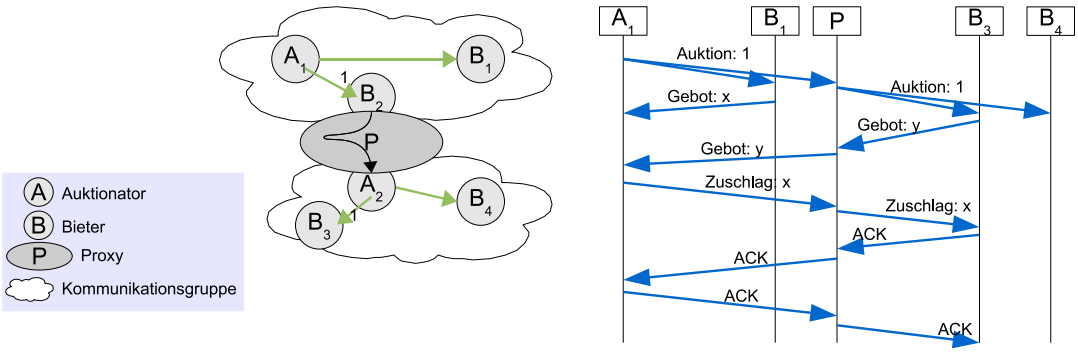

Abbildung 6.14: Ablauf einer Auktion über Sequenz-Grenzen hinweg

In der Baumdarstellung eines Workflow-Pfades (vgl. Abschnitt 6.4.2) übernimmt der Proxy damit die Position einer strukturierten Aktivität. Abhängig von der Art der repräsentierten strukturierten Aktivität sind durch den Proxy ggf. zusätzliche Aufgaben zu erledigen. Tabelle 6.2 gibt darüber einen Überblick.

STRUKTURIERTE AKTIVITÄT	PROXY-VERHALTEN
Sequenz	Nachrichten werden unverändert weitergeleitet (vgl. Beispiel 6.10).
Schleife	Auktionsankündigungen und Gebote werden entsprechend der zu erwartenden Anzahl der Schleifendurchläufe multipliziert bzw. dividiert.
Fallunterscheidung, Parallelausführung	Synchronisation der betroffenen Workflow-Pfade notwendig.

Tabelle 6.2: Aufgaben des Auktions-Proxys, abhängig von der jeweils repräsentierten strukturierten Aktivität

Im Folgenden werden die Aufgaben des Proxys für die aufwändigeren Fälle genauer diskutiert.

Behandlung von Schleifen Bei der Auktion von Antwortzeit-SLO-Anteilen über die Grenzen einer Schleife hinweg führt der Proxy (neben der bereits im vorhergehenden Abschnitt erwähnten Unterstützung der Kommunikation zwischen Auktionator und Bieter) die in

Abschnitt 6.4.6 beschriebenen Umrechnungen für die Ankündigung von Auktionen und die Gebotsabgabe durch.

Beispiel 6.11: Abbildung 6.15 zeigt den Ablauf einer Vickrey-Auktion, bei der Antwortzeit-SLO-Anteile über eine Schleifengrenze hinweg versteigert werden. Der Auktionator A_1 bietet $i = 10$ Antwortzeit-SLO-Anteile zum Verkauf und informiert die ihm bekannten Bieter B_1 und B_2 darüber. B_1 gibt ein Angebot der Höhe x ab, als B_2 agiert der Proxy P in einer Bieter-Rolle. Nach Erhalt der Auktionsankündigung rechnet P diese entsprechend der zu erwartenden Anzahl der Schleifendurchläufe $n = 10$ um $(\frac{i}{n} = \frac{10}{10} = 1)$ und informiert in der Rolle A_2 die Bieter B_3 und B_4 über die Auktion des Antwortzeit-SLO-Anteils.

B_3 bietet den Betrag y für den Antwortzeit-SLO-Anteil. Dieses Gebot wird von P wiederum auf die Gesamtsumme umgerechnet $(y \times i = 10y)$ und an den Auktionator weitergeleitet. Da im Beispiel $10y > x$ gilt, erhält B_3 vom Auktionator den Zuschlag für den Betrag x. Die weiteren Nachrichten werden durch P nicht modifiziert, sodass B_3 als Sieger der Auktion den unveränderten Preis x zahlt.

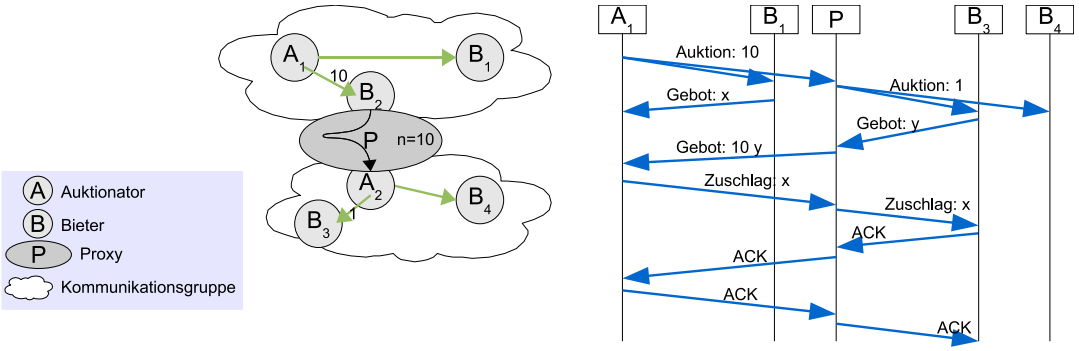

Abbildung 6.15: Ablauf einer Auktion über Schleifengrenzen hinweg

Durch die vom Proxy vorgenommenen Anpassungen der Auktionsankündigung und der Gebote wird sichergestellt, dass ein Bieter die Bewertung einer Auktionsankündigung unabhängig von der eigenen Position in der Workflow-Struktur vornehmen kann. Der Proxy korrigiert die Höhe des abgegebenen Gebots entsprechend, damit einem Bieter aufgrund seiner Position innerhalb der Schleife kein Nachteil gegenüber Bietern außerhalb der Schleife entsteht.

Im Fall des Gewinns der Auktion zahlt der Bieter jedoch den unmodifizierten Preis, im Gegenzug erhält ein Dienst innerhalb einer Schleife im Fall einer Utility-funktionsgestützten Zuordnung von Kapital bei einem Dienstaufruf (vgl. Abschnitt 6.5.3.1) pro Workflow-Durchlauf entsprechend mehr Kapital als ein Dienst außerhalb der Schleife.

Befindet sich der Auktionator innerhalb der Schleife, übersetzt der Proxy die Auktionsankündigung und eingehende Angebote in umgekehrter Weise – ein erfolgreicher Bieter außerhalb

der Schleife erhält angebotene Antwortzeit-SLO-Anteile dementsprechend zu sehr günstigen Konditionen.

Synchronisation unterschiedlicher Workflow-Pfade Parallelausführungen und Fallunterscheidungen in der Workflow-Struktur werden durch Proxy-Elemente repräsentiert, die Auktionator- und Bieterrollen für jeden betroffenen Workflow-Pfad einnehmen können. Aufgabe des Proxys ist es, die Antwortzeit-SLOs der Teilpfade so zu synchronisieren, dass trotz der Modifikation des Antwortzeit-SLOs eines Teilpfades das Gesamt-Antwortzeit-SLO des Workflows nicht verletzt wird.

Der Proxy tritt dabei in der übergeordneten Struktur und in den einzelnen untergeordneten Teilpfaden bei Bedarf sowohl als Auktionator als auch als Bieter auf. Der Proxy orientiert sich in seinem Verhalten am größten Antwortzeit-SLO, das für einen Teilpfad vergeben wurde (vgl. Gleichung 6.10).

Benötigt dieser Teilpfad weitere Antwortzeit-SLO-Anteile, agiert der Proxy wie zuvor für Auktionen über Sequenz-Grenzen beschrieben. Allen anderen Teilpfaden wird nach erfolgreicher Transaktion – wie in Abbildung 6.16 dargestellt – die Differenz zum maximalen Antwortzeit-SLO durch den Proxy als Auktionator ebenfalls zur Verfügung gestellt. Auf diese Weise kann eventuell vorhandener zusätzlicher Antwortzeit-SLO-Bedarf anderer Pfade konform zu Regel **R3** abgedeckt werden.

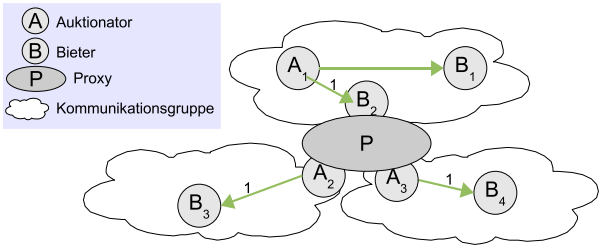

Abbildung 6.16: Rolle der Proxy-Komponente bei der Synchronisation unterschiedlicher Teilpfade

Benötigt die übergeordnete Struktur Antwortzeit-SLO-Anteile aus den einzelnen Teilpfaden, so können diese nur abgegeben werden, falls in allen Teilpfaden mit maximalem Antwortzeit-SLO entsprechende Anteile abgegeben werden können. Dies entspricht der in Regel **R4** geforderten Vorgehensweise.

Im Gegensatz zu der für Sequenzen und Schleifen beschriebenen zustandslosen Proxy-Funktionalität muss der Proxy zur Synchronisation unterschiedlicher Workflow-Teilpfade einen Überblick über die Gesamt-Antwortzeit-SLOs der einzelnen Teilpfade behalten. Er kann damit nicht zustandslos realisiert werden.

6.5.4.2 Abbildung der Rollen auf physikalische Komponenten

Zur Realisierung der in Abschnitt 6.5.4.1 auf logischer Ebene beschriebenen Auktionen müssen die dort definierten Interaktionsrollen auf im System vorhandene physikalische Komponenten abgebildet werden. Abschnitt 5.3 spezifiziert die Zuordnung von jeweils einer QoS-Manager-Instanz zu einer einem SOA-Dienst unterlagerten Implementierung. Auf logischer Ebene werden innerhalb einer Auktion `Invoke`-Aktivitäten durch eine Auktionator- bzw. Bieter-Rolle repräsentiert, strukturierte Aktivitäten sind als Proxy mit Auktionator- bzw. Bieter-Rolle in jedem untergeordneten Teilpfad sowie in der übergeordneten Struktur vertreten. Im Folgenden wird die Abbildung dieser Rollen getrennt für Basisaktivitäten und strukturierte Aktivitäten diskutiert.

Basisaktivitäten Die Vorgehensweise zur Abbildung der Auktionator- und Bieter-Rollen von `Invoke`-Aktivitäten auf vorhandene QoS-Manager ist vergleichsweise einfach: Die Rollen werden auf Kommunikations-Plugins (vgl. Abbildung 5.7) des für die unterlagerte Dienstimplementierung verantwortlichen QoS-Managers abgebildet.

Ein Dienst – und damit auch dessen QoS-Manager – kann sowohl gleichzeitig durch unterschiedliche Workflows genutzt werden als auch innerhalb desselben Workflows an unterschiedlichen Stellen und mit potenziell unterschiedlichen Dienstgüteanforderungen eingebunden werden. Durch den QoS-Manager wird die Unterstützung paralleler Dienstgüteanforderungen (wie in Abschnitt 5.4.3.3 beschrieben) realisiert. Innerhalb eines QoS-Managers sind die notwendigen Kommunikations-Plugins mehrfach instanziiert, sodass sich eine Plugin-Instanz jeweils genau einer Position in einem bestimmten Workflow zuordnen lässt.

Strukturierte Aktivitäten Strukturierte Aktivitäten können nicht direkt einem bestimmten Dienst zugeordnet werden. Logisch sind sie an die Ausführung eines Workflows und damit an eine Workflow-Engine gekoppelt. Je der Anzahl der durch eine Workflow-Engine kontrollierten parallel ablaufenden Workflows und deren Komplexität existiert in einer SOA eine große Anzahl von strukturierten Aktivitäten. Eine einheitliche Zuordnung von Auktionator- und Bieter-Rollen strukturierter Aktivitäten zum QoS-Manager eines Workflows ist prinzipiell möglich, erscheint aber aus Skalierbarkeitserwägungen heraus nicht praktikabel.

Zur Abbildung der für strukturierte Aktivitäten definierten Proxy-Rollen wird daher an dieser Stelle ebenfalls ein dezentraler Ansatz gewählt: Die Akteure innerhalb einer Struktur bestimmen aus ihrer Mitte einen Akteur, der als *Koordinator* die Kommunikation mit der übergeordneten Struktur übernimmt. Als Akteure gelten einzelnen Diensten zugeordnete QoS-Manager. Der zum Koordinator bestimmte QoS-Manager instanziiert zusätzlich zu den eigenen Kommunikations-Plugins weitere Plugins, die die Auktionator- und Bieter-Rollen des Proxys übernehmen. Diese agieren dabei unabhängig von der lokal durch den QoS-Manager implementierten Strategie zur Optimierung seiner Dienstgüte.

In übergeordneten Strukturen werden ebenfalls Koordinatoren bestimmt, und auf einen einem Dienst zugeordneten QoS-Manager abgebildet. Auf diese Weise kann die Kommunikationsstruktur zur Realisierung eines Auktionssystems dezentral für den gesamten Workflow etabliert werden.

6.5.4.3 Definition von Kommunikationsgruppen

Bei der Abbildung logischer Kommunikationsrollen auf physikalische Strukturen stellt die beschränkte Sicht eines Dienstes auf seine Umgebung ein Problem dar: Der einzelne Dienst stellt seine Funktionalität lediglich über eine definierte Schnittstelle bereit. Er besitzt keine Informationen über den Kontext seiner Nutzung. Die Art und Weise, in der Dienste durch einzelne Workflows benutzt werden, ist in der Workflow-Beschreibung festgelegt und somit ausschließlich der ausführenden Workflow-Engine und dem für einen Workflow zuständigen QoS-Manager bekannt.

Die Definition von Kommunikationsgruppen für einzelne Dienste wird daher vom QoS-Manager eines Workflows vorgenommen und den QoS-Managern der Dienste im Rahmen der Aushandlung initialer Antwortzeit-SLOs mitgeteilt. Zur Unterstützung der Koordinator-Funktionalität wird die Zugehörigkeit jedes QoS-Managers zu eindeutig benannten, typisierten Kommunikationsgruppen als Stapel formuliert. Jeder QoS-Manager tritt zunächst der auf seinem Stapel zuoberst angegebenen Gruppe bei und entfernt diese aus dem Stapel. Innerhalb jeder Gruppe wird ein Koordinator bestimmt, der die Gruppenmitglieder in der übergeordneten Struktur repräsentiert. Wird ein QoS-Manager zum Koordinator seiner Gruppe bestimmt, so tritt er zusätzlich der nun zuoberst auf seinem Stapel liegenden übergeordneten Gruppe bei, in der wiederum ein Koordinator bestimmt wird, usw. Ist der eigene Stapel leer, existiert keine weitere übergeordnete Struktur im Workflow.

Kommunikationsgruppen sind typisiert: Der Typ einer Gruppe ergibt sich aus der der Gruppe übergeordneten strukturierten Aktivität wie in Tabelle 6.2 beschrieben. Aus dem Gruppentyp leitet der Koordinator die Art des zu instanziierenden Proxys ab (vgl. Abschnitt 6.5.4.1).

> **Beispiel 6.12:** Abbildung 6.17 zeigt die Definition von Gruppen für die in Abbildung 6.6 gezeigten Workflow-Pfade.
>
> Im gezeigten Beispiel werden die einzelnen Pfade durch `Flow`-Aktivitäten unterschieden. In der Grafik sind die Aktivitäten der unterschiedlichen Pfade farbig gekennzeichnet. Aktivitäten, die von mehreren Pfaden genutzt werden, sind farblich nicht hervorgehoben. Der Workflow ist entsprechend seiner Struktur in die Ebenen $A - F$ eingeteilt. Die Bezeichnung einer Gruppe ergibt sich aus der Ebene und einer Zahl, die immer dann inkrementiert wird, wenn das Elternelement den Graphen in unterschiedliche Pfade unterteilt (Fallunterscheidungen und Parallelausführungen).

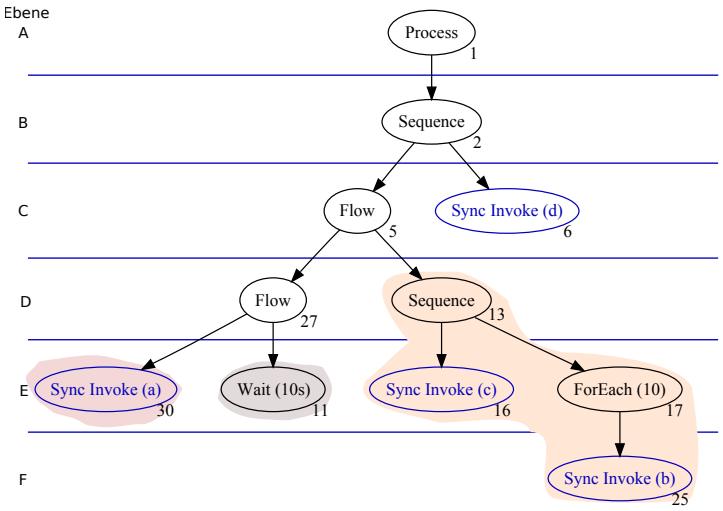

Abbildung 6.17: Identifikation von Kommunikationsgruppen auf Basis der Workflow-Struktur

Tabelle 6.3 zeigt die vom Workflow-QoS-Manager für den Beispiel-Workflow spezifizierten Stapel, die den QoS-Managern der durch die dargestellten `Invoke`-Aktivitäten repräsentierten Dienste zugeordnet werden.

INVOKE(A)	INVOKE(B)	INVOKE(C)	INVOKE(D)
E1	F1	E3	C1
D1	E3	D2	B1
C1	D2	C1	A1
B1	C1	B1	
A1	B1	A1	
	A1		

Tabelle 6.3: Zuordnung von Kommunikationsgruppen mithilfe von Stapeln

Abbildung 6.18 zeigt, wie auf Basis der in Tabelle 6.3 gezeigten Stapel schrittweise eine Kommunikationsinfrastruktur zwischen den QoS-Managern aufgebaut wird.

In Teil 1 der Abbildung erkennt man, wie die QoS-Manager der Aktivitäten `Invoke a`, `Invoke b`, `Invoke c` und `Invoke d` zunächst den Kommunikationsgruppen beitreten, die auf ihrem jeweiligen Stapel an oberster Stelle definiert sind. Im Beispiel erzeugt so jeder QoS-Manager eine eigene Kommunikationsgruppe, der er selbst beitritt. Dann erfolgt die Auswahl eines Koordinators für jede Gruppe; da im Beispiel jeweils nur ein Mitglied pro Gruppe existiert, wird

jeder QoS-Manager Koordinator seiner Gruppe.

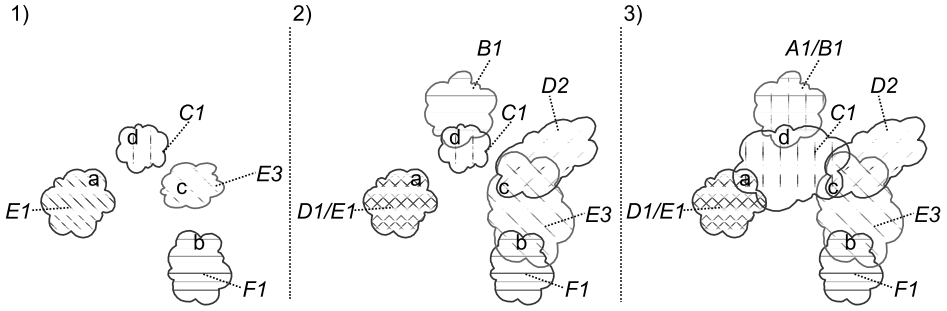

Abbildung 6.18: Schrittweise Auswahl von Koordinatoren

Der Koordinator tritt nun jeweils der übergeordneten Gruppe bei (vgl. Teil 2 der Abbildung), beispielsweise betritt der QoS-Manager von `b` die Gruppe *E3*, in der sich bereits `c` befindet. Da zuvor `c` bereits zum Koordinator von *E3* gewählt wurde, tritt `c` seinerseits der übergeordneten Gruppe *D2* bei. Die QoS-Manager von `a` und `d` treten als Koordinatoren den jeweils übergeordneten Gruppen bei. In allen neu erzeugten Gruppen werden wieder Koordinatoren bestimmt.

In Teil 3 der Abbildung treten diese Koordinatoren erneut der jeweils übergeordneten Gruppe bei: `d` wird Mitglied von *A1*, `a` und `c` treten *C1* bei. Da für *C1* bereits `d` als Koordinator bestimmt wurde und für *A1* keine weitere übergeordnete Gruppe existiert, sind nun alle notwendigen Kommunikationsgruppen zur Koordination der QoS-Manager in diesem Workflow etabliert.

Für den Beispiel-Workflow existiert damit mit der Gruppe *C1* eine direkte Kommunikation zwischen den QoS-Managern der durch die Aktivitäten `Invoke a`, `Invoke c` und `Invoke d` repräsentierten Dienste und mit der Gruppe *E3* eine direkte Kommunikation zwischen `Invoke b` und `Invoke c`.

Durch die lose Kopplung der Dienste in einer SOA kann es zur Laufzeit nötig werden, für einen nicht mehr verfügbaren oder ausgefallenen Gruppenkoordinator einen Ersatz zu bestimmen. Gründe für den Ausfall eines Koordinators können – neben dem geplanten Ersatz eines Dienstes durch eine andere Instanz – beispielsweise Kommunikationsstörungen sein, die zum (temporären) Zerfall der Kommunikationsgruppe führen.

Der Wechsel von Koordinatoren in Gruppen, die Sequenzen oder Schleifen repräsentieren, gestaltet sich unproblematisch, da die jeweilige Proxy-Funktionalität wie in Abschnitt 6.5.4.1 beschrieben zustandslos realisiert werden kann. Koordinatoren von Gruppen, die eine Fallunterscheidung oder Parallelausführung implementieren, halten als Zustand die Antwortzeit-SLOs, die für die jeweiligen Teilpfade aktuell gültig sind. Damit der Transfer des Zustands an

einen neuen Koordinator problemlos abgewickelt werden kann, muss das unterlagerte Gruppenkommunikationssystem gewährleisten, dass die aktuelle Antwortzeit-SLO-Übersicht jeweils an alle Gruppenmitglieder verteilt wird.

6.5.4.4 Anforderungen an das unterlagerte Gruppenkommunikationssystem

Um die oben beschriebene Koordination von QoS-Managern mithilfe von Auktionsprotokollen ohne zentrale Instanz realisieren zu können, wird zur Kommunikation zwischen einzelnen QoS-Managern ein Gruppenkommunikationssystem verwendet. Grundlegende Begriffe der Gruppenkommunikation sind detailliert in [Pow96] und [CKV01] beschrieben.

Gruppenkommunikationssysteme unterscheiden sich wesentlich in den bereitgestellten Abstraktionen und den Garantien, die in Bezug auf Nachrichtenauslieferung und -reihenfolge gegeben werden können. Im Folgenden wird für die einzelnen Kommunikationsschritte untersucht, welche Garantien ein Gruppenkommunikationssystem minimal zur Abwicklung der beschriebenen Protokolle bereitstellen muss.

Verwaltung der Gruppenmitgliedschaft Das Gruppenkommunikationssystem muss allen Knoten eine einheitliche Gruppensicht zur Verfügung stellen. Ausfall oder Gruppenbeitritt eines Knotens müssen durch geeignete Mechanismen festgestellt und an alle Knoten kommuniziert werden. Strenge virtuelle Synchronizität [BJ87] aller Knoten ist für die umzusetzenden Auktionsprotokolle jedoch nicht erforderlich.

Unterstützung der Auktionsprotokolle Bei der Umsetzung der in Abschnitt 6.5.3 beschriebenen Auktionsprotokolle ist neben Punkt-zu-Punkt-Kommunikation einzelner QoS-Manager phasenweise auch Multicast-Kommunikation innerhalb einer Gruppe sinnvoll. Insbesondere betrifft dies Situationen, in denen Gruppen von QoS-Managern über den Beginn einer Auktion informiert werden.

Für die Vickrey-Auktion ist kein komplexes Kommunikationsprotokoll notwendig – kausale Beziehungen zwischen Nachrichten beschränken sich auf das Beantworten einer eingegangenen Nachricht, wobei die Antwort eines Bieters als Unicast an den Auktionator versandt wird (privates Gebot). Ebenso ist die Reihenfolge des Nachrichteneingangs (der Gebote einzelner Bieter) beim Auktionator irrelevant, da der Gewinner nach Ablauf der Bietfrist und nicht basierend auf der Eingangsreihenfolge der Gebote ausgewählt wird.

Zur Unterstützung der Kommunikationsvorgänge bei der Vickrey-Auktion ist es hinreichend, wenn durch das Gruppenkommunikationssystem ein unzuverlässiger Multicast-Mechanismus für geschlossene Gruppen angeboten wird. Empfängt ein Knoten eine Auktionsankündigung nicht, kann er sich nicht als Bieter beteiligen. Dies gefährdet jedoch nicht die Konsistenz des

Protokolls. Die Benachrichtigung über den Ausgang einer Auktion wird über Unicast-Nachrichten abgewickelt (vgl. Abbildung 6.12), ebenso die Übertragung von Antwortzeit-SLO-Anteilen an den oder die Gewinner. Für Unicasts muss die unterlagerte Kommunikationsinfrastruktur einen zuverlässigen Mechanismus bereitstellten.

Bestimmung von Koordinatoren Das Gruppenkommunikationssystem muss die Bestimmung eines Gruppenkoordinators ermöglichen: Um häufige Koordinatorenwechsel innerhalb einer Gruppe zu vermeiden, soll das Verfahren möglichst beständige Ergebnisse liefern, d. h. beispielsweise bei einer erneuten, durch den Beitritt neuer Gruppenmitglieder ausgelösten Koordinatorenauswahl nach Möglichkeit den bisherigen Koordinator als solchen bestätigen.

Weiterhin muss das Gruppenkommunikationssystem geeignete Abstraktionen bereitstellen, die es ermöglichen, innerhalb eines Koordinators gehaltene Informationen über die durch die einzelnen Teilpfade beanspruchten Teil-Antwortzeit-SLOs an die anderen Gruppenmitglieder zu kommunizieren.

6.6 Dynamische Priorisierung von Diensten

Eine weitere Möglichkeit zur Verbesserung der Gesamtdienstgüte einer SOA besteht in der Privilegierung einzelner Dienste (und damit auch aller diesen Dienst nutzenden Workflows) in Bezug auf die zur Verfügung gestellten Ressourcen. In Abschnitt 6.6.1 wird zunächst kurz auf den Begriff der Virtualisierung eingegangen, im Anschluss wird die Priorisierung einzelner SOA-Dienste unter Nutzung von Technologien zur System-Virtualisierung beschrieben.

6.6.1 System-Virtualisierung

Erste Systeme zur Virtualisierung von Rechner-Ressourcen wurden bereits in den 1960er-Jahren für Großrechner entwickelt. Grundlage der Virtualisierungstechnologie ist die Schaffung einer Indirektionsstufe zwischen Betriebssystem und unterlagerter Hardware, die als *Virtual Machine Monitor (VMM)* bezeichnet wird. Ein VMM unterstützt das Erzeugen und Betreiben mehrerer so genannter *virtueller Maschinen (VMs)*, die sich die Ressourcen der unterlagerten Hardware teilen, sich typischerweise jedoch gegenüber einem innerhalb der VM installierten Betriebssystem funktional genau wie eine physikalische Maschine verhalten. Aufgabe des VMMs ist es, einzelnen VMs bei Bedarf konfliktfrei Zugang zu physikalischen Ressourcen zu verschaffen. Dabei ist je nach verwendetem VMM die dynamische Priorisierung einzelner Maschinen durch Änderungen in der Zuordnung von Ressourcen zur Laufzeit möglich.

Heute werden Virtualisierungstechnologien in Rechenzentren in der Regel zur Konsolidierung von Diensten auf wenigen, leistungsfähigen Maschinen eingesetzt. Ziele sind dabei sowohl die

Einsparung von Server Hardware-Ressourcen als auch die Reduktion des Energieverbrauchs durch einen insgesamt höheren Hardware-Nutzungsgrad. Weiterhin soll eine Erhöhung von Zuverlässigkeit und Flexibilität der bereitgestellten Dienste erreicht werden. Aktuelle VMM-Ansätze erlauben teilweise die nahezu unterbrechungsfreie Migration von VMs zwischen unterschiedlichen physikalischen Hosts eines Pools zur Laufzeit. Dadurch können durch Hardware-Defekte verursachte Ausfallzeiten minimiert werden (vgl. [BS95, CMR06]).

6.6.2 Abbildung von Diensten auf virtuelle Maschinen

Grundlage für die hier beschriebene dynamische Zuordnung von Ressourcen zu einzelnen Diensten ist die Abbildung von SOA-Diensten mit ihrer unterlagerten Implementierung auf einzelne VMs. Vereinfachend wird im Folgenden von einer 1:1-Zuordnung zwischen Dienst und VM ausgegangen, womit in der weiteren Beschreibung der Begriff VM synonym für den innerhalb dieser VM laufenden Dienst verwendet werden kann. In der Praxis kann die Implementierung eines Dienstes auf unterschiedliche VMs verteilt werden, wodurch beispielsweise Redundanzanforderungen nachgekommen werden kann.

Der einem Dienst zugeordnete QoS-Manager kann auf der gleichen VM laufen wie der Dienst selbst oder auf Ressourcen außerhalb des VM Pools angesiedelt sein.

6.6.3 Architektur

Abbildung 6.19 zeigt die Architektur des hier vorgestellten Systems zur dynamischen Ressourcenzuordnung für SOA-Dienste. Das System kann einen Pool von n physikalischen Maschinen (PMs) verwalten, die die Ablaufumgebung für insgesamt m VMs bilden. Die einzelnen SOA-Dienste (in der Abbildung als *Service* bezeichnet) werden – wie im vorherigen Abschnitt beschrieben – auf diese VMs abgebildet. Als Teil jeder PM ist im unteren Bereich der Grafik der VMM abgebildet, dieser verfügt über Schnittstellen für Monitoring und Management der Virtualisierungsumgebung.

Jeder SOA-Dienst wird – wie in Abschnitt 5.4.3 dargestellt – durch den ihm zugeordneten QoS-Manager überwacht. In Abbildung 6.19 sind diese QoS-Manager als *VM-Manager*-Ebene dargestellt. Da Dienst und VM als Einheit betrachtet werden, ist jeder QoS-Manager neben dienstspezifischer Sensorik auch mit Sensorik zur Überwachung von unterschiedlichen Parametern der VM (z. B. CPU- und Speicherauslastung) ausgestattet. Die Strategie-Optimierungskomponente des QoS-Managers beurteilt aufgrund der erhobenen Messwerte und in Verbindung mit dem vereinbarten SLA (wie in Abschnitt 6.3 beschrieben) über wünschenswerte Korrekturen an der Ressourcenzuordnung. Müssen aus Sicht des QoS-Managers Korrekturen vorgenommen werden, informiert dieser über eine Aktorik-Schnittstelle den für den Ressourcen-Pool verantwortlichen QoS-Manager (in der Abbildung als *Pool-Manager* bezeichnet).

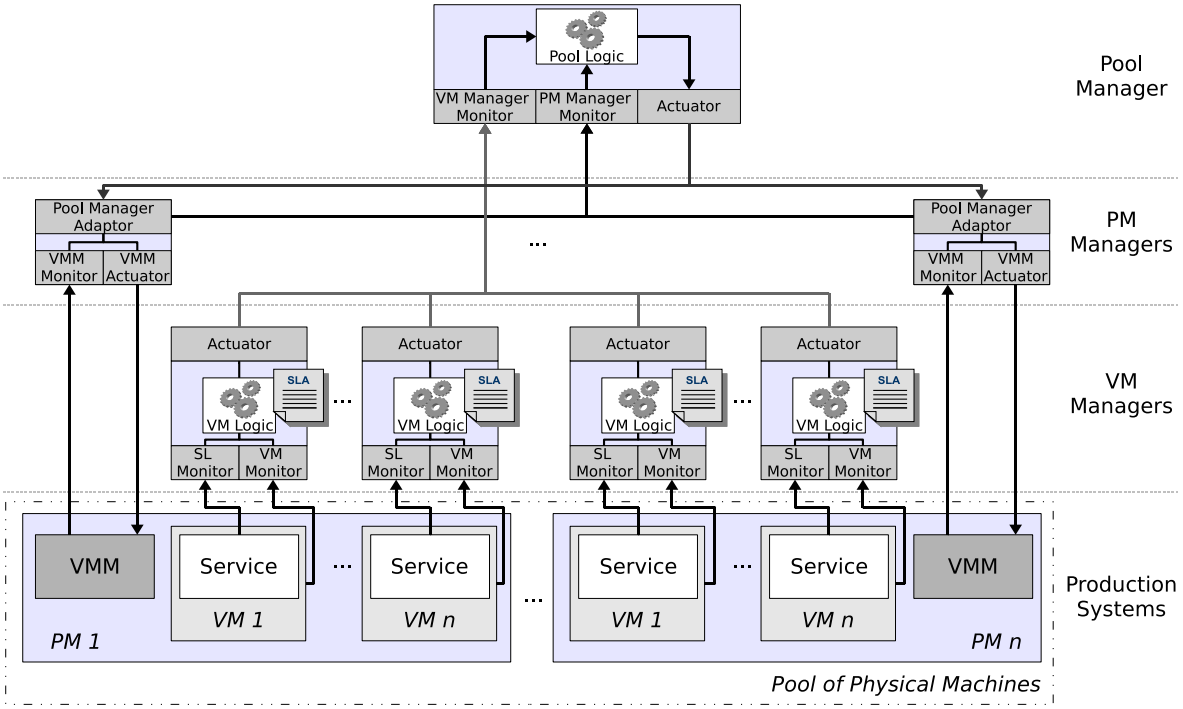

Abbildung 6.19: Management-System zur dynamischen Ressourcenzuordnung für SOA-Dienste (aus [SMK08])

Jeder physikalischen Maschine innerhalb des Pools ist ein *PM-Manager* zugeordnet. Der PM-Manager überwacht die Auslastung der ihm zugeordneten physikalischen Maschine und informiert den Pool-Manager über die aktuelle Ressourcennutzung. Auf Anweisung des Pool-Managers führt der PM-Manager über die VMM-Schnittstelle der Maschine unterschiedliche Management-Aktionen durch. Beispiele für typische Management-Aktionen sind Rekonfiguration von Ressourcenzuordnungen für einzelne VMs (z. B. Zuordnung weiterer CPU-Anteile, Erweiterung oder Reduktion der zugeordneten RAM-Menge, ...). Weiterhin kann der Pool-Manager mithilfe der durch den VMM bereitgestellten Schnittstellen VMs auf andere physikalische Maschinen migrieren.

Der Pool-Manager ist die für die Ressourcenzuordnung innerhalb des Pools von PMs letztendlich zuständige Instanz. Durch die von PM-Managern und VM-Managern übermittelten Informationen kann der Pool-Manager sich eine globale Sicht auf den Zustand des von ihm verwalteten Pools erzeugen. Basierend auf dieser Sicht entscheidet er über Rekonfiguration und Migration von VMs innerhalb des Pools. Dabei bezieht er die einzelnen Diensten auf Geschäftsebene zugeordneten Prioritäten in seine Entscheidung mit ein. Entscheidungen zur Rekonfiguration oder Migration von VMs werden mithilfe des PM-Managers der unterlagerten physikalischen Maschine ausgeführt.

Sowohl Pool-Manager als auch PM-Manager sind auf Basis einer modularen QoS-Manager-Plattform realisiert. Durch die Nutzung der modularen Plattform kann das in der Abbildung als *Pool-Logic* bezeichnete Strategiemodul entsprechend den Geschäftsanforderungen mit unterschiedlichen Algorithmen versehen werden. Die Entwicklung geeigneter Algorithmen zur Ressourcenzuordnung ist allerdings nicht trivial, da der Pool-Manager mit einem NP-harten, multidimensionalen Rucksackproblem konfrontiert ist. In [Mar08] werden Algorithmen zur Optimierung der Ressourcenzuordnung entwickelt, die mithilfe von Techniken der lokalen Suche zwar keine optimale, aber doch eine akzeptable Lösung erreichen. In der Arbeit wird der Gesamtansatz detailliert beschrieben und evaluiert.

6.7 Festlegung initialer Antwortzeit-SLOs

Im Folgenden wird ein einfacher Ansatz zur Festlegung initialer Antwortzeit-SLOs für die Aktivitäten eines Workflows vorgestellt. Der Ansatz basiert auf einer Worst-Case-Betrachtung von bisher gemessenen Antwortzeiten. In Kombination mit den in Abschnitt 4.2 zitierten Arbeiten [AP07] und [SPJ08] ist jedoch auch ein Verfahren auf Basis empirischer Verteilungsfunktionen denkbar.

6.7.1 Annahmen bezüglich der Workflow-Struktur

Die hier vorgestellte Methode zur initialen Berechnung von Antwortzeit-SLOs für individuelle Dienste basiert auf der Untersuchung der Struktur eines Workflows. Um eine erfolgreiche Antwortzeit-SLO-Zuordnung realisieren zu können, werden folgende Annahmen getroffen.

- Es werden ausschließlich Basisaktivitäten betrachtet, für die eine maximale Abarbeitungsdauer festgelegt werden kann (Worst-Case-Betrachtung).
- Für Schleifen muss die maximale Anzahl von Durchläufen aus der Workflow-Beschreibung ermittelt werden können oder vorab bekannt sein.

Aktivitäten, bei denen die Spezifikation einer maximalen Abarbeitungsdauer nicht möglich ist (z. B. Benutzerinteraktion), kann durch das hier vorgestellte Verfahren kein Antwortzeit-SLO zugewiesen werden. Allgemein eignen sich Workflow-Teilpfade mit derartigen Aktivitäten nicht für den hier entwickelten Ansatz zum automatisierten QoS-Management, da für Komponenten mit unvorhersagbarem Verhalten auch kein Regelungsalgorithmus implementiert werden kann. Das Verfahren (und damit auch die initiale Antwortzeit-SLO-Festlegung) kann aber in einem solchen Fall auf die übrigen Workflow-Teilpfade angewandt werden.

6.7.2 Vorbereitung

Ziel der Festlegung initialer Antwortzeit-SLOs ist es, für jeden innerhalb des Workflows aufgerufenen Dienst das maximale Antwortzeit-SLO zu bestimmen, sodass das sich daraus für den Workflow ergebende Antwortzeit-SLO die für den Workflow getroffene Dienstgütevereinbarung nicht verletzt. Für die Berechnung individueller Antwortzeit-SLOs muss daher zunächst der längste Pfad des Workflows (vgl. Definition 6.3) ermittelt werden.

Definition 6.3: LÄNGSTER AUSFÜHRUNGSPFAD
Die Länge eines Ausführungspfads ergibt sich aus der Gesamtzeit, die für das Durchlaufen dieses Pfades einzuplanen ist (Gesamtantwortzeit). Der längste Ausführungspfad eines Workflows ist definiert als eine Menge von Aktivitäten $W_P \subseteq A$, die durch eine Menge von Kanten $L_P \subseteq K$ verbunden sind und für deren Durchlaufen die zu erwartende Antwortzeit in Bezug auf die möglichen Ausführungspfade in W maximal ist.

Für die Bestimmung des längsten Ausführungspfades gemäß Definition 6.3 können Fallunterscheidungen und Parallelausführungen im Prozess äquivalent behandelt werden, in beiden Fällen ist jeweils nur der Teilbaum mit der längsten Ausführungszeit relevant.

Die Bestimmung des Teilbaums mit der längsten Ausführungszeit ist nur im Kontext des gesamten Prozesses möglich, da sich die Antwortzeit aus der Kombination einer festen Zeitspanne (für die Dauer von `Wait`-Aktivitäten und die Antwortzeit-SLOs von `Invoke`-Aktivitäten mit nicht veränderbarem SLA) und einem variablen Zeitanteil pro `Invoke`-Aktivität mit veränderbarem Antwortzeit-SLO ergibt. Der variable Zeitanteil ergibt sich aus der Aufteilung der (nach Abzug der für den Pfad ermittelten festen Zeitspanne) übrigbleibenden Zeit auf die einzelnen `Invoke`-Aktivitäten mit variablem Antwortzeit-SLO.

In Algorithmus 6.3 werden vorbereitende Schritte durchgeführt: Zunächst wird allen `Wait`-Aktivitäten ihre Dauer als unveränderliches Antwortzeit-SLO zugeordnet (vgl. Zeilen 1-3). Dann wird den `Invoke`-Aktivitäten mit unveränderlichem Antwortzeit-SLO der in ihrem SLA definierte Antwortzeitwert (Zeilen 4-6) und `Invoke`-Aktivitäten mit veränderbarem Antwortzeit-SLO – soweit verfügbar – der maximale bekannte historische Antwortzeitwert zugeordnet (vgl. Zeilen 7-13).

Im Folgenden wird das rekursive Vorgehen zur Bestimmung des längsten Pfades für die einzelnen Structured Activities eines BPEL-Prozesses besprochen. Die für die einzelnen Aktivitäten definierten Regeln werden rekursiv beim Durchlaufen jedes Teilbaumes ausgeführt. Parallelausführungen und Fallunterscheidungen werden dabei, wie in Abschnitt 6.4.2.5 beschrieben, in separate Teilbäume für jeden möglichen Pfad des Workflows aufgelöst. Durch Vergleich der Wurzelknoten der einzelnen Bäume kann dann der längste Pfad bestimmt werden. Da der Algorithmus somit nur auf Bäume angewandt wird, in denen keine Parallelausführungen und

Algorithmus 6.3 Vorbereitung der Antwortzeit-SLO-Zuordnung

Input: Z `//Menge d. Wait-Akt. d. unters. Pfades (vgl. Formel 6.5)`
I `//Menge d. Invoke-Akt. d. unters. Pfades (vgl. Formel 6.3)`

1: **for all** $(x \in Z)$ **do**
2: $\gamma.slo \Leftarrow$ getWaitPeriod(x);
3: **end for**
4: **for all** $(x \in I | x.fix =$**true**$)$ **do**
5: $x.slo \Leftarrow$ getfixedSlaMaximumResponseTime(x);
6: **end for**
7: **for all** $(x \in I | x.fix =$**false**$)$ **do**
8: **if** (hasHistoricalResponseTimes(x)) **then**
9: $x.slo \Leftarrow$ getMaximumResponseTime(x);
10: **else**
11: $x.slo \Leftarrow \varnothing$;
12: **end if**
13: **end for**

keine Fallunterscheidungen auftreten, wird im Folgenden lediglich auf die Behandlung von Sequenzen und Schleifen eingegangen.

6.7.3 Sequenzen

Die Gesamtantwortzeit einer Workflow-Sequenz s ergibt sich aus der Summe der Antwortzeiten der in der Sequenz enthaltenen Kindaktivitäten. Damit gilt, dass, falls eine Sequenz Teil des längsten Pfades ist, auch alle direkten Kindaktivitäten der Sequenz zum längsten Pfad des Prozesses gehören.

Algorithmus 6.4 zeigt die Analyse der Antwortzeit einer Sequenz. In den Zeilen 1 bis 3 werden die einzelnen Elemente des die Sequenz beschreibenden Tupels initialisiert.

Im Anschluss wird die Menge der unmittelbaren Kindelemente der Sequenz durchlaufen und analysiert. Dabei wird die Dauer von `Wait`-Aktivitäten und Antwortzeit-SLOs für `Invoke`-Aktivitäten mit, unveränderlichem SLA zur Variablen t_{fix} der Sequenz hinzuaddiert. Alle anderen `Invoke`-Aktivitäten werden zur Menge U hinzugefügt. Beinhaltet eine Sequenz eine weitere komplexe Aktivität, so wird deren t_{fix} mit der Anzahl ihrer Iterationen multipliziert und zum Wert der t_{fix}-Variablen addiert (Zeile 13). Weiterhin wird der Iterationszähler für jedes Element der Menge U der komplexen Kindaktivität korrigiert und im Anschluss das Element der Menge U der Sequenz hinzugefügt (Zeile 16).

Der gleiche Algorithmus kann auch für andere strukturierte Elemente des Workflows angewandt werden, etwa `Process`- oder `Scope`-Aktivitäten. Diese unterscheiden sich von der

Sequenz dadurch, dass sie meist nur ein Kindelement (i. d. R. eine strukturierte Aktivität) aufweisen, sodass die für das Kindelement errechneten Werte vom Algorithmus übernommen werden können.

Algorithmus 6.4 Analyse der Antwortzeit einer Sequenz

Input: s `//Die zu untersuchende Sequenz`
B `//Menge der Basisaktivitäten des untersuchten Pfades`
C `//Menge der komplexen Aktivitäten des untersuchten Pfades`
Output: $s.iterations$ `//Anzahl der Iterationen der Sequenz`
$s.t_{fix}$ `//Fix vergebener Zeitanteil der Sequenz`
$s.U$ `//Menge der Kindelemente mit variablen SLO-Anteilen`

1: $s.iterations \Leftarrow 1$;
2: $s.t_{fix} \Leftarrow 0$;
3: $s.U \Leftarrow \{\}$;
4: **for all** $(x \in \Gamma^+(s))$ **do**
5: **if** $(x \in B)$ **then**
6: **if** $(x.fix =$**false**$)$ **then**
7: $x.iterations \Leftarrow s.iterations$;
8: add$(\gamma, s.U)$;
9: **else**
10: $s.t_{fix} \Leftarrow s.t_{fix} + \text{slo}(x)$;
11: **end if**
12: **else if** $(x \in C)$ **then**
13: $s.t_{fix} \Leftarrow s.t_{fix} + (x.iterations \times x.t_{fix})$;
14: **for all** $(\alpha \in x.U)$ **do**
15: $\gamma \Leftarrow \alpha$;
16: $\gamma.iterations \Leftarrow x.iterations \times \alpha.iterations$;
17: add$(\gamma, s.U)$;
18: **end for**
19: **end if**
20: **end for**

6.7.4 Schleifen

Die Gesamtantwortzeit einer Schleife ergibt sich aus der Antwortzeit des Schleifenkörpers, multipliziert mit der Anzahl der maximal zu erwartenden Schleifendurchläufe. Im Unterschied zur Sequenz kann eine Schleife in WS-BPEL nur ein direktes Kindelement haben, dieses kann eine beliebige Aktivität sein.

Damit lässt sich der zur Analyse der Antwortzeit einer Sequenz entworfene Algorithmus 6.4 mit einer geringfügigen Änderung auch zur Analyse von Schleifen benutzen: In Zeile 1 muss nur die für die Schleife maximal zu erwartende Anzahl von Durchläufen eingesetzt werden.

6.7.5 Auswahl des längsten Pfads

Es sei t_{sla} die maximale Antwortzeit, die als Dienstgüte für den gesamten Workflow garantiert werden soll. Zur Verringerung der Wahrscheinlichkeit einer Antwortzeit-SLO-Überschreitung kann der Betreiber des WfMS einen bestimmten Anteil von t_{sla} als Sicherheitszugabe t_{sec} zu reservieren. Die maximal an die beteiligten Dienste übertragbare Zeit t_{slo} ist damit wie folgt definiert:

$$t_{slo} := t_{sla} - t_{sec} \tag{6.16}$$

Wurden die in den vorherigen Abschnitten beschriebenen Berechnungen für alle Pfade durchgeführt, kann durch Vergleich der jeweiligen Wurzelelemente p festgestellt werden, welcher der Bäume den längsten Pfad des Prozesses repräsentiert. Zum Vergleich werden für jeden Pfad zunächst die folgenden Größen berechnet:

$$\begin{aligned}
&N := \{x \in U | \operatorname{slo}(x) = \varnothing\} \\
&\text{falls } |N| > 0 \\
&\quad t_{avg} = \frac{t_{slo} - (t_{fix} + \sum_{a_i \in U} \operatorname{slo}(a_i))}{|N|} \\
&\text{sonst} \\
&\quad t_{avg} = 0 \\
&t_{all} = t_{fix} + (|N| \times t_{avg}) + \sum_{a_i \in U} \operatorname{slo}(a_i) \\
&t_{rest} = t_{slo} - t_{all}
\end{aligned}$$

Die Menge N beinhaltet alle Aktivitäten der Menge U (vgl. Algorithmus 6.4), für die keine historischen Antwortzeitinformationen vorliegen. Ist diese Menge nicht leer, so wird t_{avg} als Antwortzeit-SLO-Vorschlag für jedes Element der Menge N aus der nach Abzug aller bekannten Zeitspannen verbleibenden Zeit gebildet. Ist N leer, wird für den betrachteten Pfad $t_{avg} = 0$ gesetzt. Im Anschluss wird die durch einen Pfad insgesamt benötigte Zeit t_{all} ermittelt und die Differenz t_{rest} zum Antwortzeit-SLO ermittelt. $t_{rest} > 0$ tritt nur dann ein, wenn $|N| = 0$ ist, da ansonsten $t_{all} = t_{slo}$ gilt (vgl. Definition von t_{avg}).

Im Anschluss werden die Teilbäume in eine Liste L eingefügt. Diese wird nach den folgenden Kriterien sortiert:

1. *Absteigend nach* $t_{all.p}$
   Pfade mit großem Zeitbedarf werden am Beginn der Liste einsortiert.
2. *Pfade mit gleichem Wert für* $t_{all.p}$ *jeweils aufsteigend nach* t_{avg}
   Pfade, für die – im Vergleich mit anderen Pfaden mit gleichem Gesamtzeitbedarf – durchschnittlich sehr kleine Antwortzeit-SLOs vergeben wurden, werden weiter vorne eingeordnet.
3. *und absteigend nach* $|N|$
   Bei gleichem Gesamtzeitbedarf und gleicher durchschnittlicher Antwortzeit-SLO-Größe werden längere Pfade weiter vorne eingeordnet.

Nach abgeschlossener Sortierung ist das erste Element der Liste L der Teilbaum mit dem längsten Ausführungspfad. Durch die Sortierung wird sichergestellt, dass bei mehreren äquivalenten Ausführungspfaden mit maximalem $t_{all.p}$ derjenige an erster Stelle steht, bei dem gleichzeitig das einzelnen Diensten zuzuordnende Antwortzeit-SLO klein ist und dieses Antwortzeit-SLO einer großen Anzahl von Diensten im Pfad zuzuordnen ist.

6.7.6 Festlegung initialer Antwortzeit-SLOs

Vorbedingung für die Gültigkeit einer gefundenen Lösung ist, dass für alle Pfade $p \in L$ gilt:

$$p.t_{all} \leq t_{slo}$$

Ist dies nicht der Fall, so kann der Workflow die an ihn gestellten Dienstgüteanforderungen nicht erfüllen, da aufgrund historischer Antwortzeitinformationen anzunehmen ist, dass die beteiligten Dienste die geforderte Gesamtantwortzeit nicht erfüllen können.

Mithilfe von Algorithmus 6.5 werden nun den einzelnen Aktivitäten Antwortzeit-SLOs zugeordnet. Hierzu wird die sortierte Liste L – beginnend mit dem längsten Pfad – durchlaufen und allen Aktivitäten ohne Antwortzeit-SLO, eines in Höher ihrer t_{avg}-Variablen zugeordnet. Damit wird sichergestellt, dass keine Aktivität ein Antwortzeit-SLO erhält, das die Vorgaben eines anderen Pfades verletzt.

Abschließend werden die Zuordnungen an die jeweiligen Dienste kommuniziert. Tritt dabei ein Problem auf (z. B. weil ein Dienst ein entsprechendes Antwortzeit-SLO ablehnt), müssen die Vorgaben für diesen Dienst angepasst und die Algorithmen 6.3 – 6.5 erneut durchlaufen werden.

Algorithmus 6.5 Zuordnung von Antwortzeit-SLOs zu einzelnen Aktivitäten eines Workflows

1: $F = \{\}$ //Menge der Aktivitäten, denen bereits ein SLO zugeordnet wurde
2: **for all** $(x \in L)$ **do**
3: **for all** $(\gamma \in U_x \setminus F)$ **do**
4: **if** $\mathrm{slo}(\gamma) = \varnothing$ **then**
5: $\gamma.slo \Leftarrow x.t_{avg}$; //Zuordnung des SLOs
6: **else**
7: $\gamma.slo \Leftarrow \gamma.slo + \frac{t_{rest}}{|U_x|}$; //ggf. Erhöhung des für einen Dienst vorgeschlagenen SLOs, damit $t_{all} = t_{slo}$ erreicht wird
8: **end if**
9: $F \Leftarrow F \bigcup \gamma$; //Hinzufügen von γ zur Menge F
10: **end for**
11: **end for**

6.8 Zusammenfassung und Fazit

In diesem Kapitel wurden mehrere alternative Ansätze zur dezentralen Verbesserung des Dienstgüteverhaltens einer SOA vorgestellt.

In Abschnitt 6.2 wurden zunächst Voraussetzungen und Rahmenbedingungen für die vorgestellten Kooperationsverfahren diskutiert. In Abschnitt 6.3 wurde allgemein beschrieben, wie ein QoS-Manager eine Bewertung seiner lokalen Situation vornehmen kann. Diese Bewertung bildet sowohl die Basis für lokale Management-Entscheidungen als auch für die Teilnahme an den in den weiteren Abschnitten vorgestellten Kooperationsverfahren.

Den Schwerpunkt des Kapitels bildet das in Abschnitt 6.4 beschriebene Verfahren zur Kooperation von QoS-Managern von Diensten, die in einen Workflow, für den zuvor ein Gesamt-Antwortzeit-SLO vereinbart wurde, eingebunden sind. Voraussetzung für den beschriebenen Ansatz ist eine initiale Vereinbarung von individuellen Antwortzeit-SLOs für einzelne Workflow-Aktivitäten auf Basis des Gesamt-Antwortzeit-SLOs des Workflows. Mithilfe des entwickelten Verfahrens können zur Laufzeit ungenutzte Anteile dieser individuellen Antwortzeit-SLOs zwischen den QoS-Managern einzelner Dienste ausgetauscht werden, ohne dass das insgesamt für den Workflow zu erfüllende Antwortzeit-SLO dadurch tangiert wird. Auf diese Weise lässt sich u. U. die Ressourcennutzung innerhalb der SOA verringern, oder global eine Verbesserung der Dienstgüte erreichen. In Abschnitt 6.5 wurden zunächst unterschiedliche Auktionsprotokolle auf ihre Eignung zur Umsetzung des Verfahrens untersucht, im Anschluss wurde ein Ansatz zur Umsetzung des Koordinationsverfahrens auf Basis der Vickrey-Auktion beschrieben.

In Abschnitt 6.6 wurde ein ergänzender Ansatz zur Optimierung der Ressourcennutzung unterschiedlicher Dienste skizziert, der auf der Nutzung von Virtualisierungstechniken beruht.

Er nutzt eine Management-Infrastruktur für einen Pool von virtuellen Maschinen, in der die einer einzelnen Maschine zugeteilten Ressourcen entsprechend einer globalen Priorisierung und individueller Antwortzeit-SLO-Anforderungen justiert werden können. Einzelnen Diensten zugeordnete QoS-Manager wenden sich bei einer Verletzung der an sie gestellten Dienstgüteanforderungen an das Management-System, um die Zuteilung zusätzlicher Ressourcen zu erreichen. Das System evaluiert konkurrierende Anforderungen und priorisiert Dienste entsprechend ihrer Wichtigkeit.

Die beiden vorgestellten Verfahren zur Optimierung der Dienstgüte einer SOA basieren auf einer initial vorzunehmenden Ableitung individueller Antwortzeit-SLOs aus dem (z. B. vom Administrator) global für den Workflow vergebenen Antwortzeit-SLO. Beispielhaft ist ein solches Verfahren, das auf der Analyse der Struktur des gegebenen Workflows beruht, zur Abrundung des Kapitels in Abschnitt 6.7 beschrieben.

Die in den Abschnitten 6.4 und 6.6 beschriebenen Verfahren stehen jedoch nicht in Konkurrenz zueinander. Basierend auf der Bewertung der eigenen Situation kann ein QoS-Manager sich entscheiden, entweder eine Lockerung der eigenen Dienstgütevorgaben anzustreben oder eine Verbesserung der eigenen Dienstgüte durch Zuweisung zusätzlicher Ressourcen zu verfolgen. Eine solche Entscheidung kann – wie in Abschnitt 6.3 beschrieben – basierend auf der Auswertung einer Utility-Funktion getroffen werden.

Teil III

Realisierung

7 Prototypische Umsetzung

In diesem Kapitel wird die prototypische Umsetzung des in den Kapiteln 5 und 6 entwickelten Ansatzes vorgestellt. Die resultierende Implementierung dient als Grundlage für die Evaluation des Gesamtansatzes in Kapitel 8.

In Abschnitt 7.1 wird die Implementierung der in Abschnitt 5.4 beschriebenen QoS-Manager für Workflows und für Dienste anhand konkreter Beispiele vorgestellt: Als gemeinsame technische Grundlage wird in Abschnitt 7.1.1 zunächst das so genannte Selfmanager-Framework eingeführt, in Abschnitt 7.1.2 wird auf die Integration des WS-Agreement-Protokolls eingegangen. Die Realisierung eines QoS-Managers für Workflows wird in Abschnitt 7.1.3 am Beispiel der BPEL-Engine Apache ODE diskutiert. Abschnitt 7.1.4 zeigt dann die Realisierung eines QoS-Managers für Dienste am Beispiel des Managements eines JBoss-Applikationsservers. Eine Erweiterung zum Bereitstellen unterschiedlicher Dienstgüteklassen wird in Abschnitt 7.1.5 vorgestellt.

Abschnitt 7.2 geht detailliert auf die Realisierung der in Kapitel 6 besprochenen Ansätze zur Selbstorganisation von QoS-Managern ein. Für die Evaluation des Ansatzes wird ein Simulator vorgestellt, der des ermöglicht das Verhalten der diskutierten Algorithmen in großen Umgebungen zu beobachten.

7.1 Implementierung der Management-Komponenten

Die in der SLM-Architektur zum Einsatz kommenden QoS-Manager für Workflows und für Dienste werden technisch auf ein gemeinsames modulares Manager-Framework abgebildet. Im Folgenden werden die Elemente des Manager-Frameworks im Detail vorgestellt.

7.1.1 Das Selfmanager-Framework

Abbildung 7.1 zeigt die prinzipielle Architektur des Manager-Frameworks. Es untergliedert sich in eine Kernkomponente (*Manager Core*) und spezialisierte Erweiterungsmodule. Die prinzipielle Struktur des Frameworks ist an [DKK+01] angelehnt.

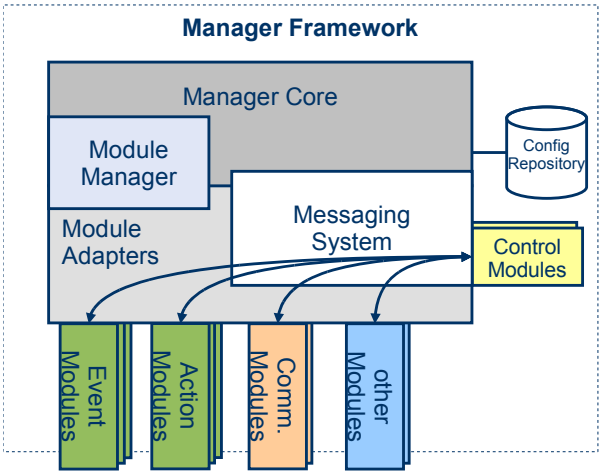

Abbildung 7.1: Modulares Framework zur Realisierung von QoS-Managern

Die Kernkomponente des Frameworks beinhaltet Basisfunktionen zur Verwaltung des Lebenszykluses einzelner Erweiterungsmodule (in der Abbildung als *Module Manager* bezeichnet) und stellt den Modulen eine nachrichtenorientierte Kommunikationsinfrastruktur zur Verfügung. Weiterhin beinhaltet das Framework ein zentrales Konfigurations-Repository.

7.1.1.1 Verwaltung von Erweiterungsmodulen

Geladene Erweiterungsmodule können sich in unterschiedlichen Zuständen befinden. Direkt nach der Instanziierung befinden sie sich im Zustand `DOWN`. In diesem Zustand nehmen sie keine Nachrichten von anderen Modulen entgegen. Nach erfolgreichem Durchlaufen der Initialisierungsphase wechselt ein Modul in den Zustand `UP` und signalisiert dadurch seine Betriebsbereitschaft. Tritt bei der Initialisierung oder im Betrieb ein nicht korrigierbarer Fehler auf, wechselt das Modul in den Zustand `ERROR`. Im `ERROR`-Zustand reagiert es nicht auf Nachrichten anderer Module. Zur Korrektur des Fehlers kann die Plattform versuchen, eine erneute Initialisierung des Moduls durchzuführen.

Jedes Erweiterungsmodul implementiert eine einheitliche Verwaltungsschnittstelle. Über diese können Module initialisiert und beendet werden. Weiterhin kann ein Erweiterungsmodul über die Verwaltungsschnittstelle veranlasst werden, die im Repository abgelegte Konfiguration neu einzulesen und ggf. den Initialisierungsvorgang zu wiederholen. Eine `Check`-Methode dient der Überprüfung der Funktionalität eines Moduls: Ein positiver Rückgabewert signalisiert dem Framework, dass das Erweiterungsmodul und ggf. davon abhängende Komponenten (z. B. Datenbankverbindungen) fehlerfrei funktionieren und das Modul sich im Zustand `UP` befindet.

Erweiterungsmodule können durch den Module Manager zur Laufzeit auch mehrfach instanziiert werden. Jede Instanz wird über eine eindeutige ID referenziert und verfügt über eine eigene Konfiguration im Repository.

7.1.1.2 Initialisierung des Frameworks

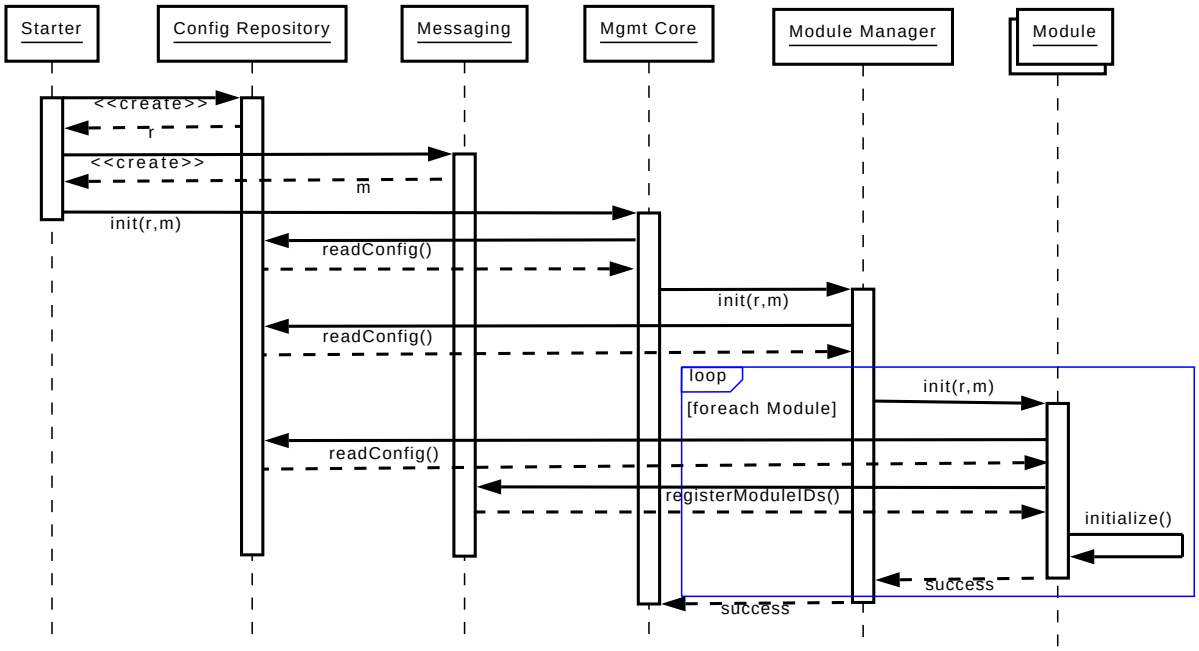

Abbildung 7.2: Start des Frameworks und Initialisierung der Module

Abbildung 7.2 beschreibt die Vorgänge beim Start des Frameworks. Zunächst erzeugt eine *Starter*-Komponente jeweils eine global nutzbare Instanz des Repositorys und des zentralen Messaging Systems. Im Anschluss wird der eigentliche Management-Kern als separater Thread erzeugt. Dieser liest nach seiner Initialisierung zunächst Konfigurationsinformationen aus dem Repository und initialisiert dann den Module Manager, der im Rahmen seiner Initialisierung die weiteren Module nachlädt. Die Liste der zu instanziierenden Erweiterungsmodule erhält der Module Manager aus dem Repository, hier sind auch spezifische Konfigurationsinformationen für einzelne Module abgelegt. Jedes Modul registriert sich zur Kommunikation mit anderen Modulen beim zentralen Messaging System.

7.1.1.3 Messaging System

Das Messaging System ermöglicht das Versenden typisierter Nachrichten an einzelne Module (Queue) oder an Gruppen von registrierten Empfängern (Topic). Das eingesetzte System

implementiert die durch das *Java Message Services* API [SUN03b] spezifizierte Funktionalität.

7.1.1.4 Klassifizierung der Erweiterungsmodule

Die eigentliche Management-Funktionalität wird durch das Zusammenspiel von Instanzen der einzelnen Erweiterungsmodule erbracht. Diese werden – entsprechend ihren Aufgaben – in mehrere Klassen unterteilt:

- Aktionsmodule (Action Modules)
- Ereignismodule (Event Modules)
- Kontrollmodule (Control Modules)
- Kommunikationsmodule (Communication Modules)
- Sonstige Module

Für die Anbindung der zu verwaltenden Komponente sind die in Abbildung 7.1 im unteren Bereich dargestellten *Aktionsmodule* und *Ereignismodule* zuständig. Aktions- und Ereignismodule unterscheiden sich in ihrem internen Aufbau und der Art der durch sie realisierten Kommunikation:

Aktionsmodule: Aktionsmodule können unterschiedliche Arten von Aufträgen ausführen. Typische Aufträge sind das Abfragen von Kenngrößen (Polling-Mechanismus) oder Eingriffe in das überwachte System. Aktionsmodule verhalten sich nach dem Start passiv und warten auf Aufträge anderer Module. Erhaltene Aufträge werden seriell durch ein Aktionsmodul abgearbeitet, wodurch sein interner Aufbau sehr einfach gehalten werden kann. Aktionsmodule kapseln die implementierten Monitoring- oder Management-Schnittstellen gegenüber den anderen Modulen der Architektur.

Ereignismodule: Im Unterschied zu Aktionsmodulen verhalten sich Ereignismodule nicht passiv. Sie können selbsttätig auf das Auftreten externer Ereignisse reagieren, indem sie Nachrichten an andere Erweiterungsmodule versenden (Push-Mechanismus). Ereignismodule überwachen das verwaltete System somit aktiv. Sie sind in der Lage, komplexe Aufträge auszuführen. Ereignismodule unterstützen beispielsweise die Benachrichtigung anderer Module bei Über- oder Unterschreiten bestimmter Schwellwerte. Sie können auftretende Ereignisse vor der Weiterleitung filtern und vorverarbeiten und so beispielsweise Statistiken über Messwerte erzeugen. Die Realisierung eines Ereignismoduls ist damit deutlich komplexer das die eines Aktionsmoduls. Analog zu Aktionsmodulen kapseln Ereignismodule die von ihnen implementierten Monitoring- und Management-Schnittstellen gegenüber anderen Modulen.

Jedes Aktions- oder Ereignismodul dient der Verwaltung einer bestimmten Klasse von Anwendungen oder Systemkomponenten, die über entsprechende Monitoring- oder Management-Schnittstellen verwaltet werden können. Zur parallelen Überwachung unterschiedlicher Anwendungen der gleichen Klasse können einzelne Erweiterungsmodule mehrfach instanziiert werden.

Kontrollmodule: Kontrollmodule implementieren Regelungsalgorithmen zum Selbst-Management einer überwachten Anwendung. Sie interagieren hierzu mit entsprechenden Aktions- und Ereignismodulen. Ein Kontrollmodul muss spezifisch an die zu verwaltende Anwendung angepasst werden. Je nach Problemstellung können unterschiedliche Arten von Algorithmen zur Kontrolle einer Anwendung eingesetzt werden. Kontrollmodule können zeit- oder ereignisgesteuert agieren oder eine Kombination von beidem realisieren. Innerhalb eines QoS-Managers realisiert ein Kontrollmodul im Zusammenspiel mit Aktions- und Ereignismodulen den in Abbildung 5.6 dargestellten Selfmanagement-Controller.

Kommunikationsmodule: Kommunikationsmodule dienen zum Informationsaustausch mit anderen QoS-Managern. Ähnlich wie Ereignismodule können sie aktiv auf externe Ereignisse reagieren und beispielsweise Nachrichten an andere Module versenden. Im Gegensatz zu Ereignismodulen implementieren sie aber keine Monitoring- oder Management-Schnittstellen sondern ein Kommunikationsprotokoll. Kommunikationsmodule kapseln das implementierte Kommunikationsprotokoll gegenüber anderen Modulen.

Sonstige Module: Das Framework beschränkt die Implementierung nicht auf die oben genannten Modultypen. Die Kategorisierung in die vorgestellten Klassen von Erweiterungsmodulen dient der Modularisierung und damit der vereinfachten Anpassung für das Selbst-Management unterschiedlicher Anwendungsklassen.

Zur Einbettung zusätzlicher Funktionalität (z. B. die in Abbildung 5.6 dargestellten logischen Komponenten zur Anbindung von WS-Agreement und zur Dienstüberwachung) werden in der Architektur weitere Module registriert. Auch die Realisierung der unterschiedlichen Repositories erfolgt durch Kapselung der Funktionalität in einer Modulschnittstelle.

Tabelle 7.1 gibt eine Übersicht über grundlegende Aktions- und Ereignismodule des Selfmanager-Frameworks, die jeweils zur Kommunikation mit einer Klasse von Anwendungen genutzt werden können, die entsprechende Monitoring- oder Management-Schnittstellen zur Verfügung stellt.

Zusammen mit einem Kontrollmodul können Ereignis- und Aktionsmodule flexibel kombiniert werden, um entsprechend den in Abschnitt 3.3 beschriebenen Modellen einen Selbst-

GRUNDLEGENDE AKTIONS- UND EREIGNISMODULE

`CIM/WBEM-Module`	Abfrage von CIM-Instanzen, Lesen und Schreiben von Attributen, Aufruf von Methoden an CIM-Klassen
`ARM-Module`	Abfrage von Messpunkten, Messwerten und grundlegenden statistischen Kenngrößen
`JMX-Module`	Abfrage von MBean-Instanzen, Lesen und Schreiben von Attributen, Aufruf von MBean-Methoden
`SNMPModule`	Kommunikation mit SNMP-Agenten, Empfangen und Versenden von SNMP Traps
`WSDMModule`	Kommunikation mit durch WSDM überwachten Web Services
`CommandModule`	Ausführen von lokalen Programmen und Shell-Skripten
`LogParserModule`	Einlesen und Analysieren von Log-Dateien
`LoggerModule`	Erzeugen von Log-Dateien

Tabelle 7.1: Module zur Anbindung von Sensorik und Aktorik über Standard-Schnittstellen

Management-Regelkreis zu etablieren. Das zur Kontrolle des Systems notwendige Management-Wissen wird dabei in einem von der Art des Kontrollmoduls abhängigen Format abgelegt.

7.1.2 Anbindung der WS-Agreement-Schnittstelle

Die Integration der Client- und Server-seitigen WS-Agreement-Funktionalität in die prototypische QoS-Manager-Realisierung basiert auf Version 0.0.6 der WSAG4J-Bibliothek[1] des Fraunhofer-Instituts für Algorithmen und Wissenschaftliches Rechnen (SCAI). Im Folgenden wird zunächst auf die Integration der WS-Agreement-Server-Schnittstelle eingegangen, im Anschluss wird die deutlich einfachere Client-seitige Einbindung von WS-Agreement vorgestellt. Beide Schnittstellen werden durch ein Selfmanager-Modul gekapselt. Basis für die Codierung von Dienstgüteanforderungen in der Implementierung ist ein einfaches XML-Schema, das im Anschluss diskutiert wird.

7.1.2.1 Die WS-Agreement-Server-Schnittstelle

Abbildung 7.3 zeigt den Aufbau des WS-Agreement-Adapters zur Anbindung der SLA-Aushandlungsfunktionalität an einen QoS-Manager. Zur Bekanntgabe der für den Abschluss von

[1] Details unter `http://packcs-e0.scai.fhg.de/mss-project/wsag4j/`

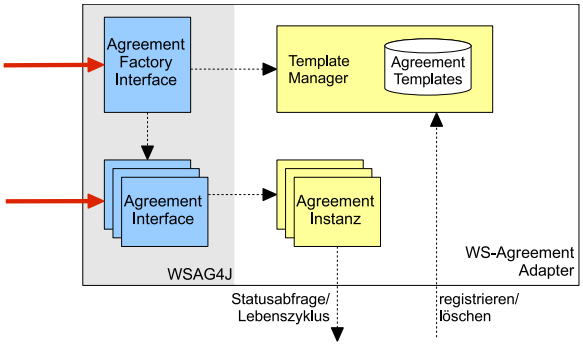

Abbildung 7.3: Aufbau des WS-Agreement-Adapters

SLAs akzeptablen Rahmenbedingungen kann der QoS-Manager eine Anzahl von SLA Templates beim Template Manager des WS-Agreement-Adapters registrieren. Für jedes SLA Template können intern ein Gültigkeitszeitraum und die maximal mögliche Anzahl gleichzeitig existierender und auf diesem Template basierender SLAs festgelegt werden. Weiterhin wird eine Callback-Schnittstelle registriert, über die der WS-Agreement-Adapter den QoS-Manager benachrichtigen kann, falls auf Basis eines registrierten Templates ein SLA abgeschlossen wird.

WSAG4J stellt eine Agreement-Factory-Schnittstelle bereit, über die ein Initiator zunächst – wie in Abschnitt 3.2.3.1 beschrieben – eine Liste der durch den Responder bereitgestellten SLA Templates anfordern kann. Nach Eingang einer entsprechenden SOAP-Anfrage konstruiert der Template Manager eine Liste aktuell gültiger SLA Templates und gibt diese an die WSAG4J Agreement Factory zurück, die diese dann an den Initiator weiterleitet.

Der Initiator kann nun entweder auf Basis eines der Templates oder ohne Referenz eines bestimmten Templates ein SLA-Offer-Dokument erstellen und an die Agreement Factory richten. Hier wird das Dokument auf Konformität mit dem zugrunde liegenden Template überprüft und – falls die Prüfung positiv verläuft – eine neue Agreement-Instanz erzeugt. Wurde kein Template angegeben, wird geprüft, ob das SLA-Offer-Dokument sich inhaltlich mit einem der Templates deckt. Ist dies der Fall, wird es dem Template zugeordnet und eine entsprechende Agreement-Instanz erzeugt, andernfalls wird das Angebot abgelehnt. Im Erfolgsfall wird dem SLA Initiator durch die WSAG4J-Bibliothek eine Referenz auf die Web-Services-Schnittstelle des neu erzeugten Agreements zurückgeliefert.

Beim Erzeugen einer Agreement-Instanz wird der QoS-Manager mithilfe der dem zugrunde liegenden Template zugeordneten Callback-Schnittstelle benachrichtigt.

WS-Agreement stellt dem Initiator über die Agreement-Schnittstelle unterschiedliche Methoden zur Verwaltung des Lebenszykluses eines SLAs und zur Statusabfrage zur Verfügung. Die zur Beantwortung derartiger Anfragen notwendigen Informationen werden ebenfalls mithilfe der Callback-Schnittstelle direkt beim QoS-Manager erhoben.

7.1.2.2 Die WS-Agreement-Client-Schnittstelle

Die WSAG4J-Bibliothek stellt auch Client-seitig eine Abstraktion der in der WS-Agreement-Spezifikation definierten Aushandlungsfunktionalität bereit. Zur Anbindung der Client-Schnittstelle an einen QoS-Manager wird diese Abstraktion lediglich um einige Hilfsklassen zur Verarbeitung von SLA-Inhalten und Verwaltung von SLAs und deren Lebenszyklus erweitert.

7.1.2.3 Codierung von Dienstgüteanforderungen in SLA-Dokumenten

Da – wie in Abschnitt 3.2.3.1 beschrieben – die WS-Agreement-Spezifikation selbst keine Definitionen bezüglich des SLA-Inhalts trifft, müssen die WS-Agreement-Dokumente zur Codierung von Dienstgüteanforderungen erweitert werden. Prinzipiell können hierbei beliebige XML-basierte Spezifikationen zum Einsatz kommen. Beispielhaft sei das von IBM entwickelte WSLA [KL03] genannt; eine vollständige Implementierung von WSLA gestaltet sich aufgrund des Umfangs der Spezifikation allerdings sehr komplex. Daher wurde für die prototypische Umsetzung des in dieser Arbeit entwickelten Dienstgüte-Management-Systems ein minimalistisches, zur Definition von SLA-Inhalten bezüglich Antwortzeit und Durchsatz geeignetes XML-Schema entwickelt. Das Schema orientiert sich lose an einer in [Ung05] beschriebenen Erweiterung der WS-Agreement-Spezifikation. Es ist in Quellcode B.1 in Anhang B dokumentiert.

Beispiel 7.1: Quellcode 7.1 zeigt die Einbindung von Dienstgüteanforderungen in ein SLA-Dokument. Im Beispiel werden ein Durchsatz von 5 bis 10 Aufrufen pro Sekunde und eine Antwortzeit von 100 ms garantiert.

```
...
<ServiceLevelObjective>
  <CustomServiceLevel>
    <qos xmlns="http://vs.cs.fh-wiesbaden.de/agreement/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance" xsi:type="and">
      <qosElement max="10" min="5" unit="s" variable="
         throughput" xsi:type="throughput"/>
      <qosElement max="100" unit="ms" variable="
         responseTime" xsi:type="responseTime"/>
    </qos>
  </CustomServiceLevel>
</ServiceLevelObjective>
...
```

Quellcode 7.1: Beispiel zur Nutzung des QoS-Schemas

Basierend auf dem XML-Schema wurden mithilfe der Java Architecture for XML Binding[2] (JAXB) Hilfsklassen zum Umgang mit Dienstgüteparametern erzeugt, die in der Implementierung zum Zugriff auf SLA-Inhalte verwendet werden.

7.1.3 QoS-Manager für Workflows

Im Folgenden wird – basierend auf der in Abschnitt 7.1.1 vorgestellten Selfmanager-Architektur – die Realisierung eines QoS-Managers für die quelloffene Workflow Engine *Apache Orchestration Director Engine*[3] (ODE) in Version 1.2 vorgestellt.

7.1.3.1 Apache ODE

ODE selbst ist in Java implementiert und unterstützt sowohl BPEL 1.1 als auch WS-BPEL 2.0 konforme Prozesse. Zur Kommunikation mit durch einen WS-BPEL-Prozess referenzierten Web Services nutzt ODE in der Regel die Apache Web-Services-Bibliothek Axis2[4], alternativ ist die Integration anderer sog. Transportprotokolle über eine einheitliche Schnittstelle möglich. Integraler Bestandteil der ODE Engine ist ein BPEL Compiler, der WS-BPEL-Prozesse in ein intern genutztes Binärformat überführt.

ODE bietet zwei Web Services Schnittstellen für das Monitoring der ODE Runtime an: Das *Process Management Interface* liefert Informationen über zur Verfügung stehende WS-BPEL-Prozesse, d. h. Prozessbeschreibungen, die der ODE Engine im Rahmen des Deployment-Prozesses bekannt gemacht wurden und deren Compilierung keine syntaktischen Fehler ergab. Das *Process Instance Management Interface* ermöglicht die Abfrage von aktiven oder bereits beendeten WS-BPEL-Prozess-Instanzen. Weiterhin liefert ODE über diese Schnittstelle Details zur Ausführung einzelner BPEL-Aktivitäten als Liste sog. Events. Mithilfe dieser Events kann eine sehr feingranulare Überwachung einzelner Prozessinstanzen erfolgen.

7.1.3.2 Architekturüberblick

Abbildung 7.4 zeigt die Modul-Architektur des QoS-Managers für Apache-ODE-basierte WS-BPEL-Prozesse. Die Architektur des QoS-Managers spiegelt die in Abbildung 5.4 gezeigte logische Struktur des Workflow-Managers wider. Die Abbildung repräsentiert damit die konkrete Konfiguration des in Abschnitt 7.1.1 vorgestellten Selfmanager Frameworks zur Realisierung des QoS-Managers für ODE.

[2] Details unter `https://jaxb.dev.java.net/`
[3] Details unter `http://ode.apache.org/index.html`
[4] Details unter `http://ws.apache.org/axis2/`

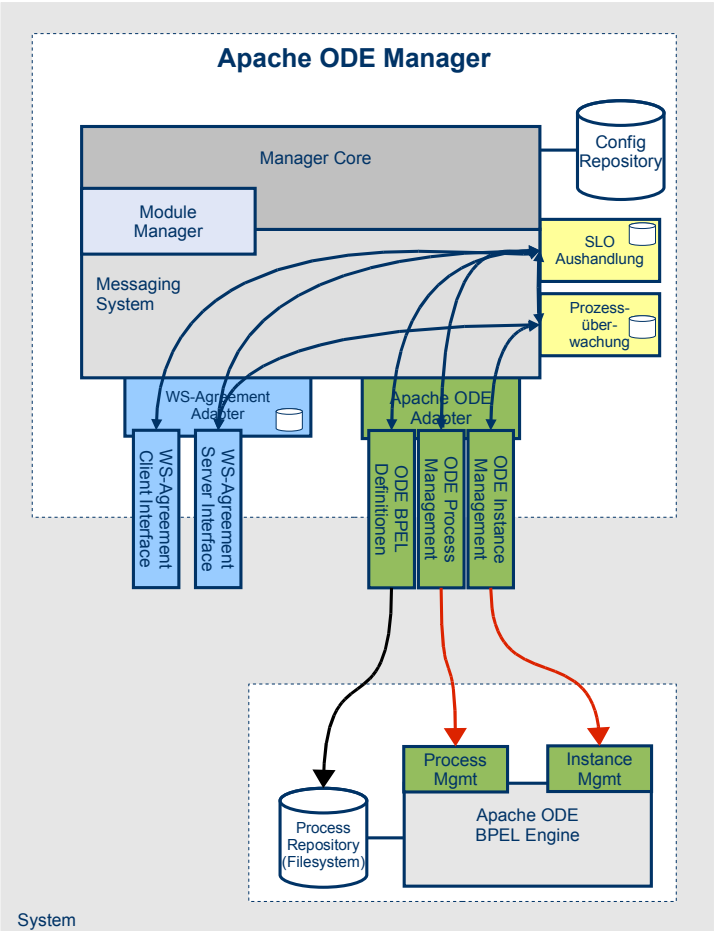

Abbildung 7.4: Modul-Architektur des QoS-Managers für Apache ODE

Die Anbindung von Workflow-Engine und WS-Agreement-Schnittstellen erfolgt über Selfmanager-Ereignismodule. Der Apache-ODE-Adapter stellt eine Schnittstelle zum Zugriff auf die weiter oben beschriebenen ODE-Schnittstellen für Process Management und Process Instance Management bereit, die Kommunikation mit der Workflow Engine erfolgt dabei mithilfe von Web Services. Das Ereignismodul fragt die ODE-Management-Schnittstellen regelmäßig ab und benachrichtigt dann aktiv andere Module über relevante Zustandsänderungen innerhalb der Workflow Engine.

Weiterhin beinhaltet der ODE-Adapter des QoS-Managers ein Modul, das das ODE Process Repository selbsttätig bezüglich des Deployments neuer WS-BPEL-Prozesse bzw. des Entfernens nicht mehr benötigter WS-BPEL-Prozesse überwacht. Das Modul liest die XML-Prozessbeschreibung für neu hinzugefügte WS-BPEL-Prozesse und stellt diese anderen Modulen zur Verfügung.

Die dargestellte Konfiguration enthält zwei Kontrollmodule, auf die im Folgenden genauer

eingegangen wird. Eine erste Version der Implementierung des QoS-Managers für Apache ODE ist detailliert in [Mik08] beschrieben.

7.1.3.3 SLO-Aushandlung und Prozessregistrierung

Das für die *SLO-Aushandlung* verantwortliche Modul wird beim Empfang eines SLA-Offer-Dokuments für einen in ODE verfügbaren WS-BPEL-Prozess aktiv. Es analysiert den WS-BPEL-Prozess und handelt basierend auf dem vorgegebenen Gesamt-SLO einzelne SLOs für die am Prozess beteiligten Dienste aus. Im Anschluss wird der Prozess mit seinem SLO in das Repository der zu überwachenden Prozesse aufgenommen. Die Analyse der Struktur von WS-BPEL-Prozessen erfolgt in einem Modul, dessen Implementierung auf dem Apache ODE BPEL Compiler basiert. Mithilfe dieses Moduls lassen sich WS-BPEL-Prozesse auf syntaktische Korrektheit prüfen und in die in Abschnitt 6.4.2 beschriebene, optimierte Darstellung überführen. Gleichzeitig löst das Modul im Prozess symbolisch durch WS-BPEL `PartnerLinks` repräsentierte Verweise auf beteiligte Dienste auf und schafft so die Grundlage für die Kontaktaufnahme mit QoS-Managern der referenzierten Dienste zur Aushandlung von individuellen SLOs.

Abbildung 7.5 zeigt die Abläufe innerhalb des Kontrollmoduls zur SLO-Aushandlung. Im Aktivitätsdiagramm ist die Kommunikation mit anderen Modulen jeweils als Nachricht eingezeichnet. Der hervorgehobene Prozessschritt umfasst den in Abschnitt 6.7 beschriebenen Algorithmus zur initialen Zerlegung des Gesamt-SLOs.

7.1.3.4 Prozessüberwachung

Das zweite Kontrollmodul dient der *Prozessüberwachung* aktiver WS-BPEL-Prozess-Instanzen. Dieses Kontrollmodul ist permanent aktiv. Es überprüft regelmäßig, ob eine WS-BPEL-Prozess-Instanz das für den Prozess vereinbarte SLO verletzt hat. Wird eine SLO-Verletzung festgestellt, informiert das Prozessüberwachungsmodul den Vertragspartner.

Abbildung 7.6 zeigt schematisch die internen Abläufe des Kontrollmoduls zur Prozessüberwachung. Das Kontrollmodul erhält immer dann eine Nachricht vom ODE-Adapter, wenn dort eine relevante Statusänderung festgestellt wird. Beim Empfang einer Statusnachricht wird zunächst untersucht, ob diese einem bekannten WS-BPEL-Prozess zugeordnet werden kann, für den bereits im Rahmen der in Abschnitt 7.1.3.3 beschriebenen SLA-Aushandlung ein Antwortzeit-SLO vereinbart wurde.

Kann das überlieferte Ereignis mit einer Instanz eines WS-BPEL-Prozesses in Verbindung gebracht werden, für den ein SLO vereinbart wurde, wird geprüft, ob sich die Ereignismeldung auf die Ausführung einer Aktivität oder den Lebenszyklus einer Prozessinstanz bezieht.

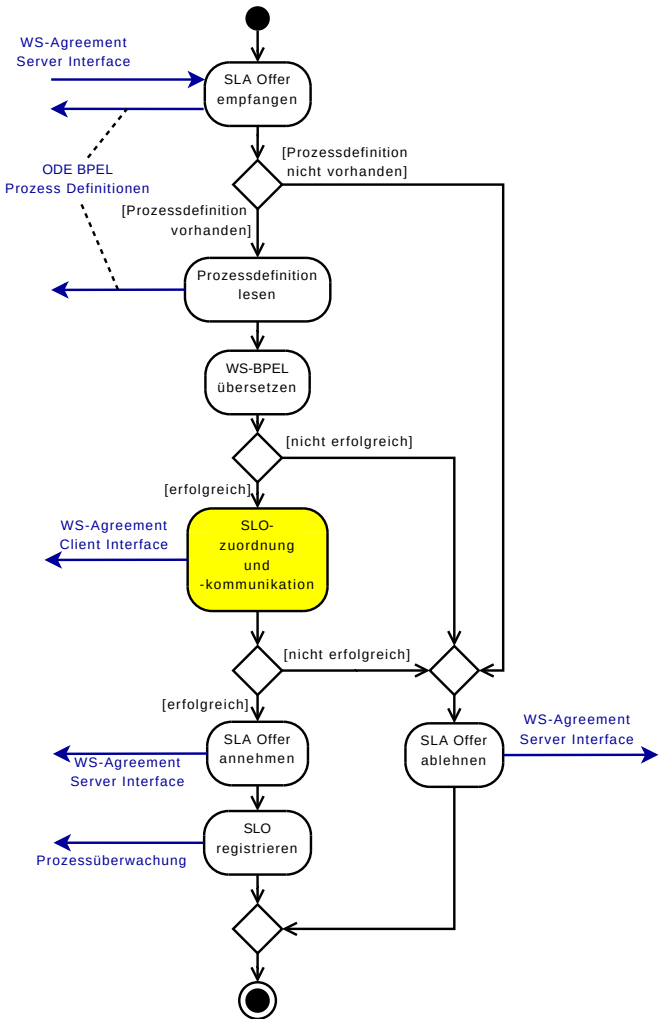

Abbildung 7.5: Modulinterne Abläufe bei der SLO-Aushandlung und Prozessregistrierung

Aktivitäten betreffende Ereignisse werden gefiltert, sodass nur Dienstaufrufe und von zuvor aufgerufenen Diensten erhaltene Antworten betrachtet werden müssen. Aus zueinander gehörenden Aufrufen und Antworten wird die Antwortzeit eines Dienstes extrahiert und als Grundlage für mögliche zukünftige SLO-Verhandlungen abgelegt.

Statusänderungen von Prozessinstanzen kündigen i. d. R. entweder den Start einer neuen Instanz oder das Lebensende einer bisher aktiven Instanz an. Wurde eine neue Instanz gestartet, wird diese in den Pool überwachter Prozessinstanzen aufgenommen. Terminiert eine Instanz regulär, kann mit der Differenz der Start- und Terminierungszeitstempel die Antwortzeit für die Prozessinstanz berechnet werden. Ergibt die Antwortzeitberechnung, dass eine SLO-Verletzung aufgetreten ist, wird der Vertragspartner benachrichtigt. Im Anschluss wird die Prozessinstanz aus der Liste der überwachten Instanzen entfernt.

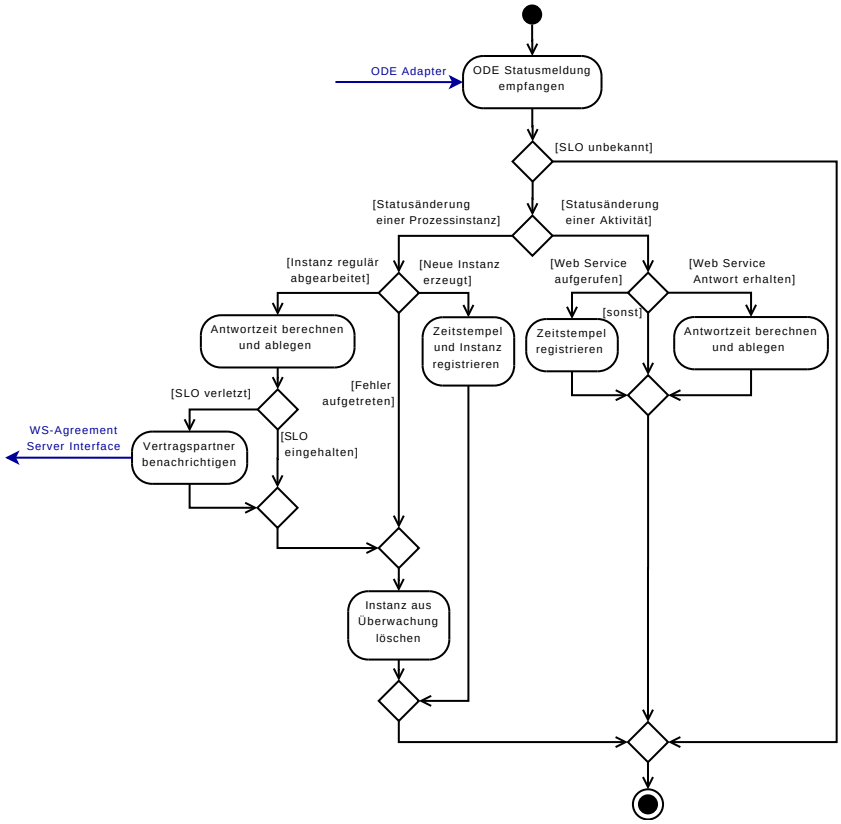

Abbildung 7.6: Modulinterne Abläufe bei der Prozessüberwachung

7.1.3.5 Fazit

Abschnitt 7.1.3 zeigt am Beispiel eines QoS-Managers für Apache ODE, wie die Anbindung eines QoS-Managers an eine Workflow Engine realisiert werden kann. Da für diese kein Managementstandard existiert, ist die Art der Anbindung derzeit davon abhängig, welche Schnittstellen durch die Workflow Engine bereitgestellt werden. Die in Abschnitt 4.1.2 zusammengefassten Ansätze zur Instrumentierung von Workflow Engines zeigen aber, dass andere Plattformen über vergleichbare Schnittstellen verfügen und der Implementierungsansatz damit mit relativ geringem Aufwand auf andere Umgebungen übertragen werden kann.

7.1.4 QoS-Manager für Dienste

Im Folgenden wird ein Beispiel für die Implementierung eines QoS-Managers für SOA-Dienste diskutiert. Wie in Abschnitt 5.3.1 beschrieben, müssen sowohl die Anbindung an bereitgestellte Monitoring- und Management-Schnittstellen als auch der verwendete Kontrollalgorithmus an die Implementierung des zu verwaltenden Dienstes angepasst werden, sodass an dieser

Stelle keine allgemein gültige Implementierung präsentiert werden kann. Stattdessen wird hier beispielhaft das QoS-Management von Diensten auf Basis des Applikationsservers JBoss vorgestellt. Anhand des Beispiels kann die allgemeine Vorgehensweise zur Realisierung eines QoS-Managers für bestimmte Anwendungsklassen abgeleitet werden.

7.1.4.1 Szenario: Management von JBoss-Instanzen

JBoss[5] ist ein quelloffener JEE-konformer Applikationsserver, der aufgrund seiner Stabilität und Skalierbarkeit auch in geschäftskritischen Umgebungen eingesetzt wird. Neben klassischen EJB- und Servlet-Anwendungen unterstützt der Applikationsserver auch den Zugriff über Web-Service-Schnittstellen und eignet sich somit für das Bereitstellen von Diensten in einer Web-Service-basierten SOA.

JBoss bietet einen Clustering-Mechanismus, der es erlaubt, Instanzen einer Anwendung auf mehrere physikalische Maschinen zu verteilen und so einen dynamischen Lastausgleich zu realisieren. Auf diese Weise kooperierende JBoss-Instanzen bezeichnet man als JBoss-Cluster. Zur Laufzeit können einem Cluster weitere Maschinen zugeführt oder auch entzogen werden. Kunden richten ihre Anfragen stets an einen Load Balancer, der dann die eingehenden Aufträge einzelnen Knoten im Cluster zuordnet.

Zur Minimierung des Ressourcenverbrauchs einer auf Basis von JBoss implementierten Anwendung ist es wünschenswert, stets nur so viele Applikationsserver-Instanzen bereitzustellen, wie zur Abdeckung der aktuell vorliegenden Last erforderlich sind. So können sich u. U. Anwendungen mit unterschiedlichem Lastprofil einen Pool von Maschinen teilen.

Ziel des hier vorgestellten QoS-Managers für JBoss-basierte Anwendungen ist die Realisierung eines solchen dynamischen Clustering-Ansatzes, der in der Lage ist, vorgegebene Antwortzeitanforderungen für unterschiedliche Lastprofile zu erfüllen, und gleichzeitig bestrebt ist, die Ressourcennutzung zu minimieren. Abbildung 7.7 gibt einen Überblick über die realisierte Architektur.

Anfragen an die Anwendungen werden grundsätzlich an einen Load Balancer gerichtet. Dieser verteilt die Arbeitsaufträge an eine Anzahl aktiver JBoss-Instanzen. Der QoS-Manager misst die Antwortzeiten der einzelnen Instanzen und entscheidet auf Basis der erhobenen Messwerte, ob zur Einhaltung der Antwortzeitvorgabe zusätzliche Instanzen gestartet werden müssen oder die die Anzahl aktiver Instanzen reduziert werden kann.

[5] Details unter `http://www.jboss.com`

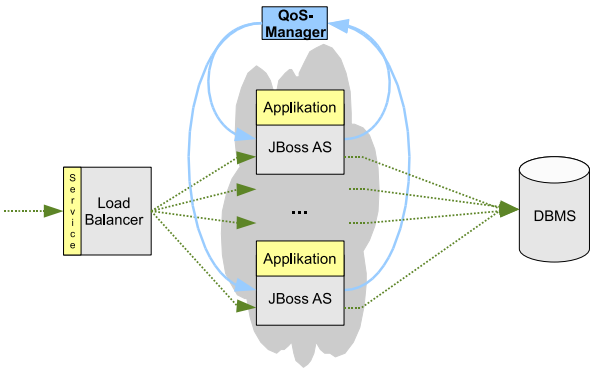

Abbildung 7.7: Dynamisches Clustering von JBoss-Servern

7.1.4.2 Architekturüberblick

Abbildung 7.8 zeigt die Modularchitektur des QoS-Managers für JBoss-basierte SOA-Dienste und die Anbindung von Sensorik und Aktorik an die verwalteten JBoss-Instanzen. Die Aushandlung von SLAs erfolgt über den in Abschnitt 7.1.2 beschriebenen WS-Agreement-Adapter.

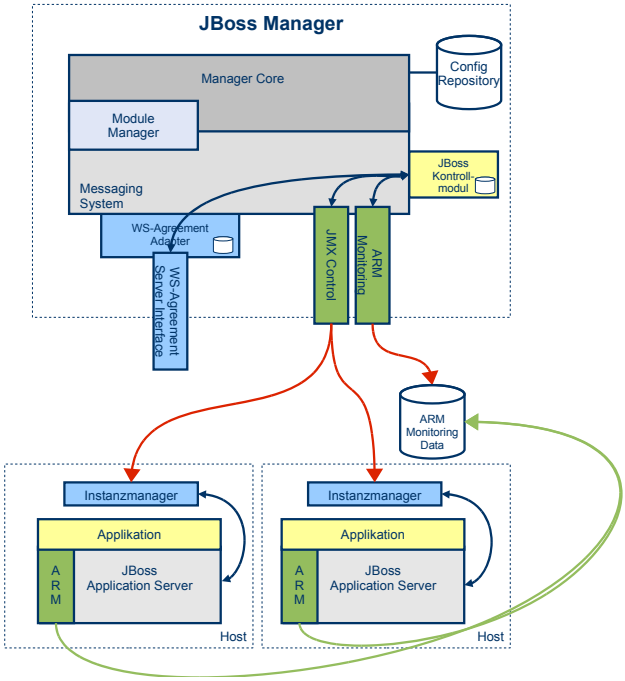

Abbildung 7.8: Modul-Architektur des QoS-Managers für den JBoss Applikationsserver

In den folgenden Abschnitten wird näher auf das Zusammenspiel der zum Management eingesetzten QoS-Manager-Module und die implementierten Kontrollalgorithmen eingegangen.

7.1.4.3 Erfassung von Antwortzeiten

Um die Antwortzeiten beliebiger Anwendungen auf Basis von JBoss transparent vermessen zu können, wurde im JBoss-Server selbst auf Request-Ebene eine ARM-Instrumentierung (vgl. Abschnitt 3.2.3.2) vorgenommen. Die ARM-Instrumentierung wird – wie in Abschnitt 4.1.1 beschrieben – mithilfe von Standard-Interceptor-Schnittstellen in den Applikationsserver integriert, sodass weder an der Anwendung noch am Applikationsserver selbst Code-Modifikationen vorgenommen werden müssen. Die Instrumentierung des JBoss-Applikationsservers ist detailliert in [STTK07] und [Ter06] beschrieben.

Die ARM-Instrumentierung der JBoss-Server legt die gemessenen Antwortzeiten für zuvor ausgewählte Arten von Applikationsaufrufen (z. B. den Aufruf einer bestimmten EJB-Methode) zentral in einer ARM-Datenbank ab. Auf diese Datenbank greift der QoS-Manager, wie in Abbildung 7.8 skizziert, über ein ARM-Ereignismodul zu.

7.1.4.4 Lebenszyklusverwaltung von JBoss-Instanzen

Zum Starten und Stoppen einzelner JBoss-Instanzen wird eine separate Anwendung – der für diesen Zweck entwickelte *Instanzmanager* – verwendet. Jeder Instanzmanager ist fest einer bestimmten Installation des JBoss-Servers zugeordnet, d. h. auf jeder Maschine, auf der potenziell eine JBoss-Instanz gestartet werden kann, befindet sich ein eigener Instanzmanager.

QoS-Manager und Instanzmanager kommunizieren über JMX [Sun08]: Der QoS-Manager kann einen Instanzmanager über eine Nachricht auffordern, den ihm zugeordneten JBoss-Server zu starten oder zu stoppen. Jeder Instanzmanager implementiert einen einfachen Zustandsautomaten, der in Abbildung 7.9 dargestellt ist. Der aktuelle Zustand des Instanzmanagers kann ebenfalls über die bereitgestellte JMX-Schnittstelle abgefragt werden.

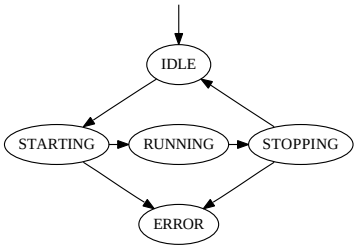

Abbildung 7.9: Zustandsgraph des Instanzmanagers

Ist die dem Instanzmanager zugeordnete JBoss-Instanz nicht gestartet, befindet sich der Manager im Zustand `IDLE`. Erhält er nun vom QoS-Manager den Auftrag, die JBoss-Instanz

zu starten, wechselt er in den Zustand `STARTING`. In diesem Zustand überprüft er die Verfügbarkeit der JBoss-Instanz durch regelmäßiges Abfragen der JBoss-eigenen JMX-Management-Konsole. Nach erfolgreichem Abschluss des Startvorgangs wechselt er in den Zustand `RUNNING`. Tritt beim Start ein Fehler auf oder startet der JBoss-Server nicht innerhalb einer zuvor definierten Zeitspanne, wechselt er in den Zustand `ERROR`. Erhält der Instanzmanager vom QoS-Manager die Aufforderung zum Stoppen des JBoss-Servers, wechselt er in den Zustand `STOPPING` und initiiert über die JBoss-JMX-Management-Konsole das geordnete Herunterfahren der Instanz. Bei Erfolg wechselt der Manager in den Zustand `IDLE`, tritt ein Fehler auf, wechselt er in den Zustand `ERROR`.

7.1.4.5 Implementierter Kontrollalgorithmus

Der QoS-Manager überwacht den Zustand der einzelnen Instanzmanager und behält so den Überblick über die Anzahl der aktuell gestarteten JBoss-Instanzen.

Das Kontrollmodul des QoS-Managers implementiert einen Zustandsautomaten, der bei Bedarf – basierend auf der ermittelten Antwortzeit RT – veranlasst, dass zusätzliche JBoss-Instanzen gestartet oder ungenutzte Instanzen beendet werden. RT repräsentiert dabei einen gleitenden Mittelwert über die ermittelten Antwortzeiten der einzelnen Instanzen.

Abbildung 7.10 zeigt den Automaten mit den Zuständen `Operating`, `HeavyLoad` und `Overload`.

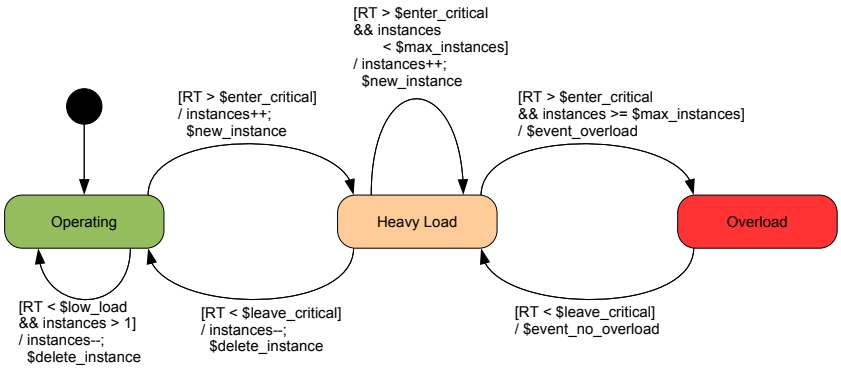

Abbildung 7.10: Automatendarstellung des Reglers

Die maximale Anzahl von JBoss-Instanzen *$max_instances* ergibt sich aus der Anzahl der Instanzmanager, die sich nicht im Zustand `ERROR` befinden. Die Schwellwerte *$low_load*, *$enter_critical* und *$leave_critical* werden in Abhängigkeit vom definierten Antwortzeit-SLO und der Charakteristik der verwalteten Anwendung gesetzt. Abbildung 7.11 gibt ein Beispiel für die Festlegung der Schwellwerte, das sich an der in Abschnitt 6.3 besprochenen Vorgehensweise orientiert.

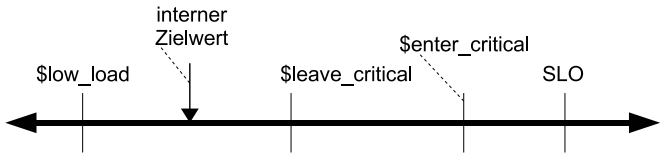

Abbildung 7.11: Beispielhafte Festlegung der Grenzwerte

Zu Beginn befindet sich der QoS-Manager im Zustand `Operating`, und mindestens eine JBoss-Instanz ist aktiv. Wird der Antwortzeit-Schwellwert *$enter_critical* überschritten, wechselt der QoS-Manager in den Zustand `HeavyLoad`, gleichzeitig wird eine weitere JBoss-Instanz gestartet. Solange sich der entsprechende Instanzmanager im Zustand `STARTING` befindet, wird keine weitere Analyse der Antwortzeit vorgenommen: Nach dem Wechsel in den `RUNNING`-Zustand wird zunächst die Historie von RT verworfen, um eine Überreaktion des Systems durch unnötigen Start weiterer Instanzen zu vermeiden. Bei Bedarf werden im Zustand `HeavyLoad` weitere Instanzen gestartet, sinkt RT unter den Schwellwert *$leave_critical*, so wird eine Instanz gestoppt, und der QoS-Manager wechselt zurück in den Zustand `Operating`. Können im `HeavyLoad`-Zustand keine weiteren Instanzen gestartet werden, so wechselt der QoS-Manager in den Zustand `Overload`. Hierbei wird intern ein *$event_overload*-Ereignis ausgelöst. Als Reaktion hierauf kann beispielsweise ein Administrator informiert werden, alternativ kann aber auch der Versuch einer Lockerung des definierten SLOs über den in Abschnitt 6.4 vorgestellten Kooperationsmechanismus erfolgen. Durch eine SLO-Lockerung ergeben sich neue Schwellwerte für den Zustandsautomaten, sodass dieser ggf. wieder in den Zustand `HeavyLoad` und später `Operating` zurückkehren kann.

Beim Zustandswechsel von `Overload` nach `HeavyLoad` wird das interne Ereignis *$event_no_overload* generiert. Als Reaktion hierauf kann beispielsweise erneut eine Benachrichtigung des Administrators erfolgen oder eine zuvor initiierte Auktion abgebrochen werden.

7.1.4.6 Fazit

Die gezeigte Konfiguration eines QoS-Managers für JBoss-basierte Anwendungen zeigt, wie ein SOA-Dienst mithilfe von Selbst-Management-Mechanismen in die dezentral organisierte Dienstgüte-Management-Umgebung eingegliedert werden kann. Der vorgestellte Zustandsautomat ist ein einfaches Beispiel für einen Selbst-Management-Regelungsalgorithmus. Gleichzeitig wird jedoch erkennbar, dass der jeweilige Regelungsalgorithmus eng auf die Charakteristik der zu verwaltenden Anwendung abgestimmt werden muss. Der hier präsentierte Ansatz ist primär für rechenintensive Anwendungen geeignet, da er auf der Annahme basiert, dass sich die mittlere Antwortzeit des Systems mit zunehmender Last erhöht.

Andere Kategorien von Anwendungen verhalten sich unter Last ggf. anders: Oftmals wird in

der Praxis beispielsweise beobachtet, dass bei zunehmender Systemlast zunächst die Varianz der Antwortzeit von Anfragen durch eine Häufung von Ausreißern steigt, die Antwortzeit der überwiegenden Mehrzahl von Anfragen aber noch unverändert bleibt. Für solche Anwendungen muss ein angepasster Regelungsalgorithmus definiert werden.

7.1.5 Die QoS-Proxy-Komponente

Der in Abschnitt 5.4.3.3 vorgestellte Ansatz zur parallelen Durchsetzung unterschiedlicher Dienstgüteanforderungen wurde prototypisch für die quelloffene SCA Runtime *Apache Tuscany*[6] in Form einer Policy-Erweiterung implementiert.

Apache Tuscany ist als erweiterbares Framework konzipiert: Es besteht aus einer *Core*-Komponente, die in Applikationsserver und andere Laufzeitumgebungen eingebettet werden kann. Über Erweiterungsschnittstellen des Tuscany Cores werden SCA Implementations, Protocol Bindings und Policies angebunden. Zur Laufzeit werden durch eine Komponente referenzierte Dienste mithilfe von registrierten Protocol-Bindings über sog. *Runtime Wires* verbunden. Ein Runtime Wire besteht aus einer Verkettung von Adaptern und Interzeptoren (*Interceptors*), die Verarbeitung und Weiterleitung von Aufrufen sicherstellen. Tuscany Interceptors können durch Konfiguration von SCA Policies in die Aufrufkette eingebracht werden und beispielsweise zur transparenten Beobachtung, Vermessung oder Verschlüsselung/Entschlüsselung einzelner Aufrufe genutzt werden. SCA Policies können für einzelne Services oder auch SCA Composites spezifiziert werden.

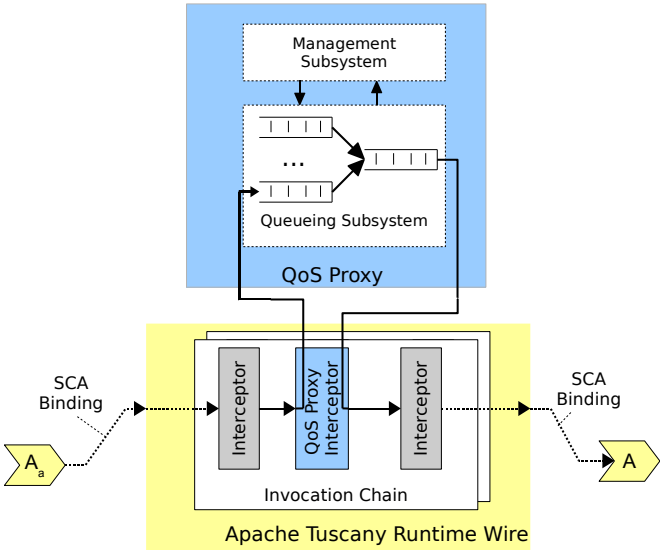

Abbildung 7.12: Einbindung der QoS-Proxy-Funktionalität in Apache Tuscany

[6] Details unter `http://tuscany.apache.org`

Abbildung 7.12 zeigt die Einbindung der QoS-Proxy-Funktionalität in die Apache Tuscany SCA Runtime. Der durch den QoS-Proxy in unterschiedlichen Dienstgüteklassen zu offerierende Service wird als separate Schnittstelle pro Dienstgüteklasse in einem SCA-Composite deklariert. Für jede deklarierte Schnittstelle wird eine SCA Policy spezifiziert, auf deren Basis zur Laufzeit ein QoS-Proxy-Interceptor instanziiert wird. Der Interceptor reiht empfangene Anfragen in eine durch die Proxy-Implementierung bereitgestellte dienstgüteklassenspezifische Warteschlange ein.

Der QoS-Proxy selbst besteht aus zwei wesentlichen Teilsystemen: Das Management-Teilsystem offeriert eine JMX-Schnittstelle, über die ein QoS-Manager Anforderungen an die bereitgestellten Dienstgüteklassen konfigurieren kann. Diese Anforderungen werden dann auf eine konkrete Konfiguration des Proxys abgebildet. Das Queueing-Teilsystem dient der Durchsetzung der konfigurierten Dienstgüte.

Zur Spezifikation der Dienstgüteklassen wird eine flexible Kombination aus Weighted Fair Queueing (WFQ) [DKS89] und dem Leaky-Bucket-Algorithmus (LB) [Tur02] eingesetzt, die sowohl erlaubt, die Gesamtzahl der Anfragen an das Zielsystem zu regulieren, als auch, die Durchsetzung von Prioritäten einzelner Klassen zu gewährleisten.

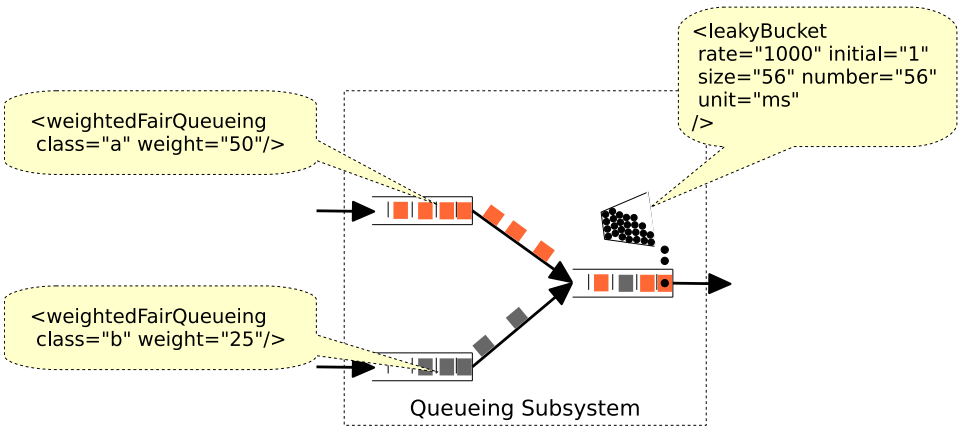

Abbildung 7.13: Beispielkonfiguration des Queueing-Subsystems zur Realisierung unterschiedlicher Dienstgüteklassen

Abbildung 7.13 zeigt eine Beispielkonfiguration mit zwei Dienstgüteklassen `a` und `b`. Mithilfe des WFQ-Algorithmuses werden in Klasse `a` eingeordnete Anfragen mit einem Gewicht von 50 gegenüber Anfragen der Klasse `b` mit einem Gewicht von 25 priorisiert. Im Beispiel ist zudem der Gesamtdurchsatz des Systems mithilfe von LB limitiert: Pro 1000ms werden 56 Tokens erzeugt, das Fassungsvermögen des Buckets beträgt ebenfalls 56 Tokens, sodass im Beispiel max. 56 Anfragen pro Sekunde an die referenzierte Implementierung weitergeleitet werden.

Eine detaillierte Beschreibung der Implementierung des QoS-Proxys findet sich in [Fre08].

7.2 Implementierung der Komponenten zur Selbstorganisation

Zur Integration der in Kapitel 6 entwickelten Verfahren zur dezentralen Koordination von QoS-Managern wurden ein QoS-Manager-Erweiterungsmodul und Koordinationsprotokolle entwickelt. Der Aufbau und die Funktionsweise des Moduls werden in Abschnitt 7.2.1 erläutert.

Als Grundlage für Untersuchungen zur Skalierbarkeit des entwickelten Ansatzes wurde ein Simulator implementiert, mit dem sich große Szenarien auf ihr Verhalten bezüglich der Koordination von QoS-Managern untersuchen lassen. Die Architektur des Simulators wird detailliert in Abschnitt 7.2.2 besprochen.

7.2.1 Koordination von QoS-Manager-Komponenten

Die in Abschnitt 6.4 beschriebenen Verfahren zur Übertragung von SLO-Anteilen wurde prototypisch auf Basis des JGroups-Gruppenkommunikations-Frameworks[7] realisiert. Die Implementierung basiert im Kern auf [Ste07].

Im Folgenden wird ein Überblick über die Integration der Koordinationsverfahren in die Architektur eines QoS-Managers gegeben. Im Anschluss wird näher auf die durch das JGroups-Framework bereitgestellten Abstraktionen eingegangen.

7.2.1.1 Architekturüberblick

Die in Abschnitt 5.4.3 auf logischer Ebene beschriebene Integration von direkter Kommunikation zwischen QoS-Managern wurde im Rahmen der prototypischen Umsetzung des Ansatzes für das in Abschnitt 7.1.1 vorgestellte Selfmanager-Framework realisiert. Abbildung 7.14 zeigt die Anbindung des in Abschnitt 6.5.3 beschriebenen Auktionsansatzes an einen QoS-Manager. Die Anbindung der Kommunikationsarchitektur für QoS-Manager erfolgt über ein in der Abbildung als *Strategiemodul* bezeichnetes Modul, das mit dem jeweiligen Kontrollmodul des QoS-Managers interagiert. Das Strategiemodul beobachtet und bewertet, wie in Abschnitt 6.3 beschrieben, das Verhalten des QoS-Managers und versucht ggf., eine Lockerung oder Straffung des bestehenden SLOs zu erreichen. Um eine flexible Anpassbarkeit der verwendeten Bewertungsfunktion sicherzustellen, wird diese in der Implementierung des Moduls durch ein Skript realisiert. Als Skriptsprache verwendet der Prototyp Tcl auf Basis der Jacl-Laufzeitumgebung (vgl. [LS97]).

[7] Details unter `http://www.jgroups.org/`

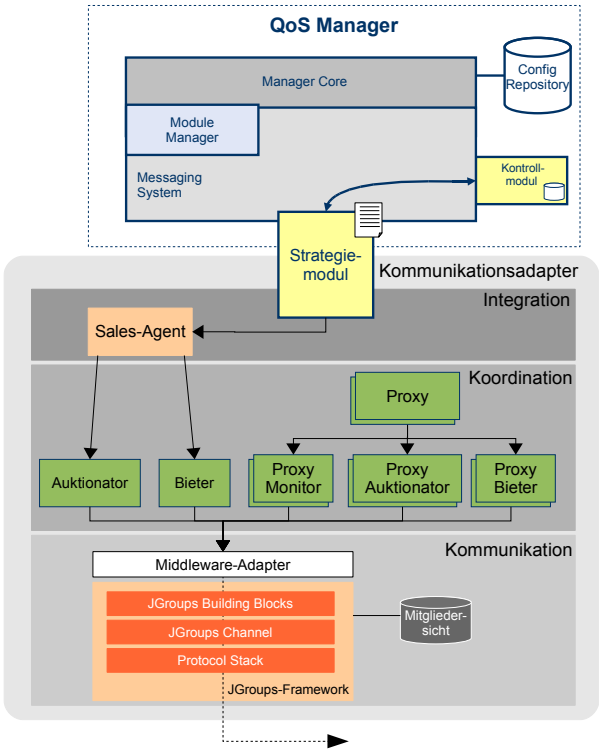

Abbildung 7.14: Implementierung des Kommunikationsadapters unter Nutzung von JGroups

> **Beispiel 7.2:** Für das in Abschnitt 7.1.4 beschriebene Szenario kann das Strategiemodul zum einen auf die Performance-Messdaten zurückgreifen, die der internen Bewertung durch das Kontrollmodul zugrunde liegen (vgl. Abbildung 7.11), und zum anderen die durch das Kontrollmodul ausgelösten Ereignismeldungen verarbeiten. Löst das Kontrollmodul das Ereignis *$event_overload* aus, veranlasst das das Strategiemodul, sich um die Lockerung des derzeit gültigen SLOs zu bemühen. Wird ein *$event_no_overload* empfangen, bricht das Strategiemodul laufende Verhandlungen über die Lockerung des SLOs ab.

Wird eine Straffung oder Lockerung des SLOs erreicht, informiert das Strategiemodul das Kontrollmodul des QoS-Managers über die Änderung, damit das Regelungsverhalten an das veränderte Ziel angepasst werden kann.

Die Kommunikation mit anderen QoS-Managern erfolgt durch den in Abbildung 7.14 abgebildeten *Kommunikationsadapter*. Dieser besteht aus drei Ebenen:

Auf der *Integrationsebene* befindet sich neben dem Strategiemodul ein *Sales Agent*, der die Anbindung des Koordinationsverfahrens gegenüber dem Strategiemodul kapselt. Der Sales Agent wird vom Strategiemodul mit der Straffung oder Lockerung des SLOs um einen be-

stimmten Betrag beauftragt und meldet nach erfolgter Koordination mit anderen QoS-Managern die erreichten Werte an das Strategiemodul zurück.

Zur Erledigung der an ihn gerichteten Aufträge kommuniziert der Sales Agent mit auf der *Koordinationsebene* angesiedelten Elementen. Hier befinden sich die Bieter- und Auktionator-Funktionalität für das umgesetzte Auktionsverfahren. Daneben sind auf der Koordinationsebene noch weitere Prozesse angesiedelt: Die Auswahl der QoS-Manager, die die in Abschnitt 6.5.4.1 vorgestellte Proxy-Funktionalität zur Realisierung strukturübergreifender Auktionen bereitstellen, erfolgt auf dieser Ebene. Wird ein QoS-Manager zum Koordinator einer Kommunikationsgruppe bestimmt, wird – transparent für die Elemente der Integrationsebene – ein entsprechendes Proxy-Modul instanziiert, das zur gruppenübergreifenden Koordination ebenfalls Auktionator- und Bieterfunktionalität instanziiert.

Die eigentliche Kommunikation mit anderen QoS-Managern wird durch die *Kommunikationsebene* realisiert. Ein Middleware-Adapter stellt für Bieter und Auktionator geeignete Abstraktionen bereit und kapselt so die in der prototypischen Implementierung verwendete Middleware JGroups.

Zur Koordination von QoS-Managern ist es notwendig, dass diese über auf Kommunikations- und Koordinationsebene kompatible Implementierungen verfügen: Grundlage für eine erfolgreiche Koordination ist zunächst die Nutzung eines einheitlichen Gruppenkommunikationssystems, weiterhin müssen die Auktionator- und Bieterkomponenten der beteiligten QoS-Manager ein gemeinsames Auktionsprotokoll implementieren.

Im Rahmen der prototypischen Implementierung des Ansatzes wurden zunächst unterschiedliche Koordinationsprotokolle auf ihre Anwendbarkeit auf die gegebene Problemstellung untersucht (Details in [Ste07]), bevor das in Abschnitt 6.4 beschriebene Verfahren entwickelt wurde.

Im Folgenden wird das Zusammenspiel der Komponenten auf der Koordinationsebene des in Abbildung 7.14 gezeigten Kommunikationsadapters anhand eines Beispiels erörtert.

> **Beispiel 7.3:** Abbildung 7.15 zeigt den Ablauf einer Auktion zur Übertragung von überschüssigen SLO-Anteilen an einen anderen QoS-Manager. Die durchgeführte Auktion folgt dem in Abbildung 6.12 in Abschnitt 6.5.3 dargestellten Ablauf.
>
> Das Beispiel zeigt einen QoS-Manager, der zum Koordinator der lokalen Kommunikationsgruppe `A` gewählt wurde und deshalb eine entsprechende Proxy-Komponente instanziiert hat (in der Abbildung ganz rechts dargestellt). Aufgabe des Proxys ist die Unterstützung von strukturübergreifender Kommunikation innerhalb eines Workflows. Zu diesem Zweck erzeugt der Proxy in der Kommunikationsgruppe des QoS-Managers (Gruppe `A`) und in der übergeordneten Kommunikationsgruppe (Gruppe `B`) jeweils einen Monitor, der Auktionsankündigungen innerhalb der jeweiligen Gruppe entgegennimmt und an den Proxy weiterleitet.

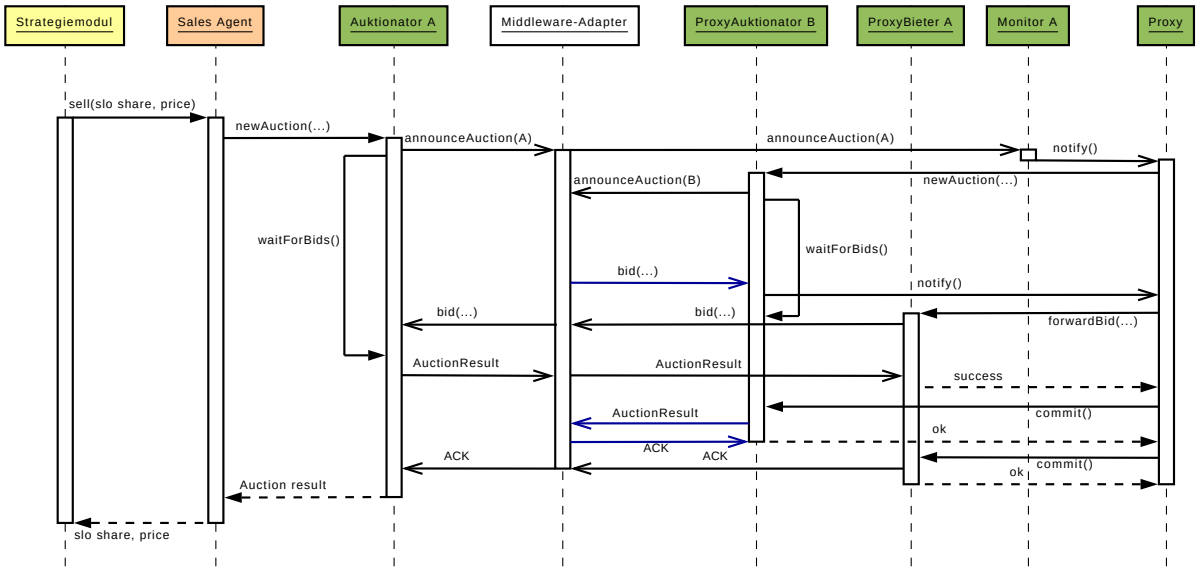

Abbildung 7.15: Ablauf einer strukturübergreifenden Auktion

Aus Gründen der Übersichtlichkeit ist der Monitor für Gruppe `B` in der Grafik nicht dargestellt.

Entscheidet das Strategiemodul des QoS-Managers, überschüssige SLO-Anteile abzugeben, erhält der Sales Agent einen entsprechenden Verkaufsauftrag. Er informiert den lokalen Auktionator darüber. Dieser beginnt nun mit der Auktion, indem er eine entsprechende Nachricht an die übrigen Gruppenmitglieder versendet. In der Abbildung wird das Versenden und Empfangen von Nachrichten durch die Kommunikation mit dem Middleware-Adapter dargestellt.

Neben den übrigen Gruppenmitgliedern empfängt auch der durch den Proxy instanziierte Monitor die Auktionsankündigung und leitet diese an den Proxy weiter. Der Proxy erzeugt nun seinerseits durch Beauftragung eines eigenen Auktionators eine äquivalente Auktion in der übergeordneten Kommunikationsgruppe `B`. Die Bietfrist für diese Auktion endet allerdings vor der Bietfrist für die in der Gruppe `A` bekannt gegebene Auktion.

Im Beispiel geht nun ein Gebot aus Gruppe `B` ein. Der Auktionator übermittelt das eingegangene Gebot an den Proxy. Dieser beauftragt seinerseits einen Proxy-Bieter zur Abgabe eines äquivalenten Gebots für die Auktion in Gruppe `A`. Der durch den Sales Agent beauftrage Auktionator empfängt neben diesem Gebot keine weiteren Gebote und erteilt dem einzigen Bieter den Zuschlag. Nach Empfang der Nachricht sendet der Bieter eine Erfolgsmeldung an den Proxy, dieser wiederum erteilt in Gruppe `B` den Zuschlag an den Bieter.

Im Anschluss erfolgt die Übermittlung der SLO-Anteile und die Bezahlung, wie

in Abschnitt 6.5.3 beschrieben, wobei die Kommunikation wieder über den Proxy abgewickelt wird. Nach Abschluss der Auktion teilt der Sales Agent dem Strategiemodul mit, zu welchem Erlös der SLO-Anteil veräußert werden konnte. Dieser korrigiert die lokalen Ziele des QoS-Managers entsprechend dem Auktionsergebnis.

7.2.1.2 Das JGroups-Gruppenkommunikationssystem

JGroups stellt einen zuverlässigen Gruppenkommunikationsmechanismus für Java-Anwendungen bereit. Grundlegende Kommunikationseigenschaften des Frameworks (z. B. bezüglich Ordnung und Zuverlässigkeit) lassen sich konfigurieren [Ban98, Red09], sodass sich JGroups leicht an Anforderungen unterschiedlicher Anwendungen anpassen lässt.

Gruppen werden in JGroups durch einen eindeutigen Namen identifiziert, eine neue Gruppe wird erzeugt, indem ein Prozess einer bisher nicht existenten Gruppe beitritt. JGroups implementiert geschlossene Kommunikationsgruppen, d. h. um eine Nachricht an die Mitglieder einer Gruppe senden zu können, muss der Absender selbst Mitglied der Gruppe sein. Prozesse können Nachrichten entweder an einzelne Prozesse (als unicast) oder alle Prozesse der Gruppe (multicast) senden.

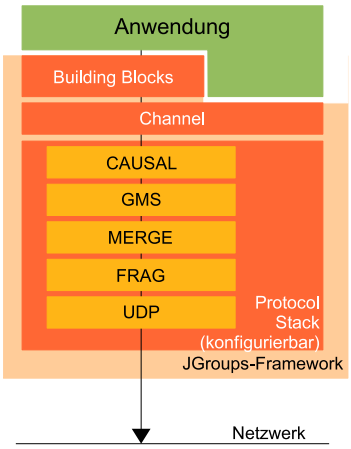

Abbildung 7.16: JGroups-Architektur nach [Red09]

Abbildung 7.16 gibt einen Überblick über die Architektur des JGroups Frameworks. Anwendungen interagieren entweder mit dem *JGroups Channel API*, das eine grundlegende asynchrone Kommunikationsschnittstelle bereitstellt, oder nutzen sog. *JGroups Building Blocks*, die – basierend auf dem Channel API – eine höhere Abstraktionsebene offerieren. Beispiele für Building Blocks sind ein Gruppen-RPC-Mechanismus oder die Realisierung synchroner Kommunikation. Unterlagert ist jedem JGroups Channel ein konfigurierbarer Protokoll-Stack, über den Nachrichten an die Gruppenmitglieder gesendet werden. Die Gruppenmitgliedschaft

eines Prozesses ist an einen Channel gebunden; um gleichzeitig in mehreren Gruppen Mitglied zu sein, kann ein Prozess parallel mehrere Channels nutzen.

Grundlage für durch JGroups leistbare Garantien bezüglich Zuverlässigkeit und Nachrichtenreihenfolge ist die Konfiguration eines geeigneten Protokoll-Stacks. Die durch JGroups bereitgestellten Protokolle sind dabei nicht nur für den Nachrichtenversand und -empfang (wahlweise über UDP oder TCP) verantwortlich, sondern realisieren auch eine Reihe funktionaler und nichtfunktionaler Eigenschaften.

JGroups-Protokolle implementieren grundlegende Funktionen wie den Mitgliederservice, Fehlererkennung, Erkennung von Gruppenfragmentierung, Zusammenführen der Teile fragmentierter Gruppen, Durchsetzen bestimmter Ordnungseigenschaften für Nachrichten (FIFO, kausale oder totale Ordnung) usw. Zusätzliche Protokolle unterstützen optional Kompression und Verschlüsselung von Nachrichten.

In Abbildung 7.16 ist zum besseren Verständnis ein Beispiel-Protokoll-Stack dargestellt:

- *CAUSAL*
  garantiert eine kausale Ordnung über die eingehenden Nachrichten, Grundlage für das Protokoll ist der Vector-Clock-Algorithmus.
- *GMS*
  implementiert den Mitgliederservice und behandelt Beitritte, Austritte und (vermutete) Fehler einzelner Gruppenmitglieder.
- *MERGE*
  erkennt und behebt temporäre Fragmentierungen einer Gruppe.
- *FRAG*
  behandelt aufgrund ihrer Größe fragmentierte Nachrichten.
- *UDP*
  implementiert den Versand/Empfang von Nachrichten über UDP.

Einen detaillierten Überblick über die Architektur und die Konfiguration von JGroups gibt [Red09].

7.2.1.3 Umsetzung der Koordinationsverfahren mit JGroups

In Abschnitt 6.5.4.4 sind die Anforderungen des Vickrey-Auktionsprotokolls an das unterlagerte Gruppenkommunikationssystem beschrieben. Mithilfe einer geeigneten Konfiguration kann JGroups diese Anforderungen erfüllen.

Auf unterster Ebene des JGroups-Protokollstacks muss zunächst über das zu verwendende Transportprotokoll entschieden werden. JGroups bietet eine unzuverlässige Kommunikation

über UDP und einen zuverlässigen TCP-basierten Transportmechanismus an. Wird UDP genutzt, steht ein echter Multicast-Mechanismus zur Verfügung, allerdings muss eine evtl. benötigte zuverlässige Nachrichtenübertragung durch zusätzlich zu konfigurierende JGroups-Protokolle abgesichert werden. Bei Verwendung des TCP-Transports wird ein Multicast auf n Unicast-Nachrichten abgebildet.

Für die Nutzung zur Kommunikation zwischen in unterschiedlichen administrativen Domänen befindlichen SOA-Diensten erscheint es fraglich, ob überhaupt auf Multicast-Kommunikation zurückgegriffen werden kann, da hierfür eine entsprechende Konfiguration auf Netzwerkebene Voraussetzung ist. Daher wird für die prototypische Implementierung im Sinne einer Worst-Case-Betrachtung der JGroups-TCP-Transportmechanismus gewählt. Dieser wird so konfiguriert, dass kleine Nachrichten an den gleichen Empfänger gebündelt werden, um die Netzwerklast zu minimieren.

Das MERGE2-Protokoll wird zum Erkennen und Beheben temporärer Gruppenfragmentierungen verwendet.

Zur Fehlererkennung kommt das FD-Protokoll zum Einsatz. Dieses tauscht zwischen den Prozessen einer Gruppe regelmäßige Heartbeat-Nachrichten aus. Reagiert ein Prozess nicht mehr, wird dies an den Mitgliederservice weitergeleitet, damit der defekte Prozess aus der Gruppe entfernt werden kann.

Der JGroups Mitgliederservice (GMS) garantiert eine einheitliche Sicht aller Gruppenmitglieder. Der GMS liefert allen Mitgliedern zudem eine einheitlich sortierte Mitgliederliste, sodass zur Auswahl eines Koordinators kein gesondertes Protokoll eingesetzt werden muss. In der prototypischen Implementierung wird als Koordinator einer Gruppe der erste Listeneintrag des GMS bestimmt, sodass dieser Knoten den notwendigen Kommunikations-Proxy instanziiert.

Weiterhin stellt das JGroups-Framework einen Building Block bereit, der die Replizierung von Daten zwischen Gruppenmitgliedern erlaubt. Dieser wird zur Kommunikation der durch die von der zustandsbehafteten Proxy-Komponente vorzuhaltenden Informationen genutzt, sodass im Fall eines Ausfalls des gewählten Koordinators eine andere Komponente diese Aufgabe übernehmen kann.

7.2.2 Simulator für Manager-Kooperation

Neben der in Abschnitt 7.2.1 beschriebenen prototypischen Umsetzung der Koordinationsschnittstelle eines QoS-Managers wurde ein Simulator zur Untersuchung der implementierten Koordinationsverfahren entwickelt. Mit dessen Hilfe können isoliert Skalierbarkeitsaspekte dieser Verfahren untersucht werden.

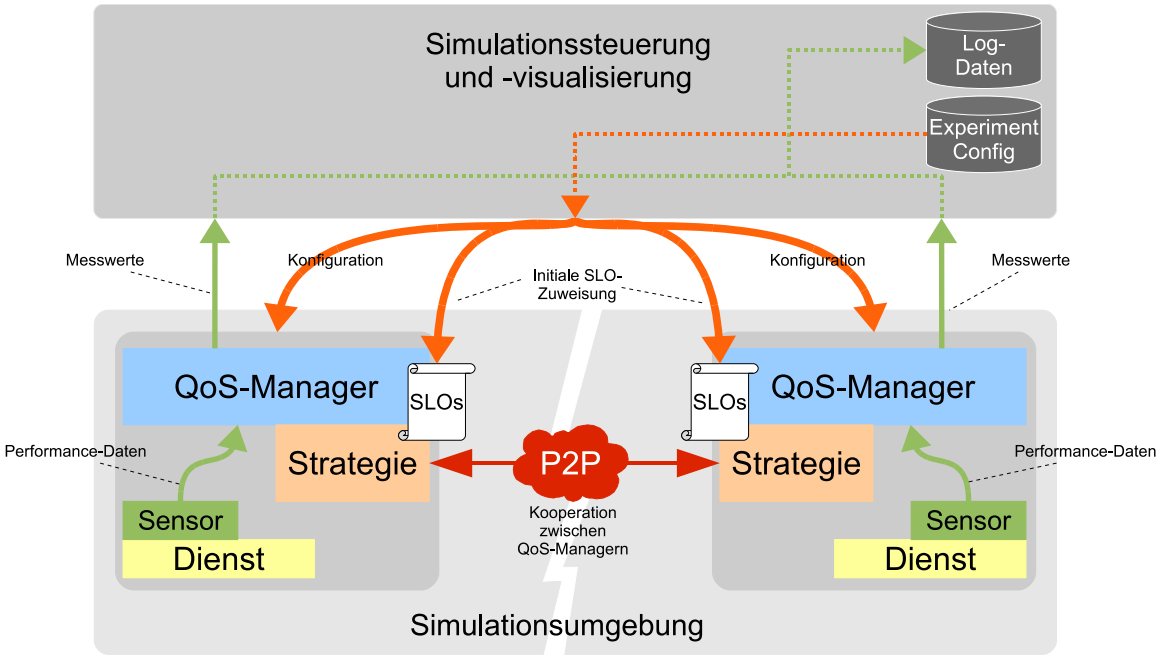

Abbildung 7.17: Simulator zum Test von Mechanismen zur Manager-Kooperation

Abbildung 7.17 zeigt die Architektur des entwickelten Simulators im Überblick. Diese besteht aus zwei Teilen, der *Simulationssteuerung und -visualisierung* und der eigentlichen *Simulationsumgebung*.

7.2.2.1 Simulationssteuerung

Mithilfe der Simulationssteuerung können unterschiedliche Dienste-Topologien kreiert werden. Für jeden Dienst wird dabei ein Antwortzeitprofil hinterlegt. Für einen Simulationslauf werden die Dienste einer Topologie temporär durch Workflows verknüpft. Ein Workflow ist als strukturierte Abfolge von Aktivitäten modelliert. Aktivitäten referenzieren ihrerseits Dienste einer Topologie. Beim Aufsetzen einer Simulation werden Diensten initiale SLOs zugeordnet, die (wie in Abschnitt 6.7 beschrieben) aus dem Gesamt-SLO des Workflows abgeleitet werden. Zur Laufzeit der Simulation transferieren die QoS-Manager der einzelnen Dienste SLO-Anteile nach dem in Abschnitt 6.4 beschriebenen Verfahren.

Die Simulationssteuerung visualisiert den aktuellen Zustand eines Experiments zur Laufzeit. Abbildung 7.18 stellt die Workflow-Struktur einer laufenden Simulation dar. Dienste, die aktuell das ihnen zugeordnete SLO verletzen, werden rot dargestellt, Dienste ohne SLO-Verletzung grün. Weiterhin wird der Zustand jedes Dienstes in einer Log-Datei abgelegt, sodass sich nach Abschluss eines Experimentlaufs detaillierte Auswertungen durchführen lassen.

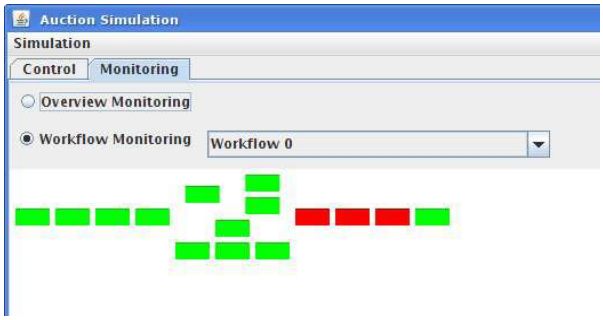

Abbildung 7.18: Überwachung einer laufenden Simulation durch die Simulationssteuerung

7.2.2.2 Simulationsmodell

Das entwickelte Simulationsmodell reduziert einen SOA-Dienst auf die für die Simulation relevanten Aspekte. Für die Simulation kooperierender QoS-Manager ist die durch den verwalteten Dienst erbrachte Geschäftsfunktionalität zweitrangig, von Interesse sind insbesondere die Struktur des modellierten Workflows und das Antwortzeitverhalten der einzelnen Dienste. Die Struktur des Workflows wird zur initialen SLO-Zuordnung (vgl. Abschnitt 6.7) und zur Definition von Kommunikationsgruppen (vgl. Abschnitt 6.5.4.3) benötigt. Auf Basis der aktuellen Antwortzeit eines Dienstes und dessen aktuellem SLO kann der QoS-Manager entscheiden, ob er weitere SLO-Anteile benötigt, überzählige SLO-Anteile abgeben oder nichts unternehmen will. Daher wird ein Dienst in der Simulation im Weiteren auf ein flexibles Antwortzeitmodell reduziert.

Abbildung 7.19 zeigt beispielhaft eine durch den Simulator interpretierbare Dienste-Topologie sowie ein Workflow-Modell, das Dienste dieser Topologie verknüpft. Die der XML-Beschreibung zugrunde liegenden Schemata sind in Anhang B abgebildet. Das Dienst-Modell ordnet Diensten jeweils eine numerische ID und ihre Antwortzeit zu Beginn der Simulation zu. Durch `<responseTimeModel>`-Elemente kann das Antwortzeitverhalten eines Dienstes im Laufe der Simulation modifiziert werden. In der Abbildung sind solche Änderungen für die Dienste mit ID `0` und `1` festgelegt. Für Dienst `2` wird im Antwortzeitmodell eine Zeitspanne definiert, innerhalb der sich die Antwortzeit des Dienstes gleichverteilt auf einen zufälligen Wert zwischen 10 und 25 Zeiteinheiten verändert.

Das Workflow-Modell definiert geschachtelte Workflow-Strukturen, in denen `<activity>`-Elemente die Dienste der Topologie referenzieren. In der Abbildung werden die Dienste sequenziell aufgerufen, wobei jede Aktivität die ID des referenzierten Dienstes spezifiziert. Das Workflow-Modell definiert das Workflow-weite SLO durch das Element `<maxResponseTime>`, aus dem entsprechend der Workflow-Struktur die SLOs für die einzelnen Dienste abgeleitet werden. Weiterhin kann für jeden Workflow eine numerische Priorität festgelegt werden, die dem QoS-Manager eines Dienstes in einer Simulation mit mehreren Workflows dazu dient, in Konfliktsituationen Priorisierungen vorzunehmen.

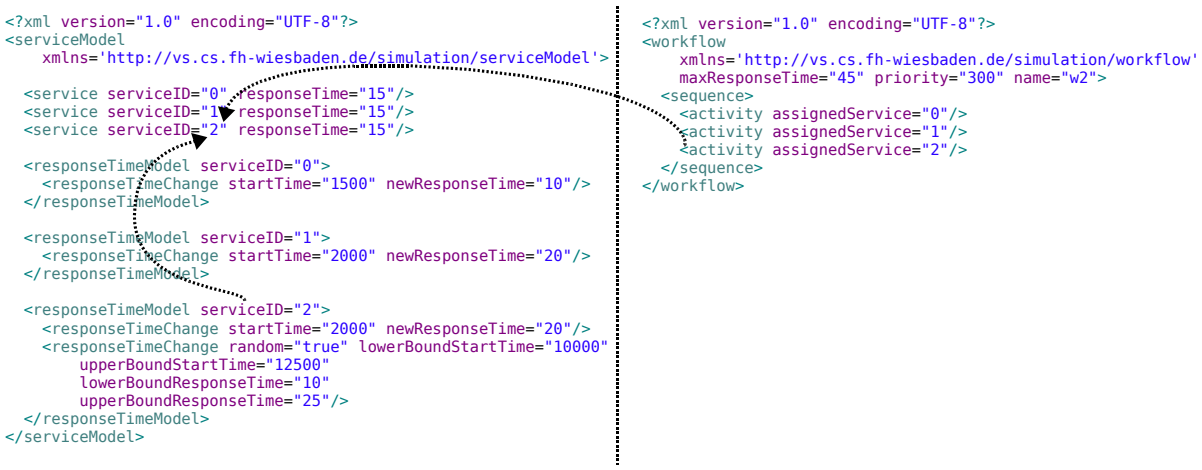

Abbildung 7.19: Beispiel für das Zusammenspiel zwischen Dienst- und Workflow-Modell

7.2.2.3 Simulationsumgebung

Die Simulationsumgebung stellt eine Laufzeitumgebung für die einzelnen Dienste und ihre QoS-Manager bereit. Beide werden dabei stark vereinfacht modelliert: Dienste werden im Wesentlichen durch ein Antwortzeitmodell und einen Performance-Sensor repräsentiert. Der einem Dienst zugeordnete QoS-Manager fragt diesen Performance-Sensor ab und entscheidet innerhalb seines Strategiemoduls, ob das dem Dienst zugewiesene SLO erweitert oder gestrafft werden soll. Detailliert modelliert ist der Kommunikationsadapter des QoS-Managers: Er wird durch ein Teilsystem realisiert, das die in Abschnitt 6.5.3 spezifizierten Protokolle umsetzt und den QoS-Manager eines Dienstes mit der Kommunikationsinfrastruktur des Simulators verbindet. Die Kommunikation zwischen unterschiedlichen Managern wird auf das echte JGroups-Framework abgebildet, d. h. nicht simuliert. Die Simulationsumgebung realisiert somit einen hybriden Ansatz aus Teilsimulation und Echtsystem.

Aufsetzen eines Simulationsszenarios Abbildung 7.20 zeigt die Interaktion zwischen Simulationssteuerung und Simulationsumgebung beim Aufsetzen eines Simulationsszenarios. Die Kommunikation zwischen der als *Simulation Controller* bezeichneten Simulationssteuerung den den anderen Systemkomponenten erfolgt über JMX, sodass sich Steuerung/Visualisierung und Simulationsumgebung auch auf unterschiedlichen Rechnern befinden können.

Im oberen Teil der Abbildung sind die Abläufe beim Erzeugen eines Dienstes aus dem Topologiemodell dargestellt. Die Simulationssteuerung liest die Topologie ein und beauftragt die Simulationsumgebung über `createService()`-Aufrufe, die spezifizierten Dienste zu erstellen. Die Simulationsumgebung erzeugt für jeden Dienst eine Instanz der Klasse `Service`. Diese repräsentiert die Implementierung der Geschäftsfunktion eines Dienstes. Ihre Funktion

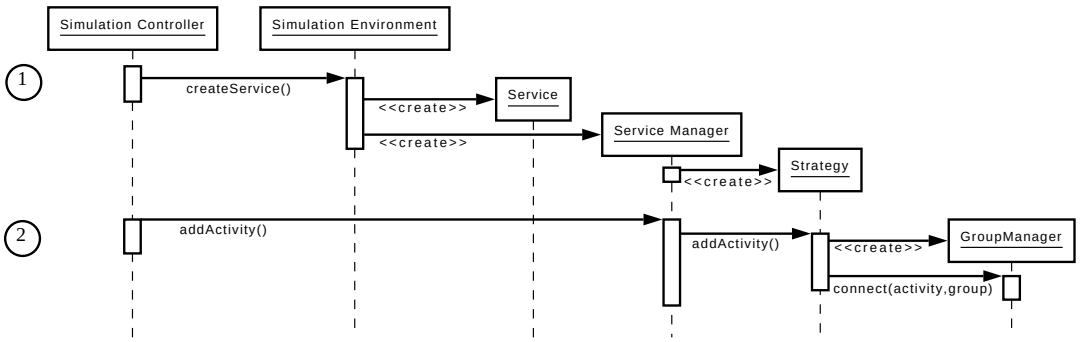

Abbildung 7.20: Interaktion zwischen Simulationssteuerung und Simulationsumgebung beim Aufsetzen einer Simulation

innerhalb der Simulation beschränkt sich auf das Bereitstellen von in der Topologie spezifizierten Antwortzeitinformationen (vgl. Abbildung 7.19). Weiterhin wird für jeden Dienst ein `ServiceManager` erzeugt, der den dem Dienst zugeordneten QoS-Manager repräsentiert und Zugriff auf das aktuelle SLO und Antwortzeitinformationen des verwalteten Dienstes hat. Der ServiceManager instanziiert den Kommunikationsadapter zur Kooperation mit anderen Diensten. Dieser wird in der Abbildung durch die Klasse `Strategy` repräsentiert. Die Strategy-Klasse kapselt das Zusammenspiel der in Beispiel 7.3 beschriebenen Module.

Nachdem die Simulationssteuerung auf diese Weise die Topologie in der Simulationsumgebung instanziiert hat, können die vorhandenen Dienste über Aktivitäten eines Workflow-Modells verknüpft werden. Das Hinzufügen einer Aktivität ist im unteren Teil von Abbildung 7.20 dargestellt. Die Simulationssteuerung liest ein Workflow-Modell ein und ruft für jede hinzuzufügende Aktivität den `ServiceManager` des durch die Aktivität referenzierten Dienstes auf. Der `ServiceManager` registriert die Aktivität und informiert die dem Dienst zugeordnete Strategie (als Repräsentation des Kommunikationsadapters) über die neue Aktivität. Die Strategy-Klasse registriert den durch die Aktivität referenzierten Dienst dann in den entsprechenden JGroups-Kommunikationsgruppen. Im Rahmen dieser Registrierung werden auch die Auswahl des für eine Kommunikationsgruppe verantwortlichen Koordinators und die Registrierung der Proxy-Funktionalität durchgeführt.

Der Start der Simulation erfolgt durch Starten der einzelnen `ServiceManager`-Komponenten. Jeder `ServiceManager` läuft dabei als eigener Thread, sodass einzelne Dienste zeitlich unabhängig voneinander Entscheidungen treffen und Kommunikationsprozesse anstoßen können. Weitere Threads werden durch das JGroups-Gruppenkommunikationssystem für die Realisierung der Channel-Infrastruktur instanziiert.

Ablauf der Auktionsmechanismen Nach dem Start evaluiert ein `ServiceManager`, ob zur Einhaltung des SLOs zusätzliche SLO-Anteile benötigt werden, ob überschüssige SLO-

Anteile abgestoßen werden sollen oder ob derzeit nichts unternommen werden muss. Hierzu bezieht der `ServiceManager` Informationen aus dem Performance-Sensor des Dienstes. Müssen zusätzlich SLO-Anteile erworben werden, erzeugt der `ServiceManager` eine `BUY`-Aktivität, sollen Anteile abgegeben werden entsprechend eine `SELL`-Aktivität. Diese Aktivitäten sind als Threads realisiert und werden in einen Thread Pool eingegliedert, der der Auktionsstrategie zugeordnet ist.

Während eine `BUY`- oder `SELL`-Aktivität aktiv ist, verhält sich der `ServiceManager` passiv, d. h. er stößt parallel keine weiteren Aktivitäten an.

Jede Aktivität erzeugt und kontrolliert eine Reihe von Jobs, über die das in Abschnitt 6.5.3 spezifizierte Vickrey-Auktionsprotokoll abgewickelt wird. Jeder Job ist wiederum als Thread implementiert. Im Rahmen einer `BUY`-Aktivität wird beispielsweise zunächst ein `BidJob` erzeugt, der auf Auktionsankündigungen wartet und bei Eingang einer geeigneten Ankündigung ein Gebot abgibt. Gehen innerhalb einer definierten Zeitperiode keine Ankündigungen ein, beendet sich der `BidJob`, und die `BUY`-Aktivität wird erfolglos beendet. Hat er ein Gebot abgegeben, instanziiert der `BidJob` einen `AwaitAuctionOutcome`-Job und beendet sich selbst. Der `AwaitAuctionOutcome`-Job koordiniert dann die weitere Abhandlung des Auktionsprotokolls.

Die Durchführung einer Auktion im Rahmen einer `SELL`-Aktivität und die Proxy-Funktionalität zur gruppenübergreifenden Koordination von QoS-Managern sind analog implementiert. Durch die vorgenommene Modularisierung lassen sich elegant komplexe Protokolle realisieren.

Simulation großer Systeme Die Umgebung erlaubt die Konfiguration von zwei alternativen Durchführungsszenarien: Zum einen können – wie oben beschrieben – Dienste und QoS-Manager innerhalb der Simulationsumgebung als *Threads* gestartet werden, zum anderen können Dienste und Manager in einer verteilten Simulation als *Systemprozesse* auf unterschiedlichen, über ein Netzwerk verbundenen Rechnern betrieben werden. Im ersten Fall erfolgt die Kommunikation zwischen den QoS-Managern über einen JGroups-Loopback-Protokolladapter innerhalb des Prozesses der Simulationsumgebung. Die Größe und Komplexität der simulierbaren Umgebung ist hierbei durch die Ressourcen des Systems begrenzt, auf dem die Simulation ausgeführt wird. Im zweiten Fall wird zur Kommunikation ein vollständiger JGroups-Protokoll-Stack verwendet. Hier können zusätzlich Aspekte wie Netzwerklatenzen und induzierte Last untersucht werden. Die gewonnenen Ergebnisse sind jedoch lediglich für die in der Implementierung verwendete Middleware JGroups aussagekräftig, Schlussfolgerungen lassen sich nur schlecht auf alternative Gruppenkommunikationsplattformen übertragen.

Das Aufsetzen einer verteilten Simulation ist deutlich komplexer als die lokale Variante, erlaubt aber die Abbildung großer Systeme, für deren Simulation die Ressourcen eines einzelnen Rechners nicht ausreichen. Wird die Simulation verteilt gestartet, wird die Funktionalität der

zentralen Simulationssteuerung durch die einzelnen Simulationsumgebungen übernommen. Konfigurationsdateien mit Topologie und Workflow-Modell werden dann lokal auf jedem Knoten vorgehalten. Zur Auswertung einer Simulation werden die jeweils lokal erzeugten Log-Dateien nach Abschluss der Simulation zusammengeführt.

Eine detaillierte Beschreibung der Implementierung des Simulators findet sich in [Jun08].

7.3 Zusammenfassung

In diesem Kapitel wurde die prototypische Umsetzung des in den Kapiteln 5 und 6 entwickelten Ansatzes beschrieben.

In Abschnitt 7.1 wurde die Implementierung der in Abschnitt 5.4 spezifizierten QoS-Manager für Workflows und für Dienste anhand konkreter Beispiele vorgestellt. Als technische Grundlage für alle implementierten QoS-Manager wurde zunächst in Abschnitt 7.1.1 das Selfmanager-Framework eingeführt.

In Abschnitt 7.2.1 wurde detailliert auf die Umsetzung des Kommunikationsadapters eines QoS-Managers mithilfe des JGroups-Gruppenkommunikationssystems eingegangen. Weiterhin wurde eine hybride Simulationsumgebung (vgl. Abschnitt 7.2.2) zur Untersuchung der beschriebenen Kooperationsverfahren vorgestellt.

In Kapitel 8 werden Erfahrungen im Umgang mit der Simulationsumgebung präsentiert. Diese wird im Rahmen der Evaluation für die Untersuchung der Skalierbarkeit des Ansatzes zur QoS-Manager-Kooperation eingesetzt.

8 Evaluation

In diesem Kapitel wird der Ansatz zur Übertragung von Antwortzeit-SLO-Anteilen zwischen kooperierenden Managern einer Evaluation unterzogen. Diese Evaluation konzentriert sich im Wesentlichen auf zwei Punkte: Zunächst wird in Abschnitt 8.1 untersucht, inwieweit die in Abschnitt 5.1 definierten Anforderungen durch die in Kapitel 7 beschriebene prototypische Implementierung des Ansatzes erfüllt werden. Dann wird mithilfe des in Abschnitt 7.2.2 vorgestellten Simulators die Skalierbarkeit des Ansatzes bewertet. Dabei wird die Funktionsweise des Simulators in Abschnitt 8.2 zunächst anhand eines überschaubaren Beispiels beschrieben, in Abschnitt 8.3 werden dann Ergebnisse aus der Vermessung größerer Workflows präsentiert.

8.1 Betrachtung der Implementierung im Hinblick auf die definierten Anforderungen

Im Folgenden wird der implementierte Prototyp des Dienstgüte-Management-Systems im Hinblick auf die Erfüllung der in Kapitel 5 definierten Anforderungen betrachtet. Abschnitt 8.1.1 untersucht das System in Bezug auf funktionale Anforderungen. In Abschnitt 8.1.2 werden nichtfunktionale Aspekte des Systems diskutiert. Abschnitt 8.1.3 widmet sich schließlich der Betrachtung von Integrationsaspekten.

8.1.1 Funktionale Anforderungen

Tabelle 8.1 gibt einen Überblick über den Erfüllungsgrad der in Abschnitt 5.1.1 definierten funktionalen Anforderungen für das Dienstgüte-Management-System. Vollständig erfüllte Anforderungen sind durch das Symbol ☑ gekennzeichnet, größtenteils erfüllte Anforderungen durch das Symbol ☑. Anforderungen, die nur ansatzweise erfüllt werden konnten, sind durch das Symbol □ gekennzeichnet.

ANFORDERUNG		BEMERKUNG
F1 *Formale Definition von Dienstgüteanforderungen*	☑	Mithilfe der WS-Agreement-SLA-Spezifikation (vgl. Abschnitt 3.2.3.1) und des in Anhang B.1 vorgestellten Schemas zur Definition von SLOs für die Dienstgütedimension *maximale Antwortzeit* erfüllt.
F2 *Entgegennahme von Dienstgüteanforderungen*	☑	Durch Integration der WS-Agreement Client/Server-Schnittstellen in die QoS-Manager für Workflows und für Dienste realisiert (vgl. Abschnitte 7.1.3 und 7.1.4).
F3 *Abbildung von Dienstgüteanforderungen*	☑	Definition initialer Antwortzeit-SLOs Bestandteil des QoS-Managers für Workflows (vgl. Abschnitt 6.7), Implementierung des Verfahrens als Teil des Simulators (vgl. Abschnitt 7.2.2).
F4 *Durchsetzung von Dienstgüteanforderungen*	☑	Die Anforderung muss spezifisch für den verwalteten Dienst und die betrachtete Dienstgütedimension umgesetzt werden. Realisierte Beispiele: Management von Diensten auf Basis virtueller Maschinen (vgl. Abschnitt 6.6.1), Management des JBoss AS (vgl. Abschnitt 7.1.4).
F5 *Eskalation*	☑	Durch Integration der WS-Agreement Client/Server-Schnittstellen in die QoS-Manager für Dienste und für Workflows realisiert (vgl. Abschnitt 5.4.1.2).
F6 *Behandlung von Konflikten*	☐	Erste Ansätze zur Behandlung von SLO-Konflikten sind in Form der QoS-Proxy-Komponente (vgl. Abschnitt 7.1.5) und im SLM-Ansatz für Dienste auf Basis virtueller Maschinen beschrieben (vgl. Abschnitt 6.6.1).
F7 *Verteilte Kontrolle*	☑	Die entwickelte Architektur basiert auf unabhängigen QoS-Managern, die mit dem Ziel der Einhaltung übergeordneter SLOs kooperieren. Die Kontrolle über einen Dienst liegt stets beim lokal zugeordneten QoS-Manager.

Tabelle 8.1: Evaluation bezüglich der in Abschnitt 5.1.1 definierten funktionalen Anforderungen

Durch die Konzentration der Arbeit auf maximale Antwortzeit wird Anforderung **F1** nicht allgemein, sondern nur in Bezug auf diese Dienstgütedimension erfüllt. In der Folge konzentrieren sich die zur Behandlung der Anforderung **F6** genannten Ansätze ebenfalls auf diese Dimension. Konfliktlösungsstrategien für SLOs sind allerdings prinzipiell abhängig von der jeweils betrachteten Dienstgütedimension.

8.1.2 Nichtfunktionale Anforderungen

Tabelle 8.2 gibt einen Überblick über den Erfüllungsgrad der in Abschnitt 5.1.2 für das Dienstgüte-Management-System definierten nichtfunktionalen Anforderungen. Der jeweilige Erfüllungsgrad einer Anforderung ist mit den in Abschnitt 8.1.1 beschriebenen Symbolen gekennzeichnet.

ANFORDERUNG		BEMERKUNG
N1 *Orientierung an Geschäftsarchitektur*	☑	Die Kommunikationsbeziehungen des Systems orientieren sich weitestgehend an der Geschäftsarchitektur der verwalteten SOA: Es kooperieren die QoS-Manager der inhaltlich durch einen Workflow verbundenen Dienste. QoS-Manager und Dienst bilden eine Einheit (vgl. Abschnitt 5.3), die QoS-Manager unterschiedlicher Dienste sind nur lose gekoppelt.
N2 *Fokus*	☐	Das im Zentrum dieser Arbeit stehende Verfahren zur Kooperation von QoS-Managern eines Workflows fokussiert auf den einzelnen Workflow und nicht auf die Applikationslandschaft als Ganzes. Als applikationsübergreifender SLM-Mechanismus kann aber der Ansatz zum Management für Dienste auf Basis virtueller Maschinen betrachtet werden (vgl. Abschnitt 6.6.1).
N3 *Skalierbarkeit*	☑	Untersuchungen zur Skalierbarkeit werden in Abschnitt 8.3 präsentiert.
N4 *Robustheit*	☑	Das entwickelte Kooperationsverfahren für QoS-Manager ist gegenüber Änderungen auf Dienst- und Workflow-Ebene robust. Bei Änderungen an der Implementierung eines Dienstes muss der verwendete Kontrollalgorithmus angepasst werden, u. U. ist bei größeren Änderungen auch eine Anpassung der Monitoring- und Control-Funktionalität notwendig. Der modulare Aufbau eines QoS-Managers trägt zur Robustheit des Systems bei (vgl. Abschnitt 7.1.1).

Tabelle 8.2: Evaluation bezüglich der in Abschnitt 5.1.2 definierten nichtfunktionalen Anforderungen

Das in dieser Arbeit beschriebene Verfahren zur Kooperation von QoS-Managern muss im Rahmen eines SOA-weiten SLMs im Kontext der in Kapitel 4 zitierten Arbeiten [BGR+05] und [MDGA08] gesehen werden. Gemeinsam mit diesen Ansätzen kann eine Priorisierung einzelner Workflows und Dienste auf Basis von sich aus dem Geschäftskontext ergebenen Kennzahlen erfolgen: Die in dieser Arbeit vorgestellten Verfahren erlauben damit eine Verknüpfung des Managements der technischen Infrastruktur mit Geschäftszielen und Kostenaspekten.

Durch konsequente Modularisierung der Architektur wurde das Ziel verfolgt, robust auf Änderungen an Diensten und deren Management-Schnittstellen reagieren zu können. Aufgrund der engen Kopplung zwischen Dienst und QoS-Manager sind jedoch i. d. R. Anpassungen an letzterem nicht zu vermeiden. Langfristig sollten Monitoring- und Management-Aspekte jedoch als integraler Aspekt des Designs jeder auf den Bereich der Geschäftsanwendungen zielenden Softwarekomponente angesehen werden, so wie dies für Sicherheits- und Authentifizierungsaspekte heute schon gilt.

8.1.3 Integrationsanforderungen

Tabelle 8.3 gibt einen Überblick über den Erfüllungsgrad der in Abschnitt 5.1.3 für das Dienstgüte-Management-System definierten Integrationsanforderungen. Der jeweilige Erfüllungsgrad einer Anforderung ist mit den in Abschnitt 8.1.1 beschriebenen Symbolen gekennzeichnet.

ANFORDERUNG		BEMERKUNG
I1 *Transparente Integration*	☑	Ein Ansatz für die transparente Integration des Dienstgüte-Management-Systems in existierende SOA-Umgebungen wird für SCA in Abschnitt 5.5.2 vorgestellt.
I2 *Eingeschränkte Kooperation*	☑	Bei der initialen SLO-Zuordnung (vgl. Abschnitt 6.7) werden Dienste, denen kein QoS-Manager zugeordnet ist, explizit berücksichtigt. Das Verfahren zur Kooperation von QoS-Managern funktioniert auch dann, wenn sich nur ein Teil der Dienste kooperativ verhält, erreicht dann allerdings ggf. schlechtere Ergebnisse.
I3 *Nutzung existierender Funktionalität*	☑	Durch den modularen Aufbau der zum Einsatz kommenden QoS-Manager (vgl. Abschnitt 7.1.1) soll die Nutzung existierender Management-Schnittstellen vereinfacht werden. In der Implementierung werden eine Reihe von Modulen zum Zugriff auf unterschiedliche Standard-Monitoring- und Management-Schnittstellen vorgestellt.
I4 *Integration mit anderen SLM-Architekturen*	☑	Eine Integration mit anderen Management-Architekturen kann über die WS-Agreement-Schnittstellen der QoS-Manager erfolgen. Über diese Schnittstellen können auf SLA-Ebene Zielvorgaben getroffen werden, bei Verletzung der gesetzten Ziele erfolgt eine Benachrichtigung des Vertragspartners.

Tabelle 8.3: Evaluation bezüglich der in Abschnitt 5.1.3 definierten Integrationsanforderungen

8.2 Untersuchung des Kooperationsverfahrens anhand eines Beispielprozesses

Anhand eines einfachen Beispielprozesses soll im Folgenden die Funktionsweise des in Abschnitt 7.2.2 vorgestellten Simulators für die Kooperation von QoS-Managern auf Basis des Vickrey-Auktionsverfahrens nachvollzogen werden.

8.2.1 Anwendungsszenario

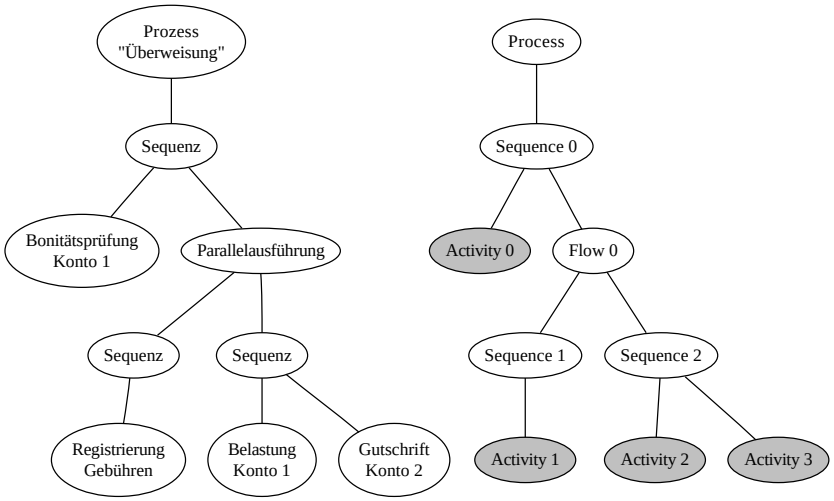

Abbildung 8.1: Beispiel-Workflow „Überweisung“ mit mehreren Sequenzen und einer Parallelausführung

Als Anwendungsbeispiel dient das Szenario einer Überweisung zwischen zwei Konten einer Bank. In einem ersten Schritt wird nach Eingang des Auftrags die Deckung des Kontos überprüft, von dem das Geld überwiesen werden soll. Ergibt diese Bonitätsprüfung ein positives Ergebnis, kann der eigentliche Überweisungsprozess eingeleitet werden. Dieser besteht aus zwei Schritten: Zunächst wird der zu transferierende Geldbetrag vom Ursprungskonto abgebucht, im Anschluss wird er auf das Zielkonto aufgebucht. Parallel zur Überweisung werden die dafür anfallenden Kosten notiert. Da diese dem Kunden erst mit der nächsten Quartalsabrechnung in Rechnung gestellt werden, erfolgt keine sofortige Belastung des Kundenkontos. Allerdings werden für den Überweisungsauftrag auch dann Kosten fällig, wenn der eigentliche Vorgang fehlschlägt (z. B. aufgrund einer ungültigen Zielkontonummer).

Die Bank untergliedert sich organisatorisch in mehrere Bereiche, die unterschiedliche Rechenzentren betreiben. Daher werden für die Inanspruchnahme von Diensten zwischen den Bereichen und Rechenzentren SLOs definiert. Die Bank ist allerdings stets auf Optimierung

der Gesamtprozesse bedacht, sodass kurzfristige Kooperationen unterschiedlicher Organisationseinheiten erwünscht sind.

Im linken Teil von Abbildung 8.1 ist der beschriebene Prozess in Baumdarstellung abgebildet, auf der rechten Seite eine durch den Simulator interpretierbare Repräsentation des Prozesses. Zur Abbildung im Simulator werden im Wesentlichen die durch den Geschäftsprozess definierten Aktivitäten nummeriert; die Aktivitäten, die einen Dienstaufruf nach sich ziehen, sind grau hinterlegt.

8.2.2 Konfiguration des Simulators

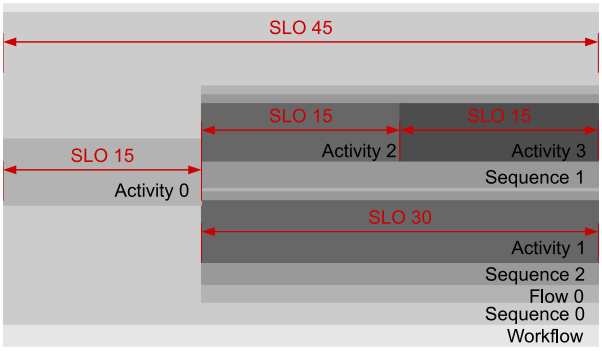

Abbildung 8.2: Initiale SLO-Zuordnung im Überweisungsprozess

Abbildung 8.2 zeigt die initial vorgegebenen Antwortzeit-SLOs für den Geschäftsprozess und die darin enthaltenen Aktivitäten. Für den Gesamtprozess ist eine Antwortzeit von 45 Zeiteinheiten festgelegt. Diese wird wie in Abschnitt 6.7 beschrieben auf die im kritischen Ausführungspfad definierten Dienste heruntergebrochen. Somit ergibt sich für diese Dienste jeweils ein SLO von 15 Zeiteinheiten. Für die parallel ablaufende und in der Abbildung mit `Activity 1` bezeichnete Registrierung der Überweisungsgebühren stehen damit 30 Zeiteinheiten zur Verfügung.

	SLO	$t_{response}$ bei t_0	$t_{response}$ bei t_{1000}
`Activity 0`	15	15	10
`Activity 1`	30	30	35
`Activity 2`	15	15	15
`Activity 3`	15	15	20

Tabelle 8.4: Antwortzeitmodell der Dienste des Überweisungsprozesses

Für jeden Dienst wird nun im Simulator ein Antwortzeitmodell hinterlegt. Zu Beginn der Simulation sollen alle beteiligten Dienste jeweils die in ihrem Antwortzeit-SLO hinterlegte Zeitspanne zur Abarbeitung benötigen. Im Verlauf der Simulation verändert sich die Antwortzeit einzelner Dienste wie in Tabelle 8.4 dargestellt.

Nach dem Start der Simulation organisieren sich die den einzelnen Aktivitäten zugeordneten QoS-Manager in lokalen Kommunikationsgruppen. Im Beispiel werden zunächst drei lokale Kommunikationsgruppen etabliert:

Gruppe 1 : Activity 0

Gruppe 2 : Activity 1

Gruppe 3 : Activity 2, Activity 3

Jede Gruppe bestimmt einen Koordinator, der eine Proxy-Rolle instanziiert und zusätzlich der nächstübergeordneten Gruppe beitritt. Die Koordinatoren der Gruppen 2 und 3 erzeugen eine weitere Kommunikationsgruppe (Nr. 4), der Koordinator dieser Gruppe tritt Gruppe 1 bei.

Zum Start der Simulation existieren die folgenden Kommunikationsgruppen im System:

Gruppe 1 : Activity 0, Koordinator 1, Koordinator 4

Gruppe 2 : Activity 1, Koordinator 2

Gruppe 3 : Activity 2, Activity 3, Koordinator 3

Gruppe 4 : Koordinator 2, Koordinator 3, Koordinator 4

Die Kommunikation der Teilnehmer erfolgt wie in Abschnitt 7.2.2 beschrieben über JGroups Channels.

8.2.3 Ablauf der Simulation

In Teil (1) von Abbildung 8.3 ist der Prozess zu Beginn der Simulation dargestellt. Für jeden grau hinterlegten Dienstaufruf ist der aktuelle Δslo-Wert angegeben, dieser beträgt zu Beginn der Simulation für alle Dienste 0.

Teil (2) der Abbildung zeigt den Zustand der einzelnen Dienste nach Erreichen des Simulationszeitpunkts t_{1000}. Entsprechend dem Antwortzeitmodell (vgl. Tabelle 8.4) verfügt Aktivität 0 nun über 5 ungenutzte Antwortzeit-SLO-Anteile, die Aktivitäten 1 und 3 weisen dagegen ein Δslo von -5 auf. SLO und Antwortzeitverhalten von Aktivität 2 sind unverändert.

Aktivität 0 agiert als Auktionator und offeriert – wie in Abbildung 8.3, Teil (3) dargestellt – die überschüssigen Antwortzeit-SLO-Anteile den anderen Mitgliedern der lokalen Gruppe.

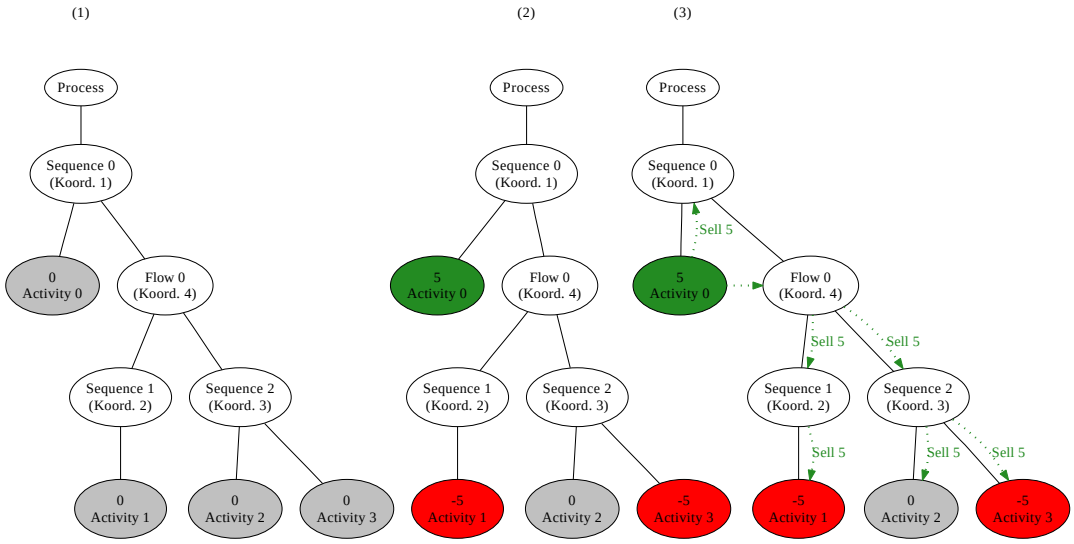

Abbildung 8.3: Zustand der Simulation bei t_0 (1) und nach t_{1000} ((2) und (3))

Innerhalb dieser lässt sich allerdings kein SLO-Ausgleich erreichen. Daher agiert das Gruppenmitglied Koordinator 4 nun seinerseits als Auktionator und bietet den Mitgliedern der untergeordneten Gruppe 4 die Antwortzeit-SLO-Anteile an. Diese sind ebenfalls Koordinatoren und erzeugen entsprechende Auktionen in den Gruppen 2 und 3.

In den Gruppen 2 und 3 befinden sich jeweils Aktivitäten mit negativem Δslo, diese geben nun Gebote für die ihnen angekündigte Auktion ab (vgl. Abbildung 8.4, Teil (4)). Aus Sicht der bietenden Aktivitäten 1 und 3 handelt es sich dabei um lokale Auktionen, die SLO-Anteile werden durch den jeweiligen Gruppenkoordinator angeboten. Die Koordinatoren 2 und 3 geben nun – im Rahmen ihrer Rollen als Sequenz-Proxy (vgl. Abschnitt 6.5.4.1) – als Weiterleitung Gebote für die Auktion in der übergeordneten Gruppe 4 ab. Da der Koordinator der Gruppe 4 einen Proxy für eine Parallelausführung implementiert, leitet er für die übergeordnete Auktion keine Gebote unterschiedlicher Kindelemente weiter (siehe Teil (5) der Abbildung): Er wählt aus den eingegangenen Geboten ein Kindelement aus und leitet dessen Gebot an den Auktionator weiter, die anderen Kindelemente erhalten zunächst eine „Auktion verloren“-Nachricht. Auf diese Weise wird die Koordination der Antwortzeit-SLOs der untergeordneten Teilpfade sichergestellt und verhindert, dass gleichzeitig mehrere Teilpfade einer Parallelausführung oder Fallunterscheidung von einem übergeordneten Dienst Antwortzeit-SLO-Anteile erwerben (was eine Änderung des Gesamt-SLOs zur Folge haben könnte).

In Teil (6) von Abbildung 8.4 vergibt der von Aktivität 0 instanziierte Auktionator den Zuschlag an den einzigen Bieter. Die Zuschlagsnachricht wird wieder durch die Koordinatoren an den ursprünglichen Bieter, Aktivität 1, weitergeleitet. Koordinator 4 registriert den Zuschlag für ein untergeordnetes Element und errechnet für die durch Koordinator 3 repräsentierte Sequenz 2 einen Δp-Wert von 5 (vgl. Abschnitt 6.4.7). Der Koordinator instanziiert als Reaktion

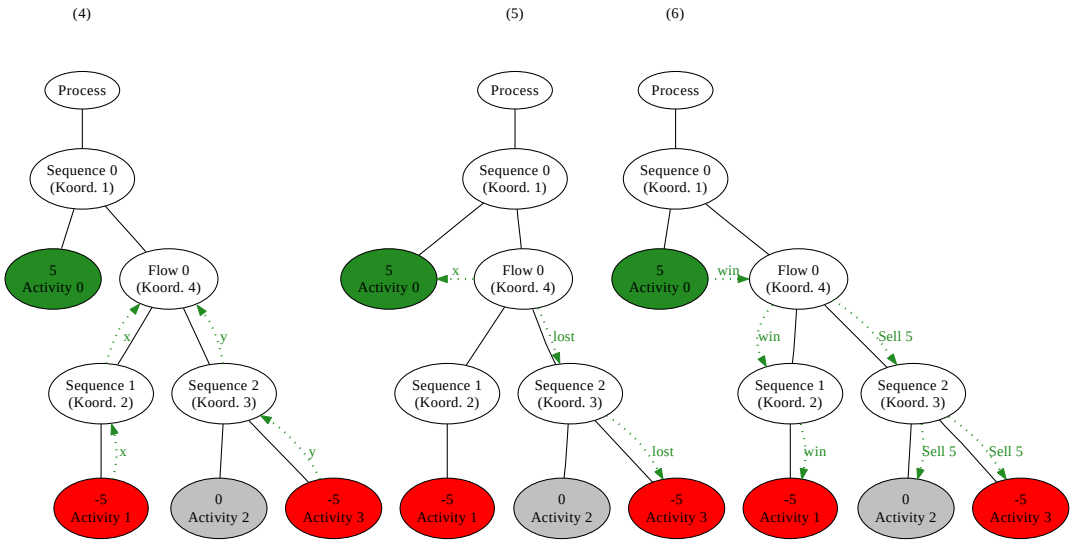

Abbildung 8.4: Abwicklung einer Auktion über Gruppengrenzen hinweg

darauf für Sequenz 2 einen Proxy-Auktionator, der 5 Antwortzeit-SLO-Anteile anbietet.

Somit agiert Koordinator 4 als Auktionator gegenüber Koordinator 3, der die Auktionsankündigung seinerseits an die Mitglieder von Gruppe 3 weiterleitet. Nun kann Aktivität 3 erneut ein Gebot abgeben (vgl. Abbildung 8.5, Teil(7)) und erhält diesmal den Zuschlag (Teil (8)). Im Anschluss an diese Auktion sind die Δslo-Werte der am Prozess beteiligten Aktivitäten wieder ausgeglichen (Teil (9)).

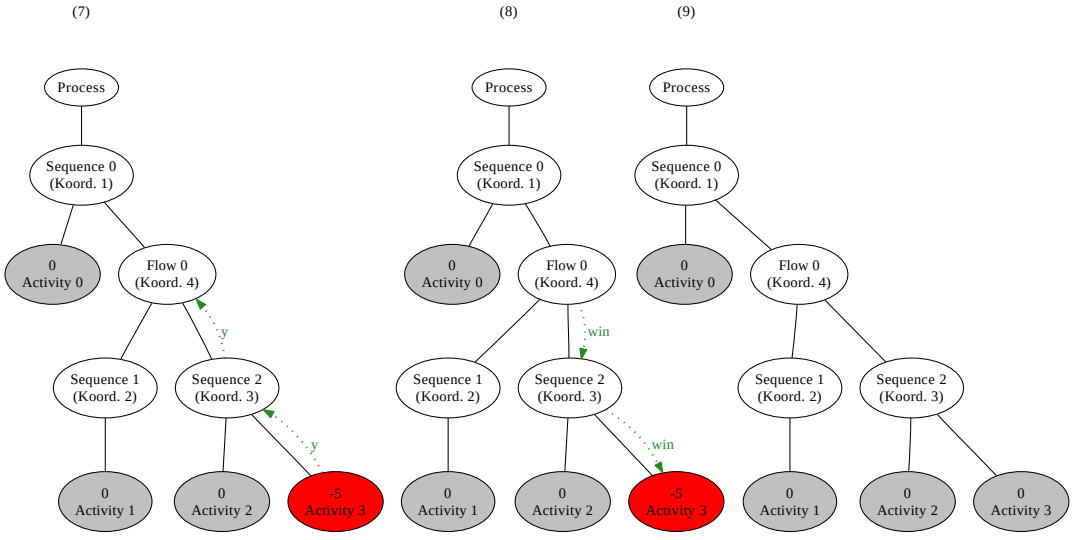

Abbildung 8.5: Ausgleich des Δslo des zweiten Teilpfades durch eine Proxy-Auktion

8.2.4 Betrachtung der Kommunikation

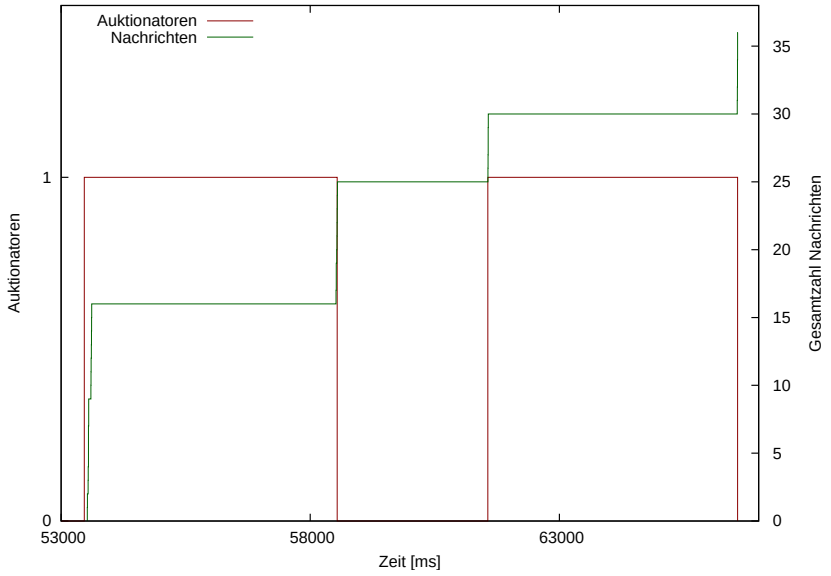

Abbildung 8.6: Ablauf der Simulation, Übersicht über versandte Nachrichten

Abbildung 8.6 visualisiert den Ablauf der in Abschnitt 8.2.3 beschriebenen Auktionen und zeigt die Anzahl der im zeitlichen Verlauf der Auktionen versandten Nachrichten. Multicast-Nachrichten sind dabei auf n Einzelnachrichten abgebildet. Der Absender einer Multicast-Nachricht bekommt diese als Mitglied der adressierten Gruppe ebenfalls zugestellt. In der Grafik sind lediglich die zeitlichen Verläufe der initialen Auktionen dargestellt, in der Folge ausgelöste Proxy-Auktionen wurden nicht visualisiert. Das Diagramm zeigt insgesamt einen Zeitraum von etwa 14 Sekunden, die Bietfrist für eine Auktion ist auf 5 Sekunden festgelegt. Die Auktionator-Rolle ist vom Beginn einer Auktion bis zum Abschluss der Übertragung der versteigerten SLO-Anteile aktiv (vgl. Ablauf des Auktionsprotokolls in Abbildung 6.12).

Tabelle 8.5 gibt einen Überblick über die während der Simulation versandten Nachrichten und die instanziierten Proxy-Rollen. Insgesamt werden 36 Nachrichten versandt. Zur Ankündigung der beiden Auktionen über die beteiligten Kommunikationsgruppen hinweg werden 12 Nachrichten benötigt. Da sich nicht alle QoS-Manager an den Auktionen beteiligen, werden insgesamt 7 Gebotsnachrichten verschickt. Ebenso viele Nachrichten werden zur Bekanntgabe des Auktionsergebnisses benötigt. Zur Abarbeitung des in Abbildung 6.12 beschriebenen Acknowledgement-Protokolls werden insgesamt 10 Nachrichten ausgetauscht.

Im Weiteren wird untersucht, wie sich der Nachrichtenaufwand bei der Durchführung von Auktionen in größeren bzw. strukturell komplexeren Workflows entwickelt.

EREIGNIS	ANZAHL
Nachrichten gesamt	36
Auktionsankündigungsnachrichten	12
Gebotsnachrichten	7
Ergebnisnachrichten	7
Acknowledgement-Nachrichten	10
Abgegebene Gebote	3
erfolgreich	2
nicht erfolgreich	1
Instanziierte Proxy-Auktionatoren	5
für Parallelausführungen	2
für Sequenzen	3

Tabelle 8.5: Im Auktionsszenario übertragene Nachrichten und instanziierte Proxy-Rollen

8.3 Untersuchungen zur Skalierbarkeit

8.3.1 Vergleich von lokaler und verteilter Auktion

Zur Abschätzung der Auswirkung von Netzwerklatenzen auf den Ablauf einer Auktion wird im Folgenden ein einfaches Simulationsszenario zunächst lokal auf einer einzigen Maschine und im Anschluss verteilt auf vier Maschinen ausgeführt. Die dabei ermittelten Nachrichtenlaufzeiten beziehen sich auf das verwendete Gruppenkommunikationssystem JGroups in Version 2.7. Die im Folgenden dargestellten Zeiten wurden mit JDK 1.6 unter Linux auf durch Gigabit-Ethernet verbundenen PC-Rechnern vom Typ Intel Core2 Duo mit 1,86 GHz und 2 GB RAM ermittelt. Für die lokale Messung wurde ein Rechner mit konfiguriertem JGroups Loopback-Protokoll verwendet, die verteilte Messung nutzt das JGroups TCP-Protokoll für einen zuverlässigen Nachrichtentransport.

Das Simulationsszenario besteht aus einer Kommunikationsgruppe, der die QoS-Manager von vier sequenziell in einem Workflow angeordneten Diensten beitreten. Zwei Teilnehmer offerieren jeweils 5 Antwortzeit-SLO-Anteile, die anderen beiden Teilnehmer benötigen jeweils 5 Antwortzeit-SLO-Anteile. Die Dauer jeder Auktion beträgt 5 Sekunden.

8.3.1.1 Lokaler Fall

Abbildung 8.7 zeigt den Verlauf der lokalen Simulation. Nach Abschluss der sequenziellen Initialisierung der Kommunikationsadapter der beteiligten QoS-Manager starten zum Simulationszeitpunkt 30551 ms beide Auktionatoren und kündigen ihre Auktionen in der Kommu-

nikationsgruppe an. Die Nachrichtenlaufzeiten betragen im lokalen Fall zwischen 2 ms und 6 ms, zusätzliche Zeit wird allerdings für die Verarbeitung der Ereignisse im Simulator benötigt: Die versandte Auktionsankündigung wird nach 40 ms vom ersten Bieter behandelt, der zweite Bieter folgt 3 ms später. Die jeweiligen Gebote liegen weitere 6 ms später beim Auktionator vor. Im dargestellten Beispiel entscheiden sich beide Bieter, Gebote für die gleiche Auktion abzugeben.

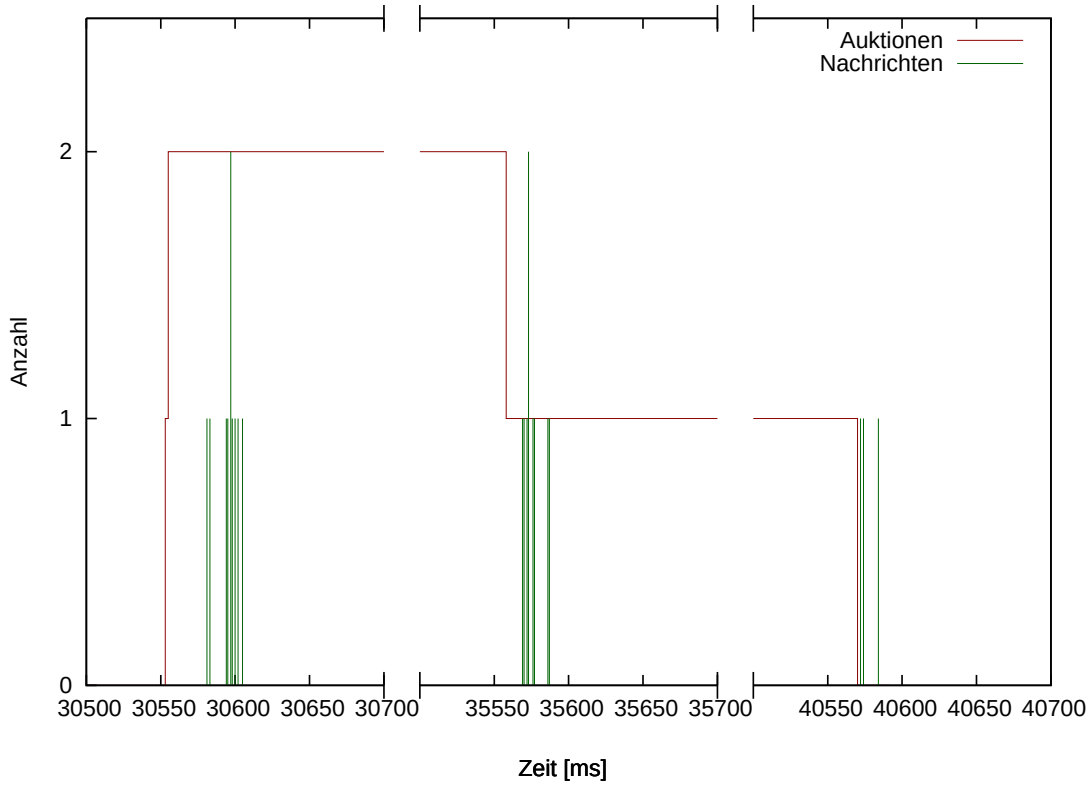

Abbildung 8.7: Zeitlicher Verlauf einer lokalen Simulation

Nach Eingang der Gebote wartet der Auktionator das Ende der Bietfrist ab, bevor die Dienste über den Ausgang der Auktion informiert werden. Der zweite Auktionator startet parallel eine neue Auktion, an der sich nun der Verlierer der ersten Auktion beteiligt. Hier geht lediglich das Gebot dieses Dienstes ein, er erhält nach Ablauf der Bietfrist den Zuschlag. Insgesamt werden in dem dargestellten Szenario 22 Nachrichten versandt. Die insgesamt zum Erreichen eines ausgeglichenen Zustands benötigte Zeit beträgt knapp 11 Sekunden.

8.3.1.2 Verteilter Fall

Abbildung 8.8 zeigt beispielhaft den Ablauf der verteilten Abarbeitung des selben Simulationsszenarios. Hierbei residiert jeder der beteiligten QoS-Manager auf einer separaten Maschine. In der verteilten Simulation erfolgt die Initialisierung der einzelnen Dienste unabhängig voneinander und ohne zentrale Kontrollinstanz. Auf Ebene der JGroups-Kommunikation variiert die Initialisierungszeit z. B. in Abhängigkeit davon, ob eine neue Kommunikationsgruppe erzeugt oder einer existierenden Gruppe beigetreten wird. Aus diesem Grund starten Auktionatoren und Bieter nicht wie im lokalen Fall zur gleichen Zeit.

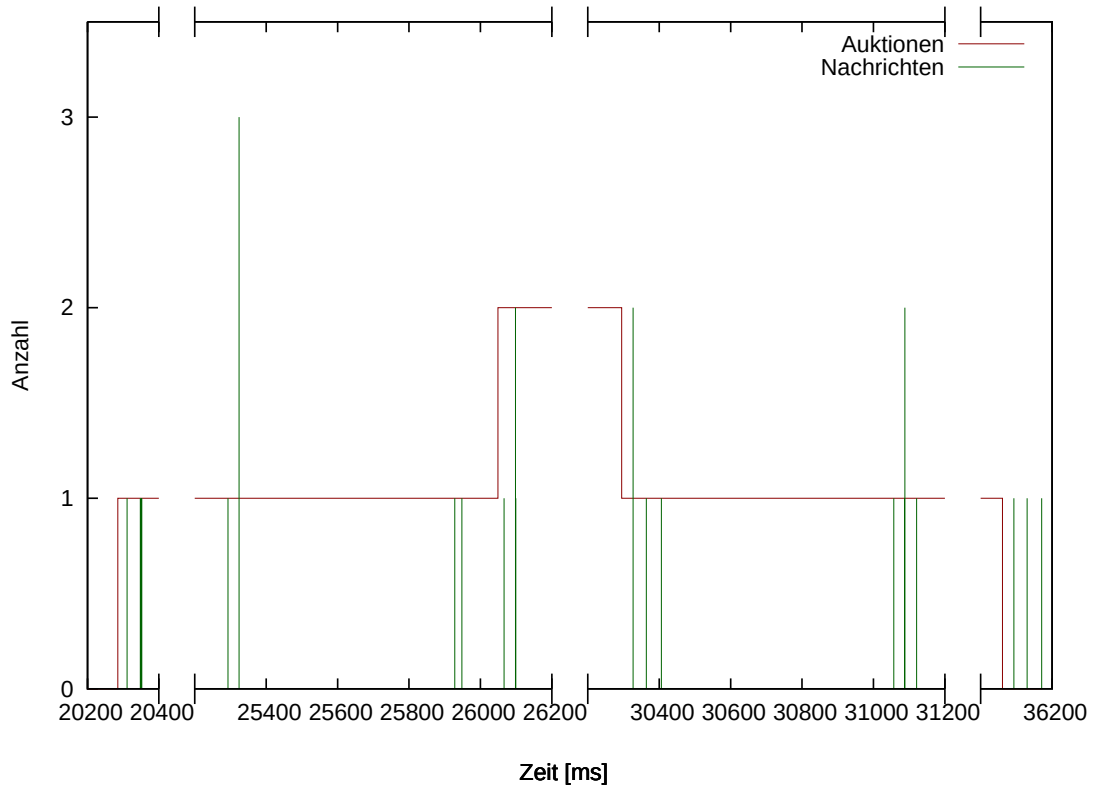

Abbildung 8.8: Zeitlicher Verlauf einer verteilten Simulation

Zum Zeitpunkt der ersten Auktionsankündigung sind die potenziellen Bieter noch nicht vollständig initialisiert, sodass die erste Auktion ohne Gebote beendet wird. Zum Simulationszeitpunkt 25290 ms kündigt der erste Auktionator eine Folgeauktion an. Diese erreicht den ersten Bieter 34 ms später. Beide Bieter geben nun für diese wiederholte Auktion Gebote ab. Die einzelnen Gebotsnachrichten benötigen 32 ms bzw. 58 ms bis zur Verarbeitung durch den Auktionator. Zum Simulationszeitpunkt 26049 ms kündigt der zweite Auktionator eine Auktion an, zu der keine Gebote eingehen. Nach Abschluss der Auktion des ersten Auktionators

bietet der Verlierer auf die Wiederholung dieser Auktion, sodass nach ca. 16 Sekunden ein ausgeglichener Zustand erreicht ist. Insgesamt werden in dem dargestellten Experiment 26 Nachrichten versandt.

8.3.1.3 Fazit

Die Zeitdauer bis zum Erreichen eines ausgeglichenen Zustands hängt im Wesentlichen von der Bietfrist für eine Auktion ab. Bei der für die Simulationen gewählten Dauer von 5 Sekunden ist die für die Nachrichtenübertragung benötigte Zeitdauer sowohl im lokalen als auch im verteilten Fall zu vernachlässigen. Für den Fall, dass Dienste über schmalbandige Datenverbindungen kooperieren, kann es evtl. notwendig sein, eine längere Auktionsdauer zu wählen, im lokalen Netzwerk sind deutliche kürzere Auktionsdauern denkbar.

Der aufgetretene Unterschied in der Dauer bis zum Erreichen eines ausgeglichenen Zustands ist durch die unglückliche Initialisierung des Systems im verteilten Fall entstanden. Daraus kann keine generelle Schlussfolgerung über Systemeinschwingzeiten abgeleitet werden. Das Beispiel zeigt aber, dass die zur Initialisierung des Kommunikationssystems notwendige Zeit nicht unterschätzt werden darf, da diese ggf. auch bei Restrukturierungen auf Geschäftsebene erneut anfallen kann.

8.3.2 Ein Auktionator, mehrere Bieter

8.3.2.1 Experimentaufbau

Im Folgenden wird untersucht, wie sich die Anzahl der Bieter einer Auktion auf die Nachrichtenanzahl auswirkt. Zu diesem Zweck wird in Simulationsläufen mit $\{2, 4, 8, 16, 32, 64, 96, 128\}$ sequenziell angeordneten Aktivitäten jeweils eine Einzelauktion durchgeführt. Eine der Aktivitäten tritt dabei als Auktionator auf, die anderen als Bieter. Das Szenario ist so aufgebaut, dass alle Bieter durch das Angebot des Auktionators befriedigt werden können. Die in den Simulationsläufen zugrunde gelegte Workflow-Struktur ist in Abbildung 8.9 dargestellt. In diesem Szenario kommunizieren alle Aktivitäten direkt miteinander, es wird für die gesamte Sequenz nur eine einzige Kommunikationsgruppe etabliert.

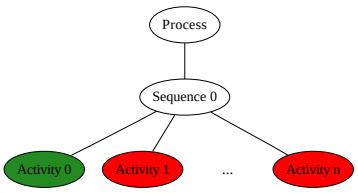

Abbildung 8.9: Testszenario mit einer Kommunikationsgruppe und einer Auktion

Zur Untersuchung der Auswirkungen komplexerer Workflow-Strukturen und damit Proxy-gestützter Kommunikation auf die Anzahl der zur Abwicklung einer Auktion notwendigen Nachrichten wird ein weiteres Experiment durchgeführt: Das oben besprochene Testszenario wird so modifiziert, dass die Bieter in jeweils 2 parallele Workflow-Sequenzen unterteilt werden. Die Aufteilung in einen Auktionator und n Bieter wird beibehalten. Abbildung 8.10 zeigt den Aufbau des Experiments.

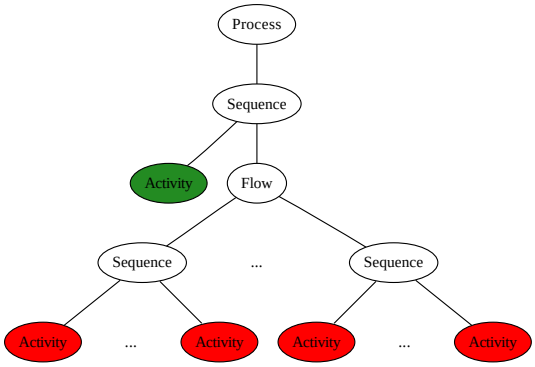

Abbildung 8.10: Testszenario mit mehreren Kommunikationsgruppen und einer Auktion

8.3.2.2 Auswertung

Sequenz Wie in Abschnitt 6.5.2.4 dargestellt, ist für eine Vickrey-Auktion mit n Teilnehmern mit einem Kommunikationsaufwand von $3n$ Nachrichten zu rechnen. Im Rahmen der Implementierung des Ansatzes wurde nach Abschluss der eigentlichen Auktion ein zusätzliches Nachrichtenpaar zwischen einem erfolgreichen Bieter und dem Auktionator zur Übertragung des zu zahlenden Betrages und der ersteigerten Antwortzeit-SLO-Anteile eingeführt (vgl. Abbildung 6.12). Insgesamt ist bei einer Auktion mit n erfolgreichen Bietern also mit einem Nachrichtenaufkommen von $5n$ Nachrichten zu rechnen.

Für jede Teilnehmeranzahl wurden 200 lokale Simulationsläufe von je einer Minute Dauer durchgeführt. Abbildung 8.11 zeigt das dabei ermittelte Nachrichtenaufkommen und die Anzahl der abgegebenen Gebote. Zusätzlich zum Mittelwert ist für jeden Messpunkt die ermittelte Standardabweichung aufgetragen. Man erkennt, dass sich die Messreihen im Szenario wie erwartet entwickeln. Für Szenarien mit vielen Diensten ergibt sich eine gewisse geringe Standardabweichung dadurch, dass die Auktionsankündigung nicht in allen Simulationsläufen durch alle Bieter innerhalb der Bietfrist empfangen wurde bzw. Gebote zu spät abgegeben wurden und so in diesen Szenarien einige wenige potenzielle Bieter nicht berücksichtigt wurden.

Zum Vergleich mit den zu erwartenden Werten sind in Abbildung 8.11 zusätzlich die Funktionen $f(x) = 5x$ und $g(x) = x$ aufgetragen.

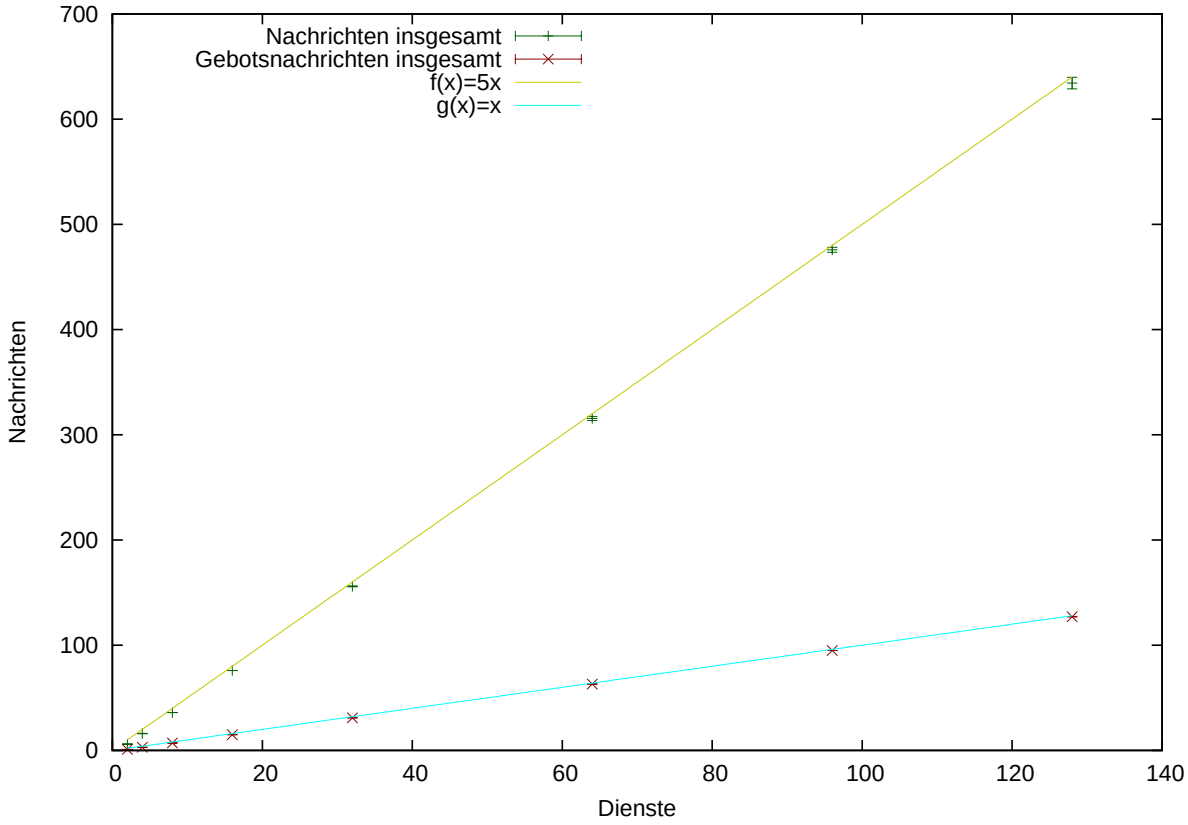

Abbildung 8.11: Nachrichtenaufkommen und Gebote einer Auktion mit einer Bietergruppe

Abbildung 8.12 zeigt den zeitlichen Verlauf eines der Simulationsläufe mit 128 beteiligten Diensten. Die dargestellten Zeiten wurden mit JDK 1.6 unter Linux auf einem PC mit 1,86 GHz Intel Core2 Duo Prozessor und 2 GB RAM mit konfiguriertem JGroups Loopback-Protokoll ermittelt.

In der Grafik sind die zu einem Zeitpunkt im System zum Versenden anstehenden Nachrichten und die derzeitige Nachfrage aller Bieter nach SLO-Anteilen dargestellt. Kurz vor Erreichen des Zeitpunkts t=46000 ms erkennen die QoS-Manager der einzelnen Dienste zusätzlichen Antwortzeit-SLO-Bedarf, die Nachfrage steigt auf insgesamt 1270 Antwortzeit-SLO-Anteile (10 Anteile pro Bieter). Zum Zeitpunkt t=46015 ms wird der Auktionator aktiv. Der Versand der Auktionsankündigung verursacht die erste Spitze der im Diagramm aufgetragenen Nachrichten. Nach Empfang der Auktionsankündigung versenden die einzelnen Bieter ihre Gebotsnachrichten. Die letzte Gebotsnachricht geht zum Zeitpunkt t=46353 ms beim Auktionator ein. Von der Auktionsankündigung bis zum Eingang des letzten Gebots sind somit 338 ms vergangen. Die Auktionsdauer ist im Testszenario auf 5 Sekunden konfiguriert, sodass der Auktionator Ergebnisnachrichten erst nach Ablauf dieser Zeitspanne an die einzelnen Dienste verteilt. Nach Erhalt der ersteigerten SLO-Anteile reduziert sich der weitere Bedarf auf 0.

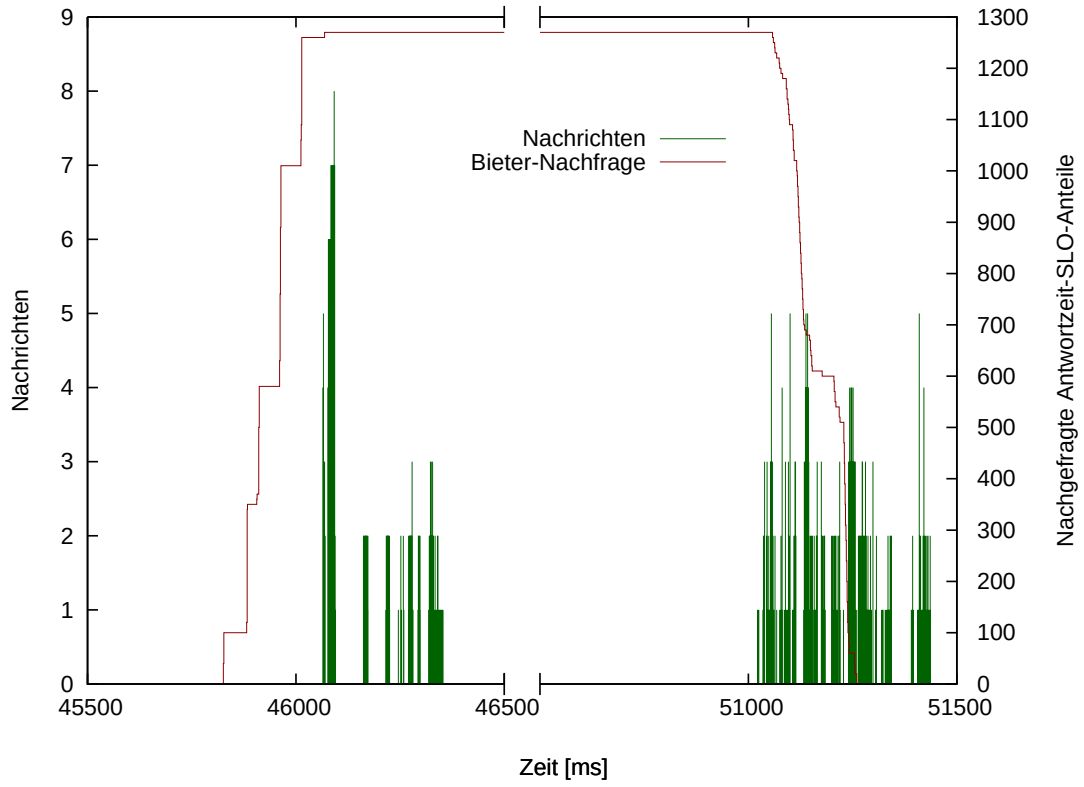

Abbildung 8.12: Zeitlicher Verlauf einer Auktion mit 128 Teilnehmern

Unterteilung in Parallelausführungen Im zweiten Teil der Simulation wird untersucht, auf welche Weise sich das Nachrichtenaufkommen verändert, wenn die Bieter nicht in einer einzigen Kommunikationsgruppe, sondern in mehreren Gruppen angeordnet sind, wie in Abbildung 8.10 gezeigt.

Betrachtet man den Aufbau des Testszenarios, so vermutet man zunächst intuitiv eine Verdreifachung des Nachrichtenaufkommens, da jeder der beiden zwischengeschalteten Proxys die ursprüngliche Nachrichtenanzahl durch Weiterleitung verdoppelt. Wie in Abschnitt 8.2.3 gezeigt, blockiert der für die Parallelausführung verantwortliche Proxy jedoch zunächst alle Gebote mit Ausnahme derer aus einem ausgewählten Teilpfad. Im Anschluss an erfolgreiche SLO-Transfers zu Aktivitäten in diesem Teilpfad agiert der Proxy seinerseits als Auktionator für die anderen ihm untergeordneten Teilpfade. Bei diesen Auktionen ist dann nur noch ein Proxy zwischengeschaltet, sodass sich für $m = 2$ Teilpfade ein Nachrichtenaufkommen von etwa $12,5\ n$ Nachrichten ergibt.

Für n Dienste in m parallelen Gruppen liegt die Zahl der durch den Proxy abgelehnten Gebote bei $\frac{m-1}{m}n$, für die in Abbildung 8.13 dargestellten Werte aus $m = 2$ Gruppen somit bei $\frac{1}{2}n$.

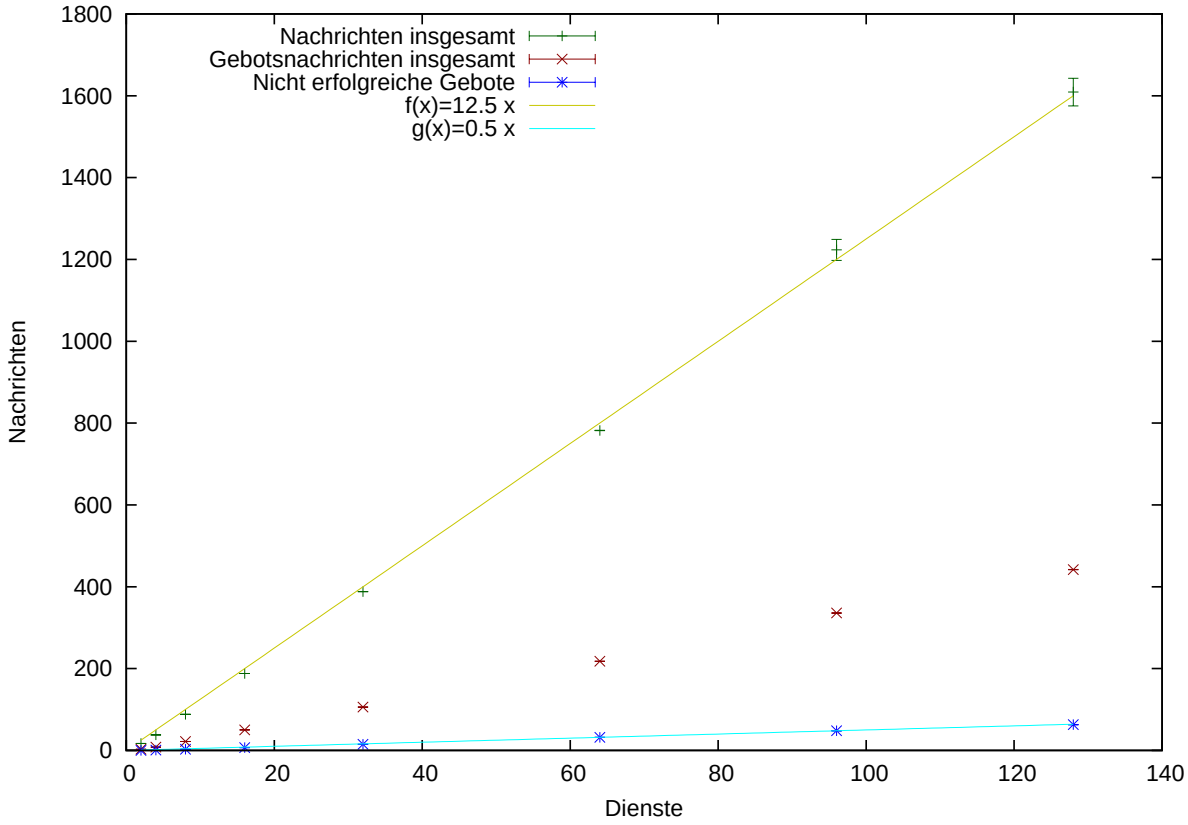

Abbildung 8.13: Nachrichtenaufkommen und abgelehnte Gebote bei einer Auktion mit zwei Bietergruppen

Zum Vergleich mit den zu erwartenden Werten sind in Abbildung 8.13 zusätzlich die Funktionen $f(x) = 12,5\ x$ und $g(x) = 0,5\ x$ gepunktet aufgetragen.

8.3.3 Mehrere Auktionatoren, mehrere Bieter

In den bisherigen Experimenten wurden ausschließlich Szenarien mit einem Auktionator betrachtet. Im Folgenden werden die Ergebnisse einiger Simulationsszenarien mit mehreren parallelen Auktionen vorgestellt.

8.3.3.1 Experimentaufbau

Zur Untersuchung des Nachrichtenaufkommens bei a parallelen Auktionen werden in jeweils 200 Simulationsläufen à 2 Minuten Dauer mit $n = \{2, 4, 8, 16, 32, 64, 128\}$ sequenziell angeordneten Aktivitäten zufällig rund $a = \frac{n}{2}$ Auktionatoren instanziiert. Die in den Simula-

tionsläufen zugrunde gelegte Workflow-Struktur ist in Abbildung 8.14 dargestellt. In diesem Szenario kommunizieren alle Aktivitäten direkt miteinander, es wird für die Sequenz nur eine einzige Kommunikationsgruppe etabliert.

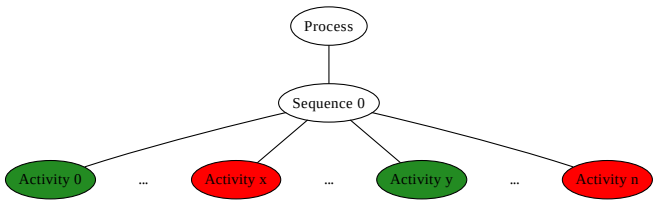

Abbildung 8.14: Szenario mit a parallelen Auktionen in einer Bietergruppe

Implementiert wird das Szenario im Simulator wie folgt: Zu einem definierten Zeitpunkt der Simulation verändert jeder Dienst seine Antwortzeit zufällig auf einen beliebigen Wert innerhalb des Intervalls $\Delta slo = [-10, 10]$ um das bisherige SLO bzw. die bisherige Antwortzeit. Damit nimmt ab diesem Zeitpunkt eine zufällige Anzahl von Aktivitäten Auktionator- bzw. Bieterrollen ein. Zudem unterscheiden sich die ablaufenden Auktionen in der Anzahl der angebotenen Antwortzeit-SLO-Anteile. Damit entstehen in unterschiedlichen Simulationsläufen sowohl Situationen, in denen ein SLO-Ausgleich möglich ist, da das Angebot größer oder gleich der Nachfrage ist, als auch Situationen, in denen die Nachfrage größer als das Angebot ist und somit insgesamt kein Ausgleich erreicht werden kann. Ein Simulationslauf endet, wenn entweder Angebot oder Nachfrage auf 0 sinken (und damit ein stabiler Zustand erreicht wird), oder die maximale Simulationsdauer von 2 Minuten überschritten wird.

8.3.3.2 Auswertung

Abbildung 8.15 zeigt, aufsteigend angeordnet, die Dauer bis zum Erreichen eines stabilen Zustands für $n = \{2, 4, 8, 16, 32, 64, 128\}$ Dienste. Die Verteilung der Zeiten zeigt den Einfluss der 5-sekündigen Bietfrist auf die Gesamtdauer der einzelnen Experimentläufe.

Für kleine n zeigt das Diagramm, dass ein Teil der Experimente ohne Durchführung von Auktionsrunden direkt in einen stabilen Zustand mündet. In diesen Fällen wurde Δslo durch die einzelnen Dienste so gewählt, dass alle Dienste sich entweder passiv verhalten (gewähltes $\Delta slo = 0$) oder im Rahmen der Δslo-Änderung einheitlich entweder nur Auktionatoren oder Bieter entstehen. Für $n = 2$ ergibt sich diese Situation in 111 Durchläufen, für $n = 4$ in 29 Fällen und für $n = 8$ Dienste noch in 3 Fällen.

Für $n = 128$ stellt das Diagramm die Gesamtdauer nur für 180 der 200 durchgeführten Simulationsläufe dar. Die übrigen 20 Experimente konnten bis zum Abbruch nach 2 Minuten (entsprechend 23 Gebotsrunden) noch keinen stabilen Zustand erreichen. In diesen Fällen ist prinzipiell das Erreichen eines stabilen Zustands möglich, hierfür wären aber weitere Gebotsrunden notwendig. Weiterhin erkennt man in Szenarien mit vielen Diensten eine erhöhte

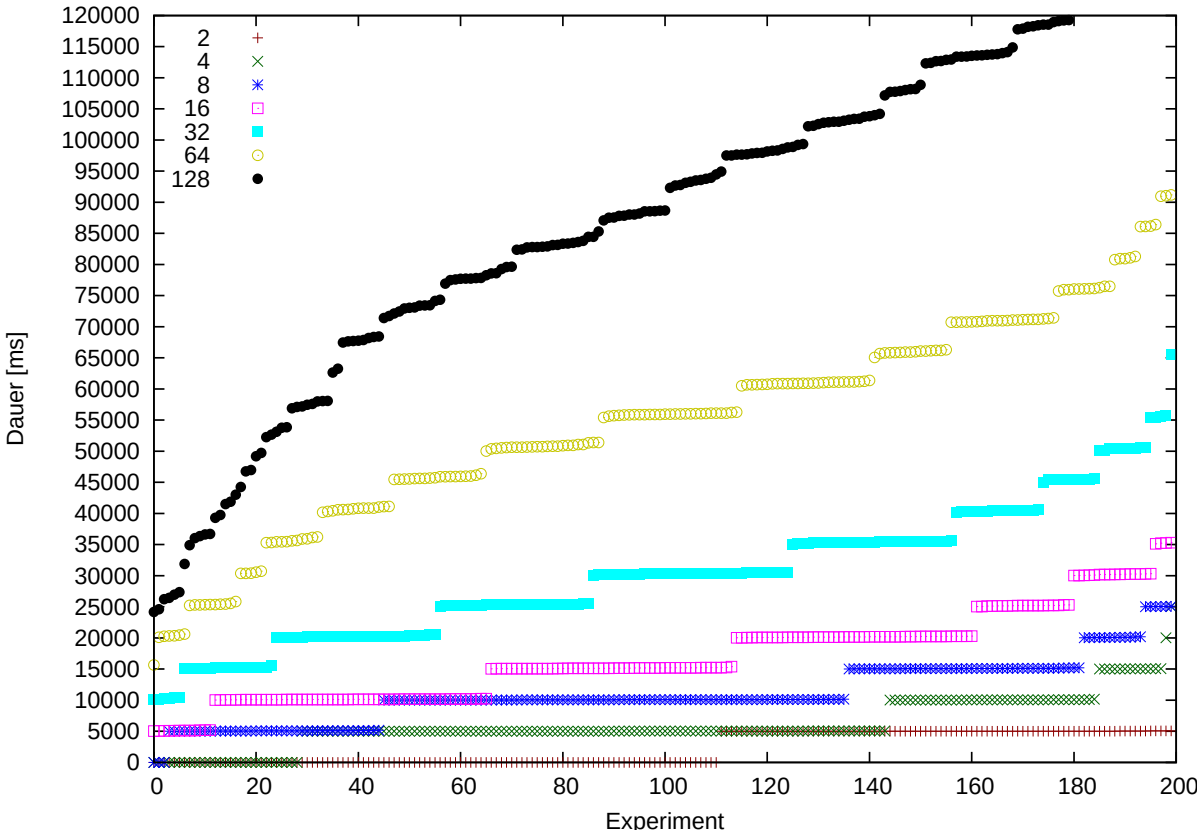

Abbildung 8.15: Zum Erreichen eines stabilen Zustands benötigte Zeitdauer bei unterschiedlicher Anzahl Dienste (aufsteigend sortiert nach Experimentdauer)

Varianz in der Experimentdauer. Diese ist auf die in diesen Fällen vorhandene hohe Anzahl parallel abzuarbeitender Threads und die durch Versand und Empfang großer Nachrichtenmengen erzeugte Systemlast zurückzuführen.

Die aus Abbildung 8.15 gewonnenen Erkenntnisse erlauben somit eine Abschätzung der Zeit (in Auktionsrunden), die für das Erreichen eines stabilen Zustands in unterschiedlich großen Workflows benötigt wird. Abbildung 8.16 zeigt für unterschiedliche Werte von n den Prozentsatz der Experimente, die nach einer bestimmten Anzahl von Auktionsrunden in einen stabilen Zustand gebracht werden konnten.

Für $n = \{2, 4, 8, 16, 32, 64\}$ lässt sich aus diesen Verteilungsfunktionen die zu erwartende Anzahl von Auktionsrunden bis zum Erreichen eines stabilen Zustands ablesen. Diese sind in Tabelle 8.6 dargestellt. Für $n = 128$ ist kein exakter Mittelwert angegeben, weil ein Teil der Experimente vor Erreichen des stabilen Zustands abgebrochen wurde.

Neben der Anzahl der zu erwartenden Auktionsrunden ist die Anzahl der zum Erreichen des stabilen Zustands zu versendenden Nachrichten von Interesse, da sich hieraus ableiten lässt,

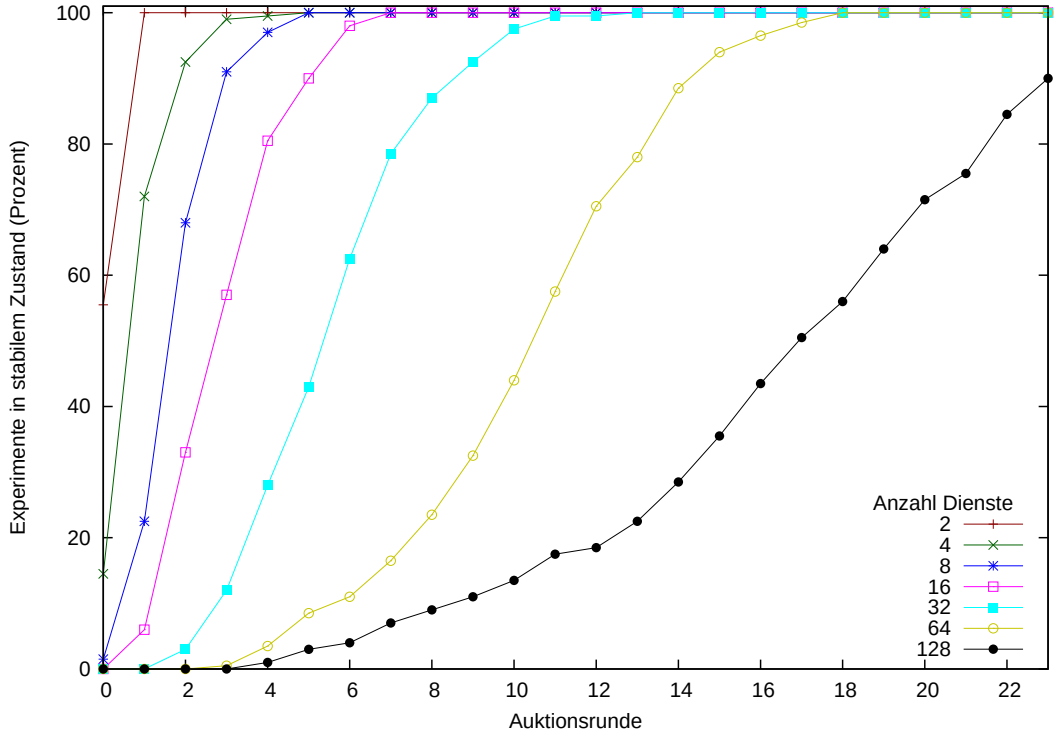

Abbildung 8.16: Prozentsatz von Experimenten in stabilem Zustand zu einer bestimmten Auktionsrunde

	Dienste						
Auktionsrunden	2	4	8	16	32	64	128
ermitteltes Maximum	1	5	5	7	13	18	23
Mittelwert	0,4	1,2	2,2	3,4	6,0	10,8	> 14,5

Tabelle 8.6: Auswertung der Messungen: Zu erwartende Anzahl von Auktionsrunden

wie ressourcenintensiv sich das Koordinationsverfahren verhält. Im Folgenden wird beispielhaft für $n = \{16, 128\}$ untersucht, ob ein Zusammenhang zwischen der Anzahl der in einem Experiment versandten Nachrichten und der Anzahl der durchgeführten Auktionsrunden besteht. Bei der Betrachtung der Nachrichtenanzahl werden Multicast-Nachrichten wieder auf n Unicast-Nachrichten abgebildet (vgl. Abschnitt 6.5.2.4).

Abbildung 8.17 zeigt für $n = 16$ beteiligte Dienste die Anzahl der in einem Experimentlauf versandten Nachrichten. Hierbei sind nur die Nachrichten dargestellt, die bis zum Eintreten eines stabiles Zustands versandt wurden. Existiert im Experiment ein Überangebot an Antwortzeit-SLO-Anteilen, so werden nach Befriedigung aller Bieter erfolgte Auktionsankündigungen nicht betrachtet. Man erkennt, dass zusätzliche Auktionsrunden tendenziell eine

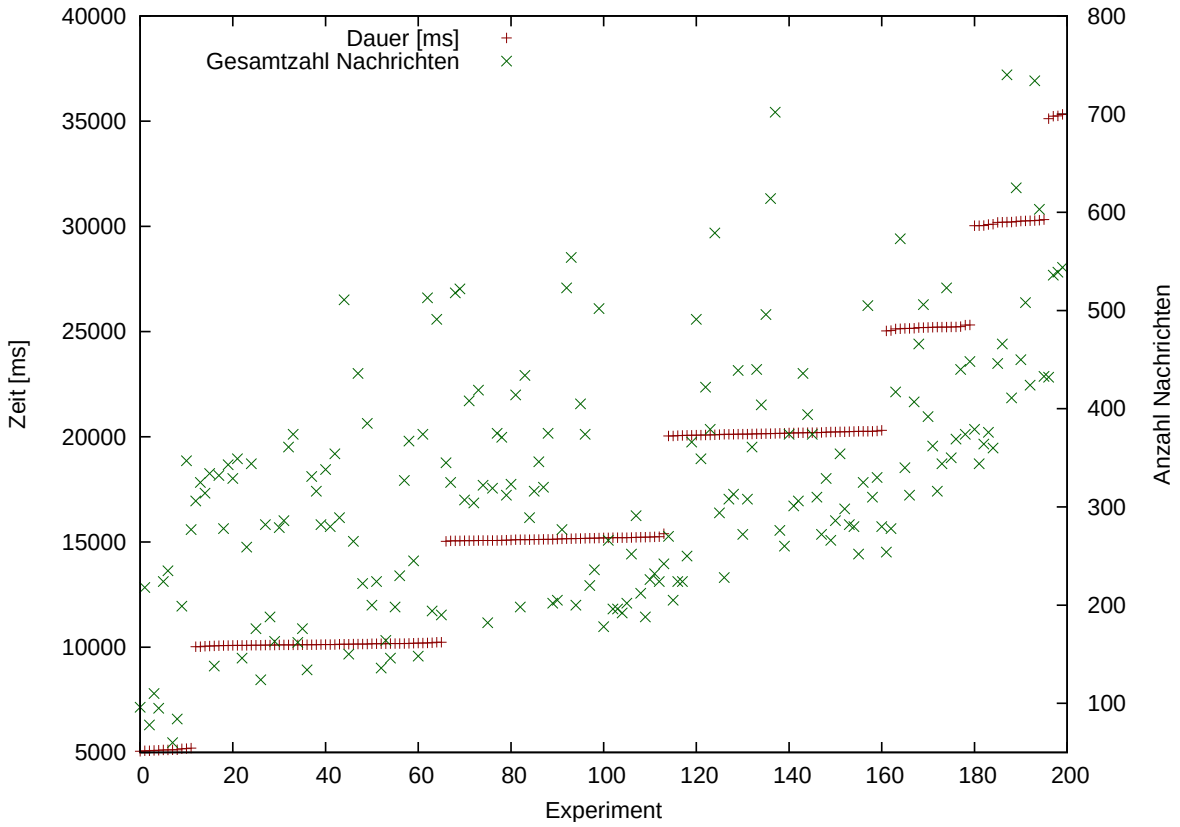

Abbildung 8.17: Zusammenhang zwischen Nachrichtenanzahl und Dauer eines Experiments für $n = 16$

größere Nachrichtenanzahl zur Folge haben.

Die Unterschiede in der Nachrichtenanzahl lassen sich auf folgende Faktoren zurückführen:

- *Auswahl einer Auktion*
  Aus a gleichzeitig angekündigten Auktionen wählt der Bieter zufällig eine Auktion aus, für die er ein Gebot abgibt. So kann es passieren, dass für einige Auktionen keine Gebote abgegeben werden, während für andere Auktionen mehr Nachfrage als Angebot besteht.
- *Wiederholung einer Auktion*
  Gehen für eine Auktion innerhalb der Bietfrist keine Gebote ein, so wiederholt der Auktionator die Auktion. Ist die durch die Bieter nachgefragte Menge geringer als das Angebot des Auktionators, wird der übrigbleibende Teil des Angebots in einer Folgeauktion erneut angeboten. Beides resultiert in weiteren n zu versendenden Auktionsankündigungsnachrichten.

Für $n = 16$ Dienste ergeben sich im Experiment die in Tabelle 8.7 dargestellten mittleren Nachrichtenzahlen.

$n = 16$	Benötigte Auktionsrunden 1	2	3	4	5	6	7
Experimente	12	54	48	47	19	16	4
$\bar{z}$	164,1	274,4	316,0	347,3	393,2	482,0	512,8
σ_z	96,5	102,5	103,8	103,4	84,1	132,7	53,9

Tabelle 8.7: Mittlere zum Erreichen eines stabilen Zustands nötige Nachrichtenanzahl $\bar{z}$

Abbildung 8.18 zeigt für $n = 128$ beteiligte Dienste die Anzahl der in einem Experimentlauf versandten Nachrichten. Wiederum sind nur die Nachrichten dargestellt, deren Versand zum Erreichen eines stabilen Zustandes notwendig ist. Die Experimentläufe, die innerhalb von 2 Minuten keinen stabilen Zustand erreichten, sind im Diagramm nicht enthalten.

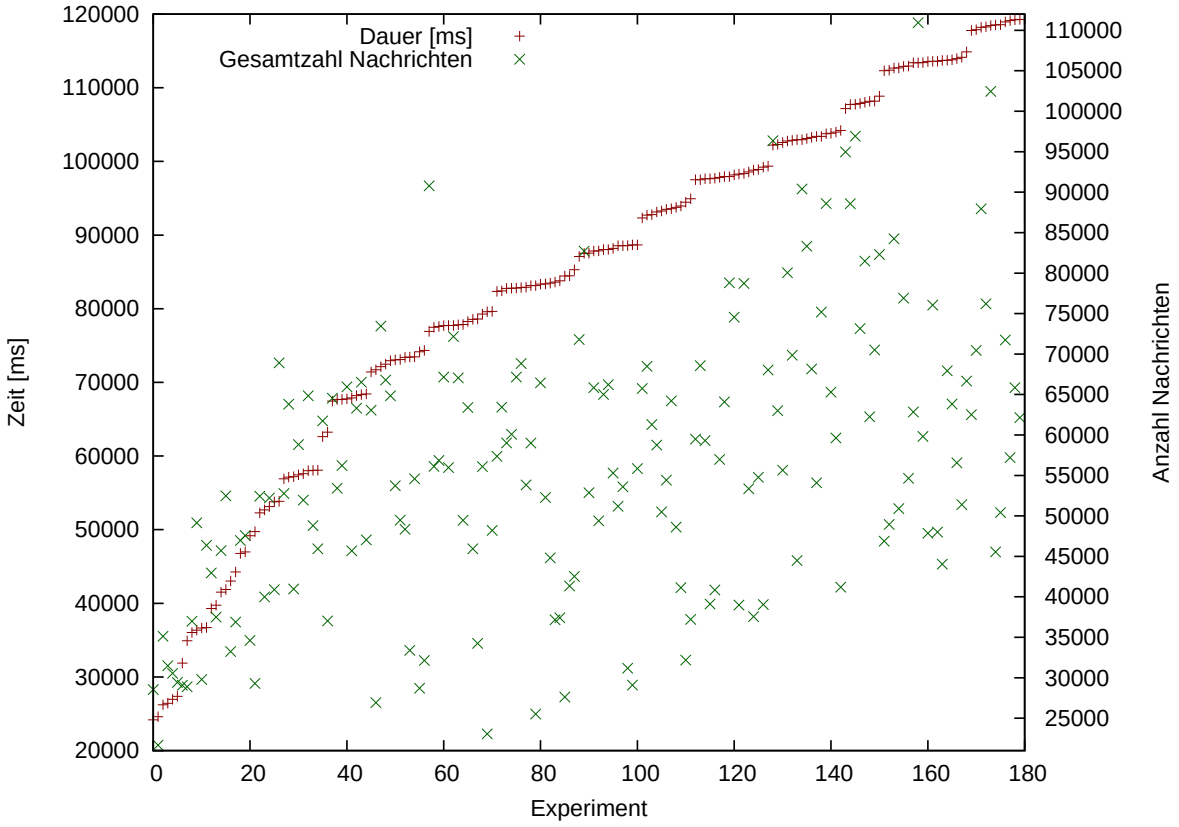

Abbildung 8.18: Zusammenhang zwischen Nachrichtenanzahl und Dauer eines Experiments für $n = 128$

Wie bereits für $n = 16$ diskutiert, bedeutet eine höhere Anzahl von Auktionsrunden tendenziell auch eine Erhöhung der Nachrichtenzahl. Tabelle 8.8 gibt einen Überblick über die mittlere

zum Erreichen eines stabilen Zustands notwendige Nachrichtenzahl.

Der zuletzt betrachtete Fall (128 kooperierende Dienste in einer Sequenz) dürfte in realen Systemen kaum anzutreffen sein. In produktiven SOA-Umgebungen findet man in der Regel relativ wenige Dienste in komplexen Workflow-Strukturen vor. Hier wird die Nachrichtenanzahl im Vergleich zur Betrachtung reiner Sequenzen durch die Verwendung der vorgestellten Proxy-Techniken reduziert (vgl. Abschnitt 8.3.2).

	Benötigte Auktionsrunden						
$n = 128$	4	5	6	7	8	9	10
Experimente	2	4	2	6	4	4	5
$\bar{z}$	25111,5	31640,5	28981,0	40452,2	42082,0	39613,0	50887,4
σ_z	4851,5	2468,8	141,4	7090,5	8688,1	9108,7	11702,0

	11	12	13	14	15	16	17
Experimente	8	2	8	12	14	16	14
$\bar{z}$	53485,5	49386,0	57834,8	49622,0	56177,6	51331,0	55184,8
σ_z	8472,3	17469,8	8471,4	16068,0	16490,8	13984,5	14783,2

	18	19	20	21	22	23	
Experimente	11	16	15	8	18	11	
$\bar{z}$	52953,1	57014,1	69016,9	81271,4	62136,2	68404,9	
σ_z	12167,3	14596,4	16624,4	12139,0	16824,5	16337,0	

Tabelle 8.8: Mittlere zum Erreichen eines stabilen Zustands nötige Nachrichtenanzahl $\bar{z}$

8.3.4 Fazit

Die hier präsentierten Ergebnisse stellen lediglich einen kleinen Ausschnitt der Aspekte dar, die mithilfe der Simulationsumgebung untersucht werden können.

Die durchgeführten Untersuchungen zur Skalierbarkeit zeigen, wie sich das Nachrichtenaufkommen im System für die Koordination einer großen Anzahl von Diensten entwickelt. Ein möglicher nächster Schritt in der Auswertung der vorliegenden Messdaten könnte eine getrennte Betrachtung der durchgeführten Experimente nach den Kriterien Überangebot bzw. Mangel an SLO-Anteilen sein, um den Zusammenhang von Nachrichtenzahl und Anzahl notwendiger Auktionsrunden genauer zu untersuchen.

8.4 Zusammenfassung

In Abschnitt 8.1 wurde der implementierte Prototyp im Hinblick auf die in Kapitel 5 definierten Anforderungen betrachtet. Im Anschluss wurde in Abschnitt 8.2 ein Beispiel-Workflow vorgestellt und dessen Ablauf im entwickelten Simulator detailliert untersucht.

In Abschnitt 8.3 wurden Ergebnisse von Untersuchungen zur Skalierbarkeit des Ansatzes vorgestellt. Zunächst wurden Nachrichtenlaufzeiten für lokale und verteilte Simulationsansätze untersucht. Die hier gezeigten Ergebnisse sind spezifisch für das Gruppenkommunikationssystem JGroups. Im Anschluss wurde untersucht, wie sich die Anzahl der zu versendenden Nachrichten für unterschiedlich große Systeme und unterschiedliche Systemstrukturen entwickelt, zum Abschluss wurde die Entwicklung des Nachrichtenaufkommens für eine Reihe zufälliger Experimente betrachtet.

Die Untersuchungen lassen vermuten, dass sich das in dieser Arbeit entwickelte auktionsbasierte Kooperationsverfahren prinzipiell auch für große Umgebungen mit vielen Diensten eignet. Das Kooperationsverfahren strebt stets das Erreichen eines stabilen Zustands an. Anhand der durchgeführten Experimente lässt sich abschätzen, innerhalb welcher Zeitspanne ein solcher Zustand für eine bestimmte Workflow-Struktur erreicht werden kann. Weiterhin erlauben die durchgeführten Experimente eine grobe Abschätzung des dabei zu erwartenden Nachrichtenaufwands.

9 Zusammenfassung und Ausblick

In SOA-basierten Geschäftsanwendungen stoßen aufgrund der vorhandenen Komplexität und Dynamik sowie der verteilten administrativen Zuständigkeiten traditionelle, hierarchische Ansätze für das Dienstgüte-Management schnell an ihre Grenzen. Zum SLM in SOA Workflows müssen temporär dienstübergreifende Kooperationen etabliert werden, die sich über mehrere administrative Domänen erstrecken können.

Ziel dieser Arbeit war die Entwicklung eines skalierbaren SLM-Ansatzes für das autonome Performance-Management in dezentralen dienstorientierten Architekturen.

9.1 Ergebnisse der Arbeit

Zunächst wurde eine allgemeine Einführung in das Anwendungsfeld der Arbeit gegeben: Der Aufbau dienstorientierter Architekturen wurde sowohl aus Geschäftssicht als auch aus technischer Sicht diskutiert. Im Rahmen dieser Diskussion wurden als Grundlage für die weitere Arbeit zentrale Begriffe definiert. Das Konzept der dienstorientierten Architektur ist unabhängig von konkreten Implementierungstechnologien. Daher wurde die Abbildung des Architekturkonzepts auf die Implementierungsebene exemplarisch anhand der Beschreibung ausgewählter Technologien zur Beschreibung von Workflows und Diensten vorgestellt (BPMN, BPEL, SCA).

Im Anschluss wurde ein Überblick über Begriffe und Technologien aus dem Bereich des IT-Managements gegeben. Nach einer allgemeinen Hinführung wurden Möglichkeiten zur Klassifizierung von IT-Management vorgestellt (FCAPS, ITIL), wobei insbesondere die Einordnung des Dienstgüte-Managements in die Gesamtthematik des IT-Managements im Vordergrund stand. Grundlage für das SLM sind Dienstgütedimensionen und in Bezug auf diese formulierte Dienstgüteanforderungen. Aufgrund ihrer Relevanz für das Verständnis der weiteren Arbeit wurde genauer auf die Dimensionen Bedien- und Antwortzeit eingegangen und Definitionen für Bedien- und Antwortzeit-SLOs gegeben. Exemplarisch wurden zwei Technologien aus dem Bereich des SLMs vorgestellt: Für das Monitoring verteilter Anwendungen im Hinblick auf Antwortzeitaspekte wurde das von der Open Group standardisierte ARM-API beschrieben. Als Basis für eine automatisierte Aushandlung von SLAs wurde die im Grid-Kontext spezifizierte – aber nicht auf diesen beschränkte – WS-Agreement-Spezifikation vorgestellt.

Aktuelles Forschungsgebiet im Bereich IT-Management ist das „Selbst-Management“, das – um der stetigen Komplexitätszunahme zu verwaltender IT-Systeme entgegenzuwirken – eine Reduktion der nach außen sichtbaren Komplexität durch Management-Automatisierung und Verlagerung von Management-Wissen in Software anstrebt. Um einen Überblick über die in diesem Bereich existierenden Ansätze zu geben, wurden unterschiedliche Architekturmodelle für selbstmanagende Systeme vorgestellt (IBM Autonomic Element, Motorola Autonomic Manager) und kurz auf die Problematik des Designs von geeigneten Kontrollalgorithmen eingegangen. Es wurden unterschiedliche Arbeiten zitiert, die sich mit der Klassifizierung selbstmanagender Systeme befassen, außerdem wurde ein Ansatz zum Design großer, dezentraler, selbstorganisierender Systeme mithilfe von geschachtelten Regelkreisen vorgestellt.

Zur Abrundung der IT-Management-Grundlagen wurde eine Reihe von Arbeiten vorgestellt, die sich mit Monitoring von Diensten und WfMSs oder mit Dienstgüte in SOA-Umgebungen befassen. Der hier entwickelte Ansatz lässt sich mit diesen Arbeiten integrieren.

Als Grundlage für den Entwurf des Dienstgüte-Management-Systems für SOA-Anwendungen wurde zunächst ein Anforderungskatalog definiert, der sich aus funktionalen und nichtfunktionalen Anforderungen sowie Integrationsanforderungen zusammensetzt. Basierend auf diesen Anforderungen wurde ein architekturelles Gesamtkonzept für QoS-Management-Komponenten und deren Interaktion mit den durch sie verwalteten Geschäftskomponenten entwickelt. Im Einzelnen besteht die entworfene Architektur aus „QoS-Managern“ genannten autonomen Management-Komponenten, die eine einheitliche Schnittstelle zur Aushandlung von SLAs zur Verfügung stellen. Funktional werden QoS-Manager für Workflows und QoS-Manager für Dienste unterschieden. Durch ihre modulare Architektur unterstützen die QoS-Manager die Anbindung an unterschiedliche Geschäftskomponenten.

Eine Instanz eines QoS-Managers für Workflows ist einer WfMS-Instanz zugeordnet, sie ist damit für die Dienstgüte der in diesem System ablaufenden Workflows verantwortlich. Eine Instanz eines QoS-Managers für Dienste wird der einem Dienst unterlagerten Implementierung zugeordnet und setzt die geforderte Dienstgüte entsprechend den durch die Schnittstellen der Implementierung gebotenen Möglichkeiten mithilfe zuvor beschriebener Selbst-Management-Techniken durch. Durch die Integration eines Proxy-Ansatzes wird dabei sichergestellt, dass ein QoS-Manager gleichzeitig divergierende Dienstgüteanforderungen aus unterschiedlichen Workflows erfüllen kann.

Die Umsetzbarkeit der zunächst abstrakt beschriebenen Architektur wurde beispielhaft mit den im Rahmen der SOA- und IT-Management-Grundlagen beschriebenen Technologien demonstriert: Um einen möglichen Weg zur Integration der QoS-Manager in bestehende dienstorientierte Anwendungen aufzuzeigen, wurde deren Architektur auf die SOA-Komponententechnologie SCA abgebildet. Die SLA-Aushandlungsschnittstelle der QoS-Manager wurde durch WS-Agreement realisiert.

Zentrales Ziel des vorgestellten Dienstgüte-Management-Systems ist die Optimierung der durch die verwalteten Geschäftskomponenten erbrachten Dienstgüte auf Workflow-Ebene.

Die Kooperation der im System vorhandenen QoS-Management-Komponenten ist eine notwendige Voraussetzung dafür, das System in die Lage zu versetzen, eine globale Optimierung im Hinblick auf Dienstgüte und Ressourcennutzung (und damit gleichzeitig auch Kosten) anzustreben. Die Arbeit konzentriert sich dabei auf die Betrachtung der Dienstgütedimension Antwortzeit.

In der Arbeit wurde eine Möglichkeit zur Optimierung der Gesamtdienstgüte eines Workflows vertieft, bei der die an einem Workflow beteiligten QoS-Manager untereinander Antwortzeit-SLO-Anteile übertragen. Ziel ist dabei die Anpassung individueller Antwortzeit-SLOs einzelner Dienste, sodass es zum einen jedem Dienst möglich ist, sein SLO mit geringstmöglichem Ressourceneinsatz zu erfüllen, zum anderen trotzdem das Gesamt-SLO des Workflows eingehalten werden kann. Zu diesem Zweck bilden die beteiligten Dienste temporäre Kommunikationsstrukturen (Selbstorganisation). Zur Koordination der Übertragung von Antwortzeit-SLO-Anteilen wurde ein Auktionsverfahren auf Basis der Vickrey-Auktion ausgewählt. Durch Nutzung von Auktionsverfahren kann die Koordination von QoS-Managern an die auf der Geschäftsebene bestehenden Ziele und Prioritäten gekoppelt werden.

Weiterhin wurde ein Verfahren zur Koordination der gemeinsamen Nutzung von Ressourcen durch unterschiedliche QoS-Manager beschrieben, das Workflow-übergreifend angewandt werden kann. Beide Verfahren können in der Praxis parallel eingesetzt werden. Grundlage für die beschriebenen Koordinationsverfahren bildet eine initial vorzunehmende SLO-Aushandlung und -Zuteilung für die an einem Workflow beteiligten Dienste. In der Arbeit wurde beispielhaft ein Ansatz für eine solche, durch den Workflow-QoS-Manager vorzunehmende Zuteilung vorgestellt.

Im Rahmen einer prototypischen Implementierung wurde die Realisierung zentraler Architekturelemente und der entwickelten Verfahren zur Selbstorganisation von Komponenten des Dienstgüte-Management-Systems beispielhaft für das Dienstgüte-Management konkreter Komponenten vorgestellt. Der Workflow-QoS-Manager wurde für das WfMS Apache ODE implementiert, anhand eines QoS-Managers für JBoss-basierte Anwendungen wurde ein Ansatz zur Realisierung von automatisiertem Dienstgüte-Management für SOA-Dienste präsentiert.

Zur Abbildung größerer Szenarien wurde zusätzlich ein hybrider Simulationsansatz für die Kooperation von QoS-Managern implementiert. Der entwickelte Simulator greift zur Kommunikation zwischen einzelnen Management-Komponenten auf Elemente der prototypischen Implementierung zurück.

Im Rahmen der Evaluation des Ansatzes wurde zunächst untersucht, inwieweit die entwickelte Architektur und die prototypische Realisierung den im Rahmen der Arbeit definierten Anforderungen genügen. Im Anschluss wurde anhand eines Beispielszenarios gezeigt, wie die Koordination der an einem Workflow beteiligten QoS-Manager im Detail ablaufen kann. Danach wurden mithilfe des implementierten Simulators Untersuchungen zur Skalierbarkeit des Ansatzes durchgeführt. Schwerpunkt dieser Untersuchungen war die Betrachtung eines

Systems aus kooperierenden QoS-Managern, insbesondere im Hinblick auf den Kommunikationsaufwand. Insgesamt lässt die Evaluation vermuten, dass der Einsatz der entwickelten Kooperationsmechanismen auch in großen Systemen möglich ist.

9.2 Ausblick

In aktuellen IT-Anwendungen kann man bei zunehmender Kritikalität der Prozesse für den Geschäftserfolg eine steigende Komplexität feststellen. Im Rahmen dieser Arbeit wurde gezeigt, dass durch eine konsequente Automatisierung von IT-Management-Prozessen Dienstgüte-Management auch in komplexen, dezentralen Anwendungsumgebungen etabliert werden kann.

Durch die Standardisierung von Schnittstellen und die explizite Modellierung von Abhängigkeiten zwischen Diensten kann mithilfe von SOA-Anwendungen eine unternehmensweit homogene Anwendungslandschaft konzipiert werden. Für das Dienstgüte-Management und IT-Management allgemein besteht dabei die Herausforderung, die Homogenität einer solchen Anwendungslandschaft zu nutzen, um ein einheitliches, anwendungsübergreifendes Management zu etablieren. Langfristig ist es erforderlich, IT-Management-Aspekte als integralen Bestandteil des Designs von Geschäftsanwendungen aufzufassen – ähnlich wie derzeit bereits andere nichtfunktionale Aspekte (Skalierbarkeit, Sicherheit, etc.) betrachtet werden. Eine solche Entwicklung würde eine Basis für standardisiert verwaltbare Dienste schaffen und damit einer „Industrialisierung“ von IT-Management auf Basis von Selbst-Management-Ansätzen Nachdruck verleihen.

Den eigentlichen Mehrwert von Management-Standardisierung in homogenen Anwendungssystemen erreicht man jedoch durch Selbstorganisation einzelner Komponenten mit dem Ziel der Kooperation. Auf diese Weise lässt sich der Fokus des IT-Managements über administrative Grenzen hinweg auf übergreifende Ziele ausweiten. Der in dieser Arbeit entwickelte Ansatz zur Selbstorganisation von Management-Komponenten ist stark spezialisiert. Langfristig ist durch eine Standardisierung von Vorgehensweisen, Protokollen und Schnittstellen anzustreben, zu einer allgemeinen Methodik für den Entwurf selbstorganisierender IT-Management-Systeme zu gelangen und somit SOA-Design-Leitlinien wie lose Kopplung von Komponenten, explizite Modellierung von Abhängigkeiten usw. von der Geschäfts- auf die IT-Management-Ebene zu übertragen.

Ein solcher allgemeiner Ansatz wäre in der Anwendung nicht auf den Bereich der dienstorientierten Architekturen beschränkt, sondern könnte auf andere Anwendungsbereiche übertragen werden. Herausforderungen stellen sich bei der Entwicklung einer allgemeinen Methodik für die Umsetzung von Systemen zum autonomen IT-Management insbesondere in Bezug auf die Kontrolle des Verhaltens solcher Systeme in zuvor nicht erwarteten Situationen und den

Umgang mit sich widersprechenden, parallel zu verfolgenden Zielen. Aktuelle Ansätze haben zum Ziel, erwünschtes emergentes Systemverhalten im Umgang mit derartigen Situationen zu nutzen. Für ein geordnetes, ingenieurmäßiges Vorgehen bei Entwicklung und Betrieb solcher Systeme wird versucht, emergentes Systemverhalten mithilfe von geschachtelten Regelkreisen zu kontrollieren. Eine große Herausforderung liegt hierbei in der Stabilisierung solcher Systeme ohne zentrale Kontrollinstanz, da hier beispielsweise bereits das Erkennen von Schwingungen innerhalb des Systems problematisch ist.

Die hier skizzierten Ansätze lassen sich auch auf Umgebungen ausweiten, bei denen IT-Management im klassischen Sinne eine untergeordnete Rolle spielt. Stellvertretend seien an dieser Stelle die Adaption und und Selbst-Optimierung von kooperierenden Systemen im Wohn- oder Arbeitsbereich (*Smart Home*) genannt. Für derartige Anwendungsbereiche ist – neben Ansätzen zur dezentralen Kooperation von Systemen – auch die Übertragung von Erfahrungen aus dem klassischen IT-Management (in Bezug auf Monitoring, Modellierung, Analyse usw.) interessant.

Literaturverzeichnis

[AC04] AUSUBEL, LAWRENCE M. und PETER CRAMTON: *Vickrey Auctions with Reserve Pricing*. Economic Theory, 23(3):493–505, 2004.

[ACD+07] ANDRIEUX, ALAIN, KARL CZAJKOWSKI, ASIT DAN, KATE KEAHEY, HEIKO LUDWIG, TOSHIYUKI NAKATA, JIM PRUYNE, JOHN ROFRANO, STEVE TUECKE und MING XU: *Web Services Agreement Specification (WS-Agreement) 1.0*. Technischer Bericht, Open Grid Forum (OGF), März 2007.

[AMW+03] AGUILERA, M. K., J. C. MOGUL, J. L. WIENER, P. REYNOLDS und A. MUTHITACHAROEN: *Performance Debugging for Distributed Systems of Black Boxes*. In: *Proceedings of ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[AP07] ARDAGNA, D. und B. PERNICI: *Adaptive Service Composition in Flexible Processes*. Software Engineering, IEEE Transactions on, 33(6):369–384, Juni 2007.

[Ban98] BAN, BELA: *Design and Implementation of a Reliable Group Communication Toolkit for Java*. Technischer Bericht, Dept. of Computer Science, Cornell University, 1998.

[BBD+06] BALASUBRAMANIAM, SASITHARAN, DMITRI BOTVICH, WILLIAM DONNELLY, MÍCHEÁL Ó FOGHLÚ und JOHN STRASSNER: *Biologically Inspired Self-governance and Self-organisation for Autonomic Networks*. In: *BIONETICS '06: Proceedings of the 1st international conference on Bio inspired models of network, information and computing systems*, Seite 30, New York, NY, USA, 2006. ACM.

[BGR+05] BERBNER, RAINER, TOBIAS GROLLIUS, NICOLAS REPP, OLIVER HECKMANN, ERICH ORTNER und RALF STEINMETZ: *An approach for the Management of Service-oriented Architecture (SoA) based Application Systems*. In: *Enterprise Modelling and Information Systems Architectures, Proceedings, 2005*, Seiten 208–221, Oktober 2005.

[BJ87] BIRMAN, K. und T. JOSEPH: *Exploiting Virtual Synchrony in Distributed Systems*. SIGOPS Oper. Syst. Rev., 21(5):123–138, 1987.

[BJM+05] BABAOGLU, OZALP, MÁRK JELASITY, ALBERTO MONTRESOR, CHRISTOF FETZER, STEFANO LEONARDI, AAD VAN MOORSEL und MAARTEN

VAN STEEN (Herausgeber): *Self-Star Properties in Complex Information Systems*, Band 3460 der Reihe *Lecture Notes in Computer Science, Hot Topics*. Springer-Verlag, 2005.

[Bro07] *Der Brockhaus: in 15 Bänden.* F.A. Brockhaus, Leipzig, Mannheim, Permanent aktualisierte Online-Auflage. 2002-2007.

[BS95] BRESSOUD, T. C. und F. B. SCHNEIDER: *Hypervisor-based fault tolerance*. In: *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, Band 29, Dezember 1995.

[BW06] BÖKER, STEFANIE und MATTHIAS WOLF: *Performance Monitoring in einer SOA Infrastruktur*. Projektarbeit, FB Informatik, Fachhochschule Wiesbaden, Februar 2006.

[Cha05] CHAPPELL, DAVID: *Introducing Windows Communication Foundation*, September 2005. `http://www.davidchappell.com/IntroducingWCFv1.2.1.pdf` (besucht am 15.01.2009).

[CKF+02] CHEN, MIKE Y., EMRE KICIMAN, EUGENE FRATKIN, ARMANDO FOX und ERIC BREWER: *Pinpoint: Problem Determination in Large, Dynamic Internet Services*. In: *Proceedings of the Int. Conf. on Dependable Systems and Networks (DSN'02)*. IEEE, 2002.

[CKV01] CHOCKLER, GREGORY V., IDID KEIDAR und ROMAN VITENBERG: *Group communication specifications: a comprehensive study*. ACM Comput. Surv., 33(4):427–469, Dezember 2001.

[CMR06] COX, ALAN L., KARTIK MOHANRAM und SCOTT RIXNER: *Dependable ≠ unaffordable*. In: *ASID '06: Proceedings of the 1st workshop on Architectural and system support for improving software dependability*, 2006.

[CW07] CANNON, DAVID und DAVID WHEELDON: *ITIL – Service Operation*. The Stationary Office, Mai 2007.

[Deb05] DEBUSMANN, MARKUS: *Modellbasiertes Sevice Level Management verteilter Anwendungssysteme*. Dissertation, Universität Kassel, 2005.

[DEF+03] DIAO, Y., F. ESKESEN, S. FROEHLICH, J. L. HELLERSTEIN, L. F. SPAINHOWER und M. SURENDRA: *Generic Online Optimization of Multiple configuration Parameters With Application to a Database Server*. In: *Proceedings of the fourteenth IFIP/IEEE Workshop on Distributed systems: Operations and Management (DSOM 2003)*, Oktober 2003.

[DG04] DEBUSMANN, MARKUS und KURT GEIHS: *Towards dependable Web services*. In: *Dependable Computing, 2004. Proceedings. 10th IEEE Pacific Rim International Symposium on*, Seiten 5–14, März 2004.

[DKK+01] DEBUSMANN, MARKUS, REINHOLD KRÖGER, THOMAS KULLMANN, D. LINDNER, THOMAS PIWEK und CHRISTOPH WEYER: *Eine Manage-*

mentlösung zur Integration CORBA-basierter Anwendungen in bestehende Managementplattformen. PIK - Praxis der Informationsverarbeitung und Kommunikation, (24 / 4):194–201, 2001.

[DKS89] DEMERS, A., S. KESHAV und S. SHENKER: *Analysis and Simulation of a Fair Queueing Algorithm*. In: *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, Seiten 1–12, New York, NY, USA, 1989. ACM.

[DLP03] DAN, ASIT, HEIKO LUDWIG und GIOVANNI PACIFICI: *Web Services Differentiation with Service Level Agreements*, 2003. `http://www.ibm.com/developerworks/library/ws-slafram/` (besucht am 14.08.2009).

[DMT99] DISTRIBUTED MANAGEMENT TASK FORCE (DMTF): *COMMON INFORMATION MODEL (CIM) SPECIFICATION, Version 2.2*, 1999. `http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf`.

[DSK05] DEBUSMANN, MARKUS, MARKUS SCHMID und REINHOLD KROEGER: *Model-Driven Self-Management of Legacy Applications*. In: KUTVONEN, LEA und NANCY ALONISTIOTI (Herausgeber): *5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece, June 2005*, Seiten 56–67. IFIP, Springer, Juni 2005.

[DSZS09] DOHLE, HELGE, RAINER SCHMIDT, FRANK ZIELKE und THOMAS SCHÜRMANN: *ISO 20.000. Eine Einführung für Manager und Projektleiter*. dpunkt Verlag, 2009.

[DWH05a] DE WOLF, T. und T. HOLVOET: *Emergence Versus Self-Organisation: Different Concepts but Promising When Combined*. Lecture Notes in Computer Science, 3464, 2005.

[DWH05b] DE WOLF, T. und T. HOLVOET: *Towards a Methodology for Engineering Self-Organising Emergent Systems*. Self-Organization and Autonomic Informatics (I), Frontiers in Artificial Intelligence and Applications, 135:18–34, 2005.

[FGLS04] FREY, HANNES, DANIEL GÖRGEN, JOHANNES K. LEHNERT und PETER STURM: *Auctions in mobile multihop ad-hoc networks following the marketplace communication pattern*. In: MAHMOUD, QUSAY H. und HANS WEGHORN (Herausgeber): *Wireless Information Systems*, Seiten 161–169. INSTICC Press, 2004.

[Fre08] FREY, MICHAEL: *Design und Implementierung eines generischen QoS-Proxys für SOA-Dienste*. Diplomarbeit, FH Wiesbaden, FB Design Informatik Medien, Dezember 2008.

[FT91] FUDENBERG, DREW und JEAN TIROLE: *Game Theory*. MIT Press, Oktober 1991.

[GC03] GANEK, A. G. und T. A. CORBI: *The dawning of the autonomic computing era*. IBM Systems Journal, 42(1):5–18, 2003.

[GS05] GARSCHHAMMER, M. und M. SCHIFFERS: *Integrated IT-Management in Large-Scale, Dynamic, and Multi-Organizational Environments*. In: *Proceedings of the 12th Annual Workshop of HP OpenView University Association*. Hewlett Packard, 2005.

[GWBS03] GU, DAZHANG, L. R. WELCH, C. BRUGGEMAN und R. SHELLY: *The applicability of social models for self-organizing real-time systems*. In: *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003.

[HAN99] HEGERING, HEINZ-GERD, SEBASTIAN ABECK und BERNHARD NEUMAIR: *Integriertes Management vernetzter Systeme*. Dpunkt Verlag, 1. Auflage, 1999.

[HK97] HORSTMANN, MARKUS und MARY KIRTLAND: *DCOM Architecture*. Microsoft Corporation, Juli 1997. `http://msdn2.microsoft.com/en-us/library/ms809311.aspx` (besucht am 25.04.2008).

[HP97] HEWLETT PACKARD: *HP OpenView IT/Operations Concepts Guide*, August 1997. B4249-90011, Version A.04.00.

[HVH06] HUMM, BERNHARD, MARKUS VOSS und ANDREAS HESS: *Regeln für serviceorientierte Architekturen hoher Qualität*. Informatik-Spektrum, 29(6):395–411, Dezember 2006. `http://www.de.capgemini-sdm.com/web4archiv/objects/download/pdf/1/sdm_pub_humm_infospek.pdf`.

[HWM06] HERRMANN, KLAUS, MATTHIAS WERNER und GERO MÜHL: *A Methodology for Classifying Self-Organizing Software Systems*. International Transactions on Systems Science and Applications, 2(1):41–50, 2006.

[IBM06a] *WebSphere Application Server Manual – Getting performance data from request metrics*, 2006. `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.base.doc/info/aes/ae/tprf_rqenable.html`.

[IBM06b] INTERNATIONAL BUSINESS MACHINES CORPORATION: *IBM Systems Software Information Center: Application instrumentation*, 2006. `http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=/ewlminfo/eicaaappinstrument.htm` (besucht am 14.08.2009).

[IBM07] *IBM Build to Manage Toolkits*, 2007. `http://www-128.ibm.com/developerworks/eclipse/btm/`.

[IN07] IQBAL, MAJID und MICHAEL NIEVES: *ITIL – Service Strategy*. The Stationary Office, Mai 2007.

[ISO92] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *Open Systems Interconnection (OSI) System Management Specification ISO-10164-x*, 1992. `http://www.iso.ch`.

[ISO05] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC 20000-1:2005 Information technology – Service management – Part 1: Specification*, 2005. `http://www.iso.ch`.

[Jai91] JAIN, RAJ: *The art of computer systems performance analysis*. Wiley, 2. Auflage, 1991.

[JB96] JABLONSKI, STEFAN und CHRISTOPH BUSSLER: *Workflow Management: Modeling, Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

[Jun08] JUNGK, BERNHARD: *Bewertung von Selbstorganisationsmechanismen für Managementkomponenten auf Basis von Simulationen und Leistungsmessungen*. Master's Thesis, FH Wiesbaden, FB Design Informatik Medien, September 2008.

[JvdMB+07] JENNINGS, B., S. VAN DER MEER, S. BALASUBRAMANIAM, D. BOTVICH, M.O. FOGHLU, W. DONNELLY und J. STRASSNER: *Towards autonomic management of communications networks*. Communications Magazine, IEEE, 45(10):112–121, Oktober 2007.

[KAB+04] KEEN, M., A. ACHARYA, S. BISHOP, A. HOPKINS, S. MILINSKI, C. NOTT, R. ROBINSON, J. ADAMS und P. VERSCHUEREN: *Implementing an SOA Using an Enterprise Service Bus*. Redbooks. International Business Machines Corporation, Juli 2004. `http://www.redbooks.ibm.com/redbooks/pdfs/sg246346.pdf`.

[KBE99] KOKAR, MIECZYSLAW, KENNETH BACLAWSKI und YONET A. ERACAR: *Control Theory-Based Foundations of SelfControlling Software*. IEEE Intelligent Systems, Seiten 37–45, 1999.

[KBS05] KRAFZIG, DIRK, KARL BANKE und DIRK SLAMA: *Enterprise SOA*. Prentice Hall, 2005.

[KC03] KEPHART, JEFFREY O. und DAVID M. CHESS: *The Vision of Autonomic Computing*. IEEE Computer, Seiten 41–50, Januar 2003.

[KL03] KELLER, ALEXANDER und HEIKO LUDWIG: *The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services*. Journal of Network and Systems Management, 11(1):57–81, März 2003.

[Kri02] KRISHNA, VIJAY: *Auction Theory*. Academic Press, 2002.

[Lew99] LEWIS, LUNDY: *Service Level Management for Enterprise Networks*. Artech House Publishers, 1999.

[LM07] LACY, SHIRLEY und IVOR MACFARLANE: *ITIL – Service Transition*. The Stationary Office, Mai 2007.

[LRJ04] LI, PENG, BINOY RAVINDRAN und E. DOUGLAS JENSEN: *Adaptive Time-Critical Resource Management Using Time/Utility Functions: Past, Present, and Future*. In: *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, 2004.

[LS97] LAM, IOI K. und BRIAN SMITH: *Jacl: a Tcl implementation in java*. In: *TCLTK'97: Proceedings of the 5th conference on Annual Tcl/Tk Workshop 1997*, Seiten 4–4, Berkeley, CA, USA, 1997. USENIX Association.

[M+07] MELZER, INGO et al.: *Service-orientierte Architekturen mit Web Services*. Spektrum Akademischer Verlag, 2. Auflage, 2007.

[Mar07] MARGOLIS, BEN: *SOA for the Business Developer: Concepts, BPEL, and SCA*. MC Press, 2007.

[Mar08] MARINESCU, DAN: *Design and Evaluation of Self-Management Approaches for Virtual Machine-Based Environments*. Master's Thesis, FH Wiesbaden, FB Design Informatik Medien, Februar 2008.

[MDGA08] MOMM, CHRISTOF, THOMAS DETSCH, MICHAEL GEBHART und SEBASTIAN ABECK: *Model-Driven Development of Monitord Web Service Compositions*. In: HARROUD, H., A. BOULMAKOUL, A. EZZAT und C. PELTZ (Herausgeber): *Proceedings of the 15th Annual Workshop of the HP Software University Association*, Seiten 93–104, Juni 2008.

[Mik08] MIKULA, ANDREAS: *Automatisierte Überwachung von Dienstgütekriterien in SOA Workflows*. Diplomarbeit, FH Wiesbaden, FB Design Informatik Medien, August 2008.

[Mol97] MOLDOVANU, BENNY: *William Vickrey und die Auktionstheorie - Anmerkungen zum Nobelpreis 1996*. Sonderforschungsbereich 504 Publications 97-08, Sonderforschungsbereich 504, Universität Mannheim & Sonderforschungsbereich 504, University of Mannheim, Januar 1997.

[MSS08] MÜLLER-SCHLOER, CHRISTIAN und BERNHARD SICK: *Controlled Emergence and Self-Organization*. In: WÜRTZ, ROLF P. (Herausgeber): *Organic Computing*, Seiten 81–104. Springer, März 2008.

[MSvdMW04] MÜLLER-SCHLOER, CHRISTIAN, CHRISTOPH VON DER MALSBURG und ROLF P. WÜRTZ: *Organic Computing*. Informatik Spektrum, 27(4):332–336, 2004.

[MWJ+07] MUEHL, GERO, MATTHIAS WERNER, MICHAEL A. JAEGER, KLAUS HERRMANN und HELGE PARZYJEGLA: *On the Definitions of Self-Managing*

and Self-Organizing Systems. In: BRAUN, TORSTEN, GEORG CARLE und BURKHARD STILLER (Herausgeber): *KIVS 2007: Kommunikation in Verteilten Systemen*. VDE Verlag, März 2007.

[Mye81] MYERSON, R.: *Optimal auction design*. Mathematics of Operations Research, 6(1):58–73, 1981.

[NMMS99] NARASIMHAN, PRIYA, LOUISE E. MOSER und P.M. MELLIAR-SMITH: *Using Interceptors to Enhance CORBA*. Computer, 32(7):62–68, 1999.

[NS03] NATIS, YEFIM V. und ROY W. SCHULTE: *SSA Research Note SPA-19-5971, Introduction to Service Oriented Architecture*. Technischer Bericht, the Gartner Group, 2003.

[OAS03] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS) UDDI SPECIFICATIONS TC - COMMITTEE: *UDDI Version 3.0.1 - UDDI Spec Technical Committee Specification*, 2003. http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf.

[OAS05a] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.0*, 2005. http://docs.oasis-open.org/wsdm/2004/12/wsdm-mows-1.0.pdf.

[OAS05b] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Web Services Distributed Management: Management using Web Services (WSDM-MUWS 1.0), Part 1*, 2005. http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf.

[OAS05c] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Web Services Distributed Management: Management using Web Services (WSDM-MUWS 1.0), Part 2*, 2005. http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part2-1.0.pdf.

[OAS06a] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Reference Model for Service Oriented Architecture 1.0 - OASIS Standard*, Oktober 2006. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[OAS06b] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Web Services Base Notification 1.3 (WS-BaseNotification) - OASIS Standard*, Oktober 2006. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf.

[OAS07] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Web Services Business Process Execution*

Language Version 2.0 - OASIS Standard, April 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

[OAS08] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Reference Architecture for Service Oriented Architecture Version 1.0 - Public Review Draft 1*, April 2008. http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf.

[OGC01] *Best Practice for Service Delivery*. ITIL - The Key to Managing IT Services. Office of Government Commerce (OGC), 2001.

[OKS+03] OGLE, DAVID, HEATHER KREGER, ABDI SALAHSHOUR, JASON CORNPROPST, ERIC LABADIE, MANDY CHESSELL, BILL HORN, JOHN GERKEN, JAMES SCHOECH und MIKE WAMBOLDT: *Canonical Situation Data Format: The Common Base Event V1.0.1*. IBM, 2003. http://www.eclipse.org/tptp/platform/documents/resources/cbe101spec/CommonBaseEvent_SituationData_V1.0.1.pdf.

[OMG07] OBJECT MANAGEMENT GROUP: *Business Process Modeling Notation (BPMN) Specification Version 1.1 – Draft*, Juli 2007. http://www.omg.org/docs/dtc/07-06-03.pdf.

[OMG08] OBJECT MANAGEMENT GROUP: *Common Object Request Broker Architecture (CORBA/IIOP) Version 3.1*, Januar 2008. http://www.omg.org/spec/CORBA/3.1/.

[OSO07a] OPEN SOA COLLABORATION: *SCA Policy Framework Version 1.0*, März 2007. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications.

[OSO07b] OPEN SOA COLLABORATION: *SCA Service Component Architecture – Assembly Model Specification Version 1.0*, März 2007. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications.

[PB98] PASSINO, K. und K. BURGESS: *Stability Analysis of Discrete Event Systems*. John Wiley & Sons, Inc., New York, 1998.

[PGH+01] PAREKH, S., N. GANDHI, J. HELLERSTEIN, D. TILBURY, T.S. JAYRAM und J. BIGUS: *Using Control Theory to Achieve Service Level Objectives In Performance Management*. In: *Proceedings of the Seventh International Symposium on Integrated Network Management (IM)*, 2001.

[PH07] PARASHAR, MANISH und SALIM HARIRI (Herausgeber): *Autonomic Computing: Concepts, Infrastructure, and Applications*. Taylor & Francis, Inc., Bristol, PA, USA, 2007.

[Pow96] POWELL, DAVID: *Group communication*. Communications of the ACM, 39(4):50–53, 1996.

[Red09] REDHAT INC.: *Design and Implementation of a Reliable Group Communication Toolkit for Java*, 1.13 Auflage, 2009. `http://www.jgroups.org/manual/pdf/manual.pdf`.

[RL07] RUDD, COLIN und VERNON LLOYD: *ITIL – Service Design*. The Stationary Office, Mai 2007.

[Sch06] SCHAEFER, JAN: *An Approach for Fine-Grained Web Service Performance Monitoring*. In: *Distributed Applications and Interoperable Systems : 6th IFIP WG 6.1 Int. Conf., DAIS 2006, Proceedings*. IFIP, Springer, Juni 2006.

[SDKH05a] SCHMID, MARKUS, MARKUS DEBUSMANN, REINHOLD KRÖGER und MARC HALBIG: *Flexible Charging and Accounting of Distributed Services*. In: MARQUES, BRUNO F., THOMAS NEBE und RAUL F. OLIVEIRA (Herausgeber): *Proceedings of the 12th Annual Workshop of HP OpenView University Association*, Seiten 53–66. Infonomics-Consulting, Juli 2005.

[SDKH05b] SCHMID, MARKUS, MARKUS DEBUSMANN, REINHOLD KRÖGER und MARC HALBIG: *Towards a Transaction-Based Charging and Accounting of Distributed Services and Applications*. In: CLEMM, ALEXANDER, OLIVIER FESTOR und AIKO PRAS (Herausgeber): *IM 2005 - IFIP/IEEE International Symposium on Integrated Network Management*, Seiten 1229–1232. IEEE, Mai 2005.

[SH06] STERRITT, R. und M.G. HINCHEY: *Biologically-inspired concepts for self-management of complexity*. In: *Engineering of Complex Computer Systems, 2006. ICECCS 2006. 11th IEEE International Conference on*, Seiten 163 – 168, 0-0 2006.

[Sha03] SHAPIRO, ROBERT: *A technical comparison of XPDL, BPML and BPEL4WS*. Business Process Trends White Paper, Januar 2003. http://www.bptrends.com/publicationfiles/Comparison%20of%20XPDL%20and%20BPML%5FBPEL%2012%2D8%2D02111%2Epdf%2Epdf.

[SHLP05] SCHMIDT, M., B. HUTCHISON, P. LAMBROS und R. PHIPPEN: *The Enterprise Service Bus: Making service-oriented architecture real*. IBM Systems Journal, 44(4), November 2005. `http://www.research.ibm.com/journal/sj/444/schmidt.pdf`.

[SMJ00] STURM, RICK, WAYNE MORRIS und MARY JANDER: *Foundations of Service Level Management*. SAMS Publishing, April 2000.

[SMK08] SCHMID, MARKUS, DAN MARINESCU und REINHOLD KROEGER: *A Framework for Autonomic Performance Management of Virtual Machine-Based Services*. In: HARROUD, HAMID, ABDEL BOULMAKOUL und CHRIS PELTZ

(Herausgeber): *Proceedings of the 15th Annual Workshop of the HP Software University Association, hosted by Al Akhawayn University in Ifrane*, Juni 2008.

[SN96] SCHULTE, ROY W. und YEFIM V. NATIS: *SSA Research Note SPA-401-068, Service Oriented Architectures, Part 1*. Technischer Bericht, the Gartner Group, 1996.

[Spa07] SPALDING, GEORGE: *ITIL – Continual Service Improvement*. The Stationary Office, Mai 2007.

[SPJ08] STEIN, SEBASTIAN, TERRY R. PAYNE und NICHOLAS R. JENNINGS: *Flexible provisioning of web service workflows*. ACM Transactions on Internet Technology, 8, 2008.

[SRS07] STRASSNER, JOHN, DAVID RAYMER und SRINI SAMUDRALA: *Providing Seamless Mobility Using the FOCALE Autonomic Architecture*. In: KOUCHERYAVY, YEVGENI, JARMO HARJU und ALEXANDER SAYENKO (Herausgeber): *NEW2AN*, Band 4712 der Reihe *Lecture Notes in Computer Science*, Seiten 330–341. Springer, 2007.

[SS05] SUZUKI, JUNICHI und TATSUYA SUDA: *A Middleware Platform for a Biologically Inspired Network Architecture Supporting Autonomous and Adaptive Applications*. IEEE Journal on Selected Areas in Communications, 23:249–260, 2005.

[SSK08] SCHMID, MARKUS, JAN SCHAEFER und REINHOLD KROEGER: *Ein MDSD-Ansatz zum QoS-Monitoring von Diensten in Service-orientierten Architekturen*. PIK - Praxis der Informationsverarbeitung und Kommunikation, (31 /4):232–238, Dezember 2008.

[SSK09] SCHMID, MARKUS, JAN SCHAEFER und REINHOLD KROEGER: *Integriertes Performance-Monitoring von SOA-Anwendungen*. In: *Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen (WowKiVS), Kassel, 2009*, Band 17 der Reihe *Electronic Communications of the EASST*, März 2009.

[SSTK08] SCHMID, MARKUS, THORSTEN STEIN, MARCUS THOSS und REINHOLD KROEGER: *An Eclipse IDE Extension for Pattern-based Software Instrumentation*. In: BAUSE, FALKO und PETER BUCHHOLZ (Herausgeber): *Proceedings 14th GI/ITG Conference Measurement, Modelling and Evaluation of Computer and Communication Systems, Dortmund*, Seiten 299–302. VDE Verlag, März 2008.

[ST07] STARKE, GERNOT und STEFAN TILKOV (Herausgeber): *SOA-Expertenwissen*. Dpunkt Verlag, 2007.

[Sta99] STALLINGS, WILLIAM: *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley, 3. Auflage, 1999.

[Ste07] STEIN, THORSTEN: *Ein Kommunikations-Framework für die Selbstorganisation von Software-Systemen*. Diplomarbeit, FH Wiesbaden, FB Design Informatik Medien, Februar 2007.

[Sti96] STILLER, BURKHARD: *Quality-of-Service – Dienstgüte in Hochleistungsnetzen*. Nummer 21 in *Thomson's Aktuelle Tutorien*. Thomson Publishing, 1. Auflage, 1996.

[Str05] STRASSNER, JOHN: *Autonomic networking theory and practice*. In: *Integrated Network Management, 2005. IM 2005. 2005 9th IFIP/IEEE International Symposium on*, Seiten 786–786, 2005. IEEE Tutorial.

[STS+04] SCHEER, A. W., O. THOMAS, C. SEEL, G. MARTIN und B. KAFFAI: *Geschäftsprozessorientierte Softwarearchitekturen: Revolution auf dem Software-Markt?* In: DADAM, P. und MANFRED REICHERT (Herausgeber): *Proceedings of 34. Jahrestagung der Gesellschaft für Informatik*, Band 1 der Reihe *Lecture Notes in Informatics*, Seiten 2 – 13, 2004.

[STTK07] SCHMID, MARKUS, MARCUS THOSS, THOMAS TERMIN und REINHOLD KROEGER: *A Generic Application-Oriented Performance Instrumentation for Multi-Tier Environments*. In: *10th IFIP/IEEE International Symposium on Integrated Network Management (IM2007), München*, Seiten 304–313. IEEE, Mai 2007.

[SUN03a] SUN MICROSYSTEMS: *Java Remote Method Invocation - Distributed Computing for Java*, August 2003. `http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp` (besucht am 25.04.2008).

[SUN03b] SUN MICROSYSTEMS: *JSR 914: Java Message Service (JMS) API*, Dezember 2003. `http://www.jcp.org/en/jsr/detail?id=914` (besucht am 14.01.2009).

[SUN05] SUN MICROSYSTEMS: *JSR-000208 Java Business Integration 1.0 Final Release*, August 2005. `http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html` (besucht am 14.01.2009).

[SUN06a] SUN MICROSYSTEMS: *Java Platform, Enterprise Edition (Java EE) Specification, v5*, Mai 2006. `http://jcp.org/aboutJava/communityprocess/final/jsr244/index.html` (besucht am 07.01.2008).

[SUN06b] SUN MICROSYSTEMS: *JSR-000220 Enterprise JavaBeans 3.0 Final Release*, Mai 2006. `http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html` (besucht am 04.05.2008).

[Sun08] SUN MICROSYSTEMS, INC.: *Java Management Extensions Technology* , 2008. http://java.sun.com/javase/technologies/core/mntr\discretionary{-}{}{}mgmt/javamanagement.

[Tay07] TAYLOR, SHARON (Herausgeber): *ITIL – The Official Introduction to the ITIL Service Lifecycle*. The Stationary Office, Mai 2007.

[Ten00] TENNENHOUSE, DAVID: *Proactive computing*. Communications of the ACM, 43(5):43–50, Mai 2000.

[Ter06] TERMIN, THOMAS: *Eine generische Performance-Instrumentierung des JBoss Application Servers*. Diplomarbeit, FH Wiesbaden, FB Design Informatik Medien, April 2006.

[Tiv03] TIVOLI SYSTEMS: *Tivoli Management Framework User's Guide*, November 2003. Version 4.11.

[TOG04] THE OPEN GROUP: *Application Response Measurement (ARM) Issue 4.0, V2 - C Binding*, 2004. http://www.opengroup.org/management/arm/.

[Tur02] TURNER, J.S.: *New directions in communications (or which way to the information age?)*. Communications Magazine, IEEE, 40(5):50–57, Mai 2002.

[Ung05] UNGER, TOBIAS: *Aggregation von QoS und SLAs in BPEL Geschäftsprozessen*. Diplomarbeit, Institut für Architektur von Anwendungssystemen, Universität Stuttgart, 2005. http://elib.uni-stuttgart.de/opus/volltexte/2005/2453/pdf/DIP_2305.pdf.

[Utk07] UTKIN, EUGENIA: *A Monitoring Tool for the Apache ODE BPEL Engine*. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, November 2007.

[Ver99] VERMA, D.: *Supporting Service Level Agreements on IP Networks*. Macmillan Technical Publishing, 1999.

[Vic61] VICKREY, WILLIAM: *Counterspeculation, Auctions, and Competitive Sealed Tenders*. The Journal of Finance, 16(1):8–37, 1961.

[W3C01] WORLD WIDE WEB CONSORTIUM: *Web Services Description Language (WSDL) 1.1*, März 2001. W3C Working Group Note, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[W3C03] WORLD WIDE WEB CONSORTIUM: *SOAP Version 1.2*, Juni 2003. http://www.w3.org/TR/soap12.

[W3C07] WORLD WIDE WEB CONSORTIUM: *Web Services Policy 1.5 - Framework*, September 2007. http://www.w3.org/TR/2007/REC-ws-policy-20070904.

[Wel93] WELLMAN, MICHAEL P.: *A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems*. Journal of Artificial Intelligence Research, 1:1–23, 1993.

[WFM05] THE WORKFLOW MANAGEMENT COALITION: *Workflow Management Coalition Workflow Standard: Process Definition Interface – XML Process Definition Language*, 2.0 Auflage, Oktober 2005. `http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf`.

[WHH+92] WALDSPURGER, C.A., T. HOGG, B.A. HUBERMAN, J.O. KEPHART und W.S. STORNETTA: *Spawn: a distributed computational economy*. Software Engineering, IEEE Transactions on, 18(2):103–117, Februar 1992.

[WvdADtH03] WOHED, PETIA, WIL M.P. VAN DER AALST, MARLON DUMAS und ARTHUR H.M. TER HOFSTEDE: *Analysis of Web Services Composition Languages: The Case of BPEL4WS*. In: *Proceedings 22nd International Conference on Conceptual Modelling*, Seiten 200–215. Springer, Juli 2003.

[WWWM01] WELLMAN, M., W. WALSH, P. WURMAN und MACKIE J. MASON: *Auction protocols for decentralized scheduling*. Games and Economic Behavior, 35:271–303, 2001.

[ZBS09] ZAID, FARID, RAINER BERBNER und RALF STEINMETZ: *Leveraging the BPEL Event Model to Support QoS-aware Process Execution*. In: *16. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2009), Kassel*, Informatik aktuell, Seiten 93–104. Springer, März 2009.

Teil IV

Anhang

A Abkürzungen

	— A —
API	Application Programming Interface
ARM	Application Response Measurement
	— B —
BPE	Business Process Engine
BPEL	Business Process Execution Language
BPML	Business Process Modeling Language
BPMN	Business Process Management Notation
	— C —
CBE	Common Base Events
CIM	Common Information Model
CORBA	Common Object Request Broker Architecture
	— D —
DMTF	Distributed Management Task Force
	— E —
EAI	Enterprise Application Integration
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
	— F —
FIFO	First in first out
	— H —
HTTP	Hypertext Transfer Protocol

— I —

ISP	Internet Service Provider/Providing
ITIL	IT Infrastructure Library
ITU	International Telecommunication Union

— J —

J2EE	Java 2 Enterprise Edition
JBI	Java Business Integration
JEE	Java Enterprise Edition
JMS	Java Message Service
JMX	Java Management Extensions

— L —

LB	Leaky Bucket

— M —

MDA	Model Driven Architecture
MDSD	Model Driven Software Design

— N —

NNM	Network Node Manager

— O —

OLA	Operational Level Agreement
OMG	Object Management Group

— Q —

QoS	Quality of Service

— R —

RFC	Request for Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call

— S —

SCA	Service Component Architecture
SLA	Service Level Agreement
SLM	Service Level Managememt
SLO	Service Level Objective
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture

— U —

UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

— V —

VM	Virtual Machine
VMM	Virtual Machine Monitor

— W —

WCF	Windows Communication Foundation
WBEM	Web based Enterprise Management
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
WSLA	Web Service Level Agreement
WS-BPEL	Buiness Process Execution Language

— X —

XML	Extensible Markup Language
XPDL	XML Process Definition Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

B XML-Schemata

B.1 XML-Schema für Dienstgüteanforderungen

Zur Verwendung in der prototypischen Realisierung der QoS-Management-Komponenten wurde ein minimalistisches XML-Schema entwickelt, das sich inhaltlich an der in [Ung05] vorgestellten WS-Agreement-Erweiterung orientiert.

Das XML-Schema erlaubt die Einschränkung und Verknüpfung von minimal/maximal Antwortzeit- und Durchsatzanforderungen mithilfe der Konstrukte `and` und `or`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://vs.cs.fh-wiesbaden.de/
   agreement/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:qos="http://vs.cs.fh-wiesbaden.de/agreement/"
   elementFormDefault="qualified" attributeFormDefault="
   unqualified">
  <xsd:complexType name="qosElement" abstract="true"/>
  <xsd:complexType name="and">
    <xsd:complexContent>
    <xsd:extension base="qos:qosElement">
      <xsd:sequence minOccurs="2" maxOccurs="2">
        <xsd:element name="qosElement" type="qos:qosElement"/>
      </xsd:sequence>
    </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="or">
    <xsd:complexContent>
    <xsd:extension base="qos:qosElement">
      <xsd:sequence minOccurs="2" maxOccurs="2">
        <xsd:element name="qosElement" type="qos:qosElement"/>
      </xsd:sequence>
    </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="responseTime">
```

```
      <xsd:complexContent>
      <xsd:extension base="qos:qosElement">
        <xsd:attribute name="unit" type="xsd:string" use="required"
           />
        <xsd:attribute name="variable" type="xsd:string" use="
           required"/>
        <xsd:attribute name="max" type="xsd:float" use="optional"
           default="0"/>
      </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="throughput">
      <xsd:complexContent>
      <xsd:extension base="qos:qosElement">
        <xsd:attribute name="unit" type="xsd:string" use="required"
           />
        <xsd:attribute name="variable" type="xsd:string" use="
           required"/>
        <xsd:attribute name="min" use="optional"/>
        <xsd:attribute name="max" use="optional" default="0"/>
      </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="qos" type="qos:qosElement"/>
</xsd:schema>
```

Quellcode B.1: XML-Schema zur Spezifikation von Dienstgüteanforderungen aus den Dimensionen Antwortzeit und Durchsatz

B.2 XML-Schemata zur Spezifikation von Simulationsszenarien

Zur Spezifikation von Simulationsszenarien wurden in [Jun08] XML-Beschreibungssprachen zur Definition von SOA-Topologien und SOA-Workflows definiert. Die diesen Beschreibunggsprachen zugrunde liegenden XML-Schemata sind im Folgenden dargestellt.

B.2.1 XML-Schema zur Definition von Dienste-Topologien

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  targetNamespace="http://vs.cs.fh-wiesbaden.de/simulation/
     serviceModel"
  xmlns:tns="http://vs.cs.fh-wiesbaden.de/simulation/serviceModel
     "
  elementFormDefault="qualified">
  <xsd:complexType name="service">
    <xsd:attribute name="serviceID" type="xsd:int" use="required"
       />
    <xsd:attribute name="responseTime" type="xsd:long" use="
       required"/>
  </xsd:complexType>
  <xsd:complexType name="responseTimeChange">
    <xsd:attribute name="startTime" type="xsd:long"/>
    <xsd:attribute name="newResponseTime" type="xsd:long"/>
    <xsd:attribute name="random" type="xsd:boolean"/>
    <xsd:attribute name="lowerBoundStartTime" type="xsd:long"/>
    <xsd:attribute name="upperBoundStartTime" type="xsd:long"/>
    <xsd:attribute name="lowerBoundResponseTime" type="xsd:long"/
       >
    <xsd:attribute name="upperBoundResponseTime" type="xsd:long"/
       >
  </xsd:complexType>
  <xsd:complexType name="responseTimeModel">
    <xsd:sequence>
      <xsd:element name="responseTimeChange" type="
         tns:responseTimeChange" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="serviceID" type="xsd:int" use="required"
       />
  </xsd:complexType>
  <xsd:element name="serviceModel">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:element name="service" type="tns:service" minOccurs=
           "0"/>
        <xsd:element name="responseTimeModel" type="
           tns:responseTimeModel" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Quellcode B.2: XML-Schema zur Definition einer Dienste-Topologie

B.2.2 XML-Schema zur Definition von Workflows

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://vs.cs.fh-wiesbaden.de/simulation/
     workflow"
  xmlns:tns="http://vs.cs.fh-wiesbaden.de/simulation/workflow"
  elementFormDefault="qualified">
  <xsd:complexType name="workflow">
    <xsd:choice>
      <xsd:element name="sequence" type="tns:sequence"/>
      <xsd:element name="parallel" type="tns:parallel"/>
      <xsd:element name="activity" type="tns:activity"/>
    </xsd:choice>
    <xsd:attribute name="maxResponseTime" type="xsd:long" use="
       required"/>
    <xsd:attribute name="priority" type="xsd:int" use="required"/
       >
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="activity">
    <xsd:attribute name="assignedService" type="xsd:int" use="
       required"/>
  </xsd:complexType>
  <xsd:complexType name="sequence">
    <xsd:sequence minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="parallel" type="tns:parallel" maxOccurs=
         "1" minOccurs="0"/>
      <xsd:element name="activity" type="tns:activity" minOccurs=
         "0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="parallel">
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="sequence" type="tns:sequence" minOccurs=
         "0"/>
      <xsd:element name="activity" type="tns:activity" minOccurs=
         "0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="workflow" type="tns:workflow"/>
</xsd:schema>
```

Quellcode B.3: XML-Schema zur Definition einer Workflow-Struktur

C Veröffentlichungen im Kontext dieser Dissertation

Die im Zusammenhang mit dieser Dissertation entstandenen wissenschaftlichen Veröffentlichungen sind im Folgenden in absteigender chronologischer Reihenfolge aufgeführt:

1. M. Schmid; J. Schäfer; R. Kröger: „*Integriertes Performance-Monitoring von SOA-Anwendungen*“, Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen (WowKiVS), Kassel, 2009, Vol. 17, Electronic Communications of the EASST, März 2009
2. M. Schmid; J. Schäfer; R. Kröger: „*Ein MDSD-Ansatz zum QoS-Monitoring von Diensten in Service-orientierten Architekturen*“, Nummer 31 /4, PIK - Praxis der Informationsverarbeitung und Kommunikation, Seite 232-238, K.G. Saur Verlag, 2008
3. M. Schmid; D. Marinescu; R. Kröger: „*A Framework for Autonomic Performance Management of Virtual Machine-Based Services*“, Proceedings of the 15th Annual Workshop of the HP Software University Association, hosted by Al Akhawayn University in Ifrane, Editors: H. Harroud; A. Boulmakoul; C. Peltz, Juni 2008
4. M. Schmid; R. Kröger: „*Decentralised QoS-Management in Service Oriented Architectures*“, IFIP, Springer Verlag, Distributed Applications and Interoperable Systems: 8th IFIP WG 6.1 International Conference, DAIS 2008, Oslo, Norway, June 4-6, 2008, Seite 44-57, Editors: R. Meier; S. Terzis, Juni 2008
5. M. Schmid; M. Thoss; T. Termin; R. Kröger: „*A Generic Application-Oriented Performance Instrumentation for Multi-Tier Environments*“, IEEE, 10th IFIP/IEEE International Symposium on Integrated Network Management (IM2007), München, Seite 304-313, Mai 2007
6. M. Schmid: „*Ein Ansatz für das Service Level Management in dynamischen Architekturen*“, VDE Verlag, KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, Bern, Schweiz, März 2007
7. M. Schmid; K. Geihs: „*Self-Organisation in the Context of QoS Management in Service Oriented Architectures*“, Infonomics-Consulting, Stuttgart, Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by University of Nice

at Cote d'Azur May 21 - 24, 2006, Seite 153-164, Editors: K. Boudaoud; N. Nobelis; T. Nebe, Mai 2006

8. Schmid, Markus: „*Selbstorganisation im Kontext des QoS-Managements Service-orientierter Architekturen*“, Workshop Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS), Kassel, März 2006
9. M. Schmid; R. Kröger: „*Selbst-Management-Ansätze im eBusiness-Umfeld*“, Nummer 28 / 4, PIK - Praxis der Informationsverarbeitung und Kommunikation, Seite 211-216, K.G. Saur Verlag, 2005
10. M. Debusmann; M. Schmid; R. Kröger: „*Model-Driven Self-Management of Legacy Applications*“, IFIP, Springer, 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece, June 2005, Seite 56-67, Editors: L. Kutvonen; N. Alonistioti, Juni 2005

D Messwerte

Im Folgenden sind die den Auswertungen in Kapitel 8 zugrunde liegenden Messwerte tabellarisch aufgeführt.

D.1 Experimentdauer und versandte Nachrichten

Die folgende Tabelle zeigt für jeden der durchgeführten Simulationsläufe jeweils die Dauer bis zum Erreichen eines stabiles Zustands oder bis zum Abbruch des Experiments nach 120 Sekunden und die Anzahl der bis zu diesem Zeitpunkt versandten Nachrichten.

Experiment	Dauer und Anzahl Nachrichten													
	2		4		8		16		32		64		128	
	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#
1	0		5034	14	10079	47	20230	1039	10359	286	50634	11218	39295	42946
2	0		5067	18	15064	83	5121	1990	35464	235	56000	8281	108022	81490
3	0		5016	14	5092	26	25152	1421	10187	312	76491	10186	43016	33253
4	5019	6	5061	10	5065	26	20122	1967	40496	439	55908	6753	82364	57367
5	0		0		5035	59	10191	2864	50470	245	50722	12862	119230	65845
6	5019	6	5022	12	10102	62	25212	1944	25333	523	86117	13718	31886	29081
7	5013	4	5020	14	10034	64	15206	1358	15064	196	71191	10494	118633	80532
8	0		5080	16	10022	55	10108	1758	20300	279	86422	12386	102290	63000
9	5016	5	5028	12	5095	32	30033	2349	30449	344	65865	14845	103411	75182
10	0		5033	14	5081	57	15169	1744	20193	405	61243	7166	78264	63394
11	5020	6	5025	12	10062	87	30041	1307	35379	364	61112	9128	99214	39057
12	5015	4	10029	26	10058	56	15156	1851	45389	554	71317	9496	118536	45556
13	0		10092	24	10025	86	30252	1502	25337	450	56065	7391	79268	56090
14	0		5051	16	5094	24	15085	2523	55469	312	45958	6594	57606	51942
15	0		5014	14	10042	86	15054	2251	30335	325	66201	9172	26433	31476
16	0		5024	12	5068	42	10086	1322	35314	146	70865	10925	82750	59021
17	0		5031	14	10063	65	10059	1718	25341	138	66096	8857	112329	46870
18	0		5063	10	5044	62	25311	1734	30281	448	50635	11322	84450	27625
19	0		5021	19	20061	96	10082	1920	40495	343	75960	10937	107753	96925
20	5021	6	0		20058	96	30028	2515	40422	379	65789	12169	27371	29422
21	0		5026	22	15090	70	20233	1598	35353	354	65923	11799	103396	54101
22	0		0		10108	62	15218	2183	30209	291	55905	14352	87546	52865
23	5013	4	10047	30	15042	102	20198	1277	10366	374	66102	8453	88545	51198
24	5013	4	5014	11	5074	63	5127	1770	45046	60	40419	10252	93278	50494
25	0		5067	12	10068	74	15111	1119	20173	289	55853	10841	76916	90788
26	5015	4	0		15165	95	30227	1584	25335	625	51351	6050	82791	60075
27	0		5015	14	20080	106	25210	3249	35257	316	65986	14428	113957	56569
28	5014	4	5017	16	10038	88	20207	1130	30425	272	45908	6604	113706	44037
29	0		10033	23	20160	115	35116	1908	50062	432	60666	14470	77603	56842
30	0		0		10056	78	10112	786	20273	374	76104	14145	73458	54591
31	5015	4	5045	15	10028	106	15119	3539	35364	316	20264	8588	103980	59624
32	0		5028	14	15062	78	10172	2893	50681	198	55963	8138	58080	45935
33	0		10082	23	0		10077	2116	20205	278	60856	12751	88663	29086
34	0		5026	18	10068	92	15249	1229	30325	224	56109	7526	74345	32149
35	0		0		15061	134	5052	1090	25458	96	60708	12404	49737	29296
36	5019	6	10044	23	10015	66	20246	1769	35295	282	25342	10892	98905	54776
37	0		5014	14	15073	110	20160	2416	25288	702	65937	13653	98341	78724
38	5020	6	20018	38	10070	56	15121	1209	10160	346	90969	15280	113734	67935
39	0		5020	9	5045	82	20234	1945	25306	298	61187	7244	107872	73133
40	0		5012	10	5094	30	10118	1132	30406	331	60909	8141	97654	59299
41	0		5026	12	10085	58	20124	884	20574	272	56023	14624	84455	41336
42	0		0		5078	38	20181	1066	25312	301	41112	7132	53842	68914
43	0		10043	24	10039	95	30198	2320	35385	466	60906	11944	113012	49963
44	0		10024	20	10133	43	10167	1594	15065	164	70726	16623	103751	88596
45	0		10047	26	25021	120	10152	2392	35249	385	76492	10147	87849	49398

Experiment	Dauer und Anzahl Nachrichten													
	2		4		8		16		32		64		128	
	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#
46	5016	5	5023	12	15067	66	30274	1463	35530	734	76134	11219	113605	48001
47	0		10046	22	10069	94	10024	1338	10340	325	55976	10372	93132	58745
48	5016	4	5061	13	10080	58	20256	1832	40490	252	35433	8523	57202	40977
49	0		15030	24	15072	72	5104	2238	25318	95	35436	9722	102867	69842
50	5015	4	0		10116	54	20272	1237	30456	330	81282	12444	107729	88584
51	0		5044	14	10027	102	25149	1328	30387	340	50930	9620	119120	57209
52	0		5064	15	15054	98	10201	1702	25254	513	50816	10100	118776	68750
53	5020	6	10074	22	5078	32	20179	1833	30382	374	50546	9449	113983	53036
54	0		0	43	10030	71	10193	1600	35471	148	70747	11552	46767	46967
55	5013	4	5020	10	5091	36	10116	1873	25272	176	81068	10935	26226	35123
56	0		5027	12	15038	83	25210	2199	20257	344	40743	12729	68344	66525
57	0		10079	20	10036	86	15173	1256	30264	374	35602	7874	119248	63351
58	0		10073	30	5082	80	15206	2869	55525	192	91194	14103	68205	63309
59	5014	4	10023	19	10025	50	10178	1630	40516	367	46140	7452	71718	26925
60	5015	4	10163	23	15064	102	25219	1426	30371	350	76060	10372	94936	37213
61	0		5030	16	10133	61	20193	3894	65593	394	55883	8733	74152	28706
62	5020	6	5022	18	15038	118	20198	1538	35441	310	45544	9363	58057	48789
63	5020	6	5067	12	5073	34	15122	1360	35386	320	80939	16552	83345	66432
64	5015	4	5059	13	25027	114	20221	1751	35492	329	71077	10853	67878	45683
65	0		0		5068	40	15060	1832	40569	408	20107	5499	118557	50405
66	0		5042	16	5066	34	15126	1982	25432	202	76288	10985	112678	84249
67	5019	6	5067	13	10072	93	15054	1778	35416	518	45626	9098	77484	56113
68	5055	4	5051	10	10033	71	5204	3124	40325	277	20346	7692	93771	48613
69	5019	6	5021	9	10068	63	15166	1115	30458	200	70963	15048	24599	21681
70	5014	4	10028	23	10098	56	30202	1013	30105	740	71086	9010	114155	75654
71	0		10057	26	10045	82	15057	718	20412	522	66319	7610	97932	64084
72	5053	4	0		25075	130	10069	2468	25181	332	25225	8093	63249	37033
73	5016	4	5019	18	5044	25	10147	1931	30326	436	61098	7710	118445	102455
74	5015	4	10035	22	10098	58	10146	1224	20121	265	46003	5749	36034	36980
75	5013	4	5021	10	20106	137	10102	1542	20210	282	91047	15465	104213	41193
76	5014	4	5022	22	10024	105	15074	1253	20087	375	60954	8721	119010	64127
77	5015	4	0		5050	66	25048	2523	35334	278	50737	11046	92783	61285
78	0		5014	14	15087	88	10151	1945	35439	222	45620	15371	24169	28542
79	5020	6	5028	14	15090	101	5080	1929	35330	78	45693	6154	53120	52212
80	0		5020	19	5059	32	15138	1500	35568	205	40859	7910	113792	63828
81	5020	6	5021	12	20111	109	10195	2335	25348	374	80783	12094	108183	70498
82	5018	5	0		10103	58	15110	1400	35482	198	25427	6525	97949	78801
83	0		5016	15	5059	44	20037	1314	25326	270	76069	11168	114552	48258
84	0		5038	15	15129	85	10116	1048	25263	134	50470	8443	53778	40890
85	0		5024	14	15107	75	10084	1314	35482	329	70816	13033	73436	33390
86	5022	6	5036	14	5095	26	15178	1799	30540	220	35320	11516	87086	71786
87	0		10104	26	20176	107	10122	2118	30415	282	45940	6982	117896	70437
88	0		5016	13	10033	70	15215	2867	55375	252	70768	15075	97507	59482
89	5021	6	10032	22	20075	134	20262	2005	25317	310	35893	9054	118148	73704
90	0		5017	13	10057	86	10113	2236	20171	162	56107	6208	102559	55656
91	5019	6	5027	12	10068	70	15070	1511	30484	319	45937	13544	83131	58994
92	0		5014	10	10039	46	5127	2613	50475	84	55959	19642	56899	52773
93	5015	4	10026	20	10033	70	10105	2356	25360	163	61132	7624	92322	65737
94	5016	5	5026	14	5066	82	10121	2289	20190	316	70733	16690	97721	40852
95	5022	6	5028	12	10054	74	10173	1580	45425	230	50658	5518	118177	62488
96	0		10041	23	15110	93	10233	3275	50385	491	25351	6648	98816	37554
97	5017	5	10082	30	10066	65	15242	722	15612	232	40854	9488	83789	37412
98	5014	4	5054	22	10169	110	35239	2323	30263	536	40619	14626	113405	110932
99	5015	4	5015	14	10080	132	20121	1810	35411	308	45903	5361	73397	48352
100	5017	4	5013	14	10076	51	15193	1719	20204	178	55758	6358	118952	71725
101	5021	6	5025	12	15038	110	10111	756	20423	361	61128	8526	117784	62524
102	0		10048	18	20063	88	20259	1542	20332	505	55832	11107	72125	73452
103	5014	4	0		5051	36	25134	1924	35268	573	60826	11342	117784	111982
104	0		5032	12	15090	138	10140	1765	45420	289	71171	11053	39754	37486
105	5017	5	5028	21	15192	158	30188	1710	40375	446	55415	11972	46986	47575
106	0		5055	15	5016	58	20153	2209	20112	614	45976	7261	102217	96370
107	0		10037	22	20034	90	20074	1221	20193	366	70933	11177	108169	62254
108	5018	5	10045	24	5037	34	5097	1437	35530	110	60679	8939	99354	68028
109	0		10018	26	15058	116	10142	2430	55678	511	61363	15970	78595	34254
110	5023	4	5024	10	10075	52	15095	788	15278	323	25854	7308	112451	79009
111	0		5027	16	15126	80	25200	1532	30476	392	20452	8246	102782	80050
112	0		5015	14	10086	116	25222	1905	45551	369	25359	7520	85297	42521
113	0		5032	14	10064	110	30262	830	25314	508	60851	13408	77855	49457
114	5016	5	0		20047	92	15196	1328	15126	266	40871	11427	98166	74560
115	0		5028	16	10030	110	20079	1264	15278	349	51409	6992	88036	66224
116	0		5071	14	10028	82	10050	1603	25191	314	66142	9569	97660	39109
117	0		0		10082	46	10162	1757	15144	200	45499	11568	79646	48208
118	0		5021	22	10090	99	30088	1958	40472	376	25309	5132	79599	23062
119	0		0		10034	65	15068	2009	30265	419	30401	7091	72461	66791
120	5013	4	0		10023	68	20132	1872	20162	361	40336	15509	88034	65007
121	0		5070	14	5130	54	35319	1669	25461	544	75961	10111	82915	53833
122	0		5056	14	10105	64	20253	1383	20388	280	36217	4241	103830	65288
123	0		5026	12	5048	34	30269	1780	25294	424	36133	4672	114136	51400
124	0		0		10028	86	20128	1961	20147	308	50706	11348	97839	56977
125	5021	4	10057	26	5051	45	20192	2601	40389	436	50777	11888	117618	79745
126	0		15015	26	15062	73	25033	1418	25115	254	56123	6591	57122	63810
127	5014	4	10029	26	5163	94	20055	961	20253	224	40595	8788	44263	36877
128	5014	4	0		15148	98	15109	1156	30522	414	45746	6093	41889	52485

Experiment	Dauer und Anzahl Nachrichten													
	2		4		8		16		32		64		128	
	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#	[ms]	#
129	0		5023	13	15035	84	35251	1458	25372	539	55899	6966	67765	65945
130	0		5026	14	10062	61	15238	2687	50414	226	66201	8467	88547	53612
131	0		5022	10	5129	36	20186	1779	30216	306	40197	8717	119271	62147
132	5014	4	15021	23	15032	60	10085	2557	35312	349	70782	13525	83376	52272
133	5054	4	5059	15	10098	64	25155	2106	20250	407	56045	9603	82869	68823
134	0		10016	26	10069	53	25189	761	15223	506	15672	6047	67459	64532
135	0		0		10051	54	15208	3324	40418	202	80936	16466	113543	47857
136	5016	5	10052	24	15033	105	25307	3037	50594	374	25452	11340	78563	45941
137	0		10050	22	5135	69	10091	1513	15312	344	65816	8278	52679	39992
138	0		5024	17	25049	129	10143	1979	45388	150	60891	10361	94462	32172
139	0		5016	15	0		15065	1436	35509	304	50591	12748	73091	49466
140	5061	6	5033	10	15045	72	20097	1422	20136	579	51079	8345	92704	68474
141	0		5031	14	10081	59	15124	1332	35361	375	71140	12420	97511	68564
142	5016	4	0		15202	78	20304	1524	30355	280	60875	10856	88583	31174
143	5015	4	15021	24	5028	53	10023	1329	15212	306	70991	9707	36644	29789
144	0		5021	10	10062	134	30205	1154	30364	411	55881	8285	72950	64848
145	0		5030	13	10085	90	10058	2020	45501	334	65689	9804	87828	65836
146	0		5041	14	15082	80	10165	1812	40449	136	71267	12728	88666	55859
147	0		5026	12	15041	118	15146	1127	30575	277	71184	12712	83149	25524
148	5019	6	5050	14	15121	81	15229	1413	15264	188	46375	8590	68429	47040
149	0		15042	28	15087	114	15059	2196	50096	307	70964	8700	118757	62165
150	5015	4	5049	13	5015	58	5174	3007	30395	199	60730	12313	114868	66671
151	5033	5	10024	18	15025	82	10210	1768	15181	194	50664	6180	83611	37166
152	5020	6	5018	15	10027	72	20167	1184	30375	260	75747	13987	113420	59811
153	0		0		10123	56	20163	1685	15284	276	86184	12091	36351	49154
154	5018	5	10047	18	15116	122	20086	1158	30433	379	20628	7507	88183	55306
155	5016	4	5016	11	5031	82	15069	803	15221	322	55646	17740	26959	30541
156	0		5039	14	10130	95	10108	2203	50513	286	30553	6342	83480	44823
157	0		15079	35	5052	34	10092	2760	45512	176	30372	6533	52274	52429
158	5055	4	5025	12	10070	104	25206	2074	30248	362	45605	9599	102944	44488
159	5016	4	5031	14	10079	117	15076	990	25439	371	61086	14390	118205	87946
160	5014	4	10054	30	15126	134	20142	1671	30476	404	61022	6723	36705	46358
161	0		5024	10	10088	108	30101	1294	20253	360	55986	6065	77726	72162
162	5014	4	5023	12	15134	84	10175	2004	20513	327	51073	6486	77810	67056
163	5053	4	0		5034	37	20103	1166	30420	294	55867	14936	34913	28881
164	5019	6	5020	9	10015	110	20039	1969	25364	205	41139	4508	62631	61739
165	5021	6	5028	14	10054	58	10103	1434	35391	188	56238	8665	93504	54420
166	0		15029	26	5038	63	10098	1974	30335	124	50946	7032	112895	76909
167	0		15031	28	5119	85	5117	1682	20096	224	55703	9711	118677	59097
168	0		15022	34	10068	66	20084	1693	30275	422	60752	7162	108861	82308
169	0		5042	14	10118	126	25128	1418	25200	417	65847	9670	82812	67153
170	5021	6	10092	26	10048	124	15179	2246	20182	236	35932	4783	67697	56217
171	5029	6	15025	34	0		20069	2327	40382	250	70973	9873	98563	53363
172	0	7	0		15058	104	15110	1460	30351	434	45464	14153	102945	90392
173	0		15078	34	10053	77	15155	2061	25320	523	60514	14541	98274	38995
174	0		0		10026	66	5177	1896	20267	347	61124	7848	103271	68156
175	5019	6	5023	10	10067	59	20257	1752	40384	325	76109	13210	113611	63115
176	5028	6	5038	14	25093	148	20226	1570	40221	266	56277	14052	67667	53427
177	0		5031	11	10084	142	15225	1595	20229	212	40651	15273	87518	82715
178	5021	6	0		10037	86	25187	1994	45477	466	50805	12645	58015	64847
179	0		5049	15	15115	114	15033	2042	35443	345	50369	5983	77724	55976
180	0		0		15077	114	10127	1912	45474	280	45533	4767	77707	67144
181	0		5014	15	5065	40	15186	1771	20301	502	25549	8519	118824	83629
182	5020	6	0		10018	78	25238	2037	30357	440	50892	9444	73045	53720
183	0		10028	20	10046	118	30292	1480	15313	603	60980	6420	113598	76053
184	0		5015	14	15080	110	10171	1802	35383	146	56035	6380	41510	45713
185	0		5019	14	10033	56	20139	3547	45520	440	86105	12843	118273	76205
186	5021	6	5050	16	10054	68	20077	1298	35070	491	50030	5622	113474	76065
187	5055	6	15036	27	10084	81	10162	1605	15105	224	30378	5938	57455	58811
188	5018	5	10025	18	5020	61	20149	1281	15174	496	56107	7818	82432	63440
189	0		10059	23	10075	74	5078	2371	30328	218	41011	4986	112937	54651
190	5041	6	10042	26	15183	81	30319	1670	30365	433	30725	4246	113400	62845
191	0		5045	22	15123	84	10126	1408	35510	338	60885	11209	49178	34614
192	0		15046	28	10108	48	10235	2469	30264	190	20329	8417	114870	55631
193	5021	5	5023	14	10088	79	10088	2157	50502	259	35653	12149	107167	94979
194	5017	6	5018	8	15103	79	15204	1745	10416	196	40846	5063	103100	83315
195	0		10047	22	15052	74	10127	822	25383	354	25305	7459	93591	64205
196	0		5026	14	25027	121	15393	2050	20169	242	35472	9790	112390	48939
197	0		25039	48	20114	88	20121	1121	25482	313	51393	7004	93938	41126
198	0		5026	23	10111	57	20066	1742	15061	224	35341	8161	71413	63074
199	0		5047	15	10088	86	15069	3388	40284	182	71431	10895	112556	99136
200	5023	6	5024	15	10084	90	20119	2689	40406	228	65062	7632	112689	50894