Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften

Verteilte Systeme und Informationssysteme

Diplomarbeit

Leistungsbewertung von J2EE-Anwendungen

Entwicklung einer Plattform zur generischen Leistungsbewertung
von Anwendungen mit Mehrschichtarchitektur

Alexander Kune

alexander@kune.de
Studiengang Informatik
Matr.-Nr. 5108265

Erstgutachter: Professor Dr. N. Ritter
Zweitgutachter: Dr. G. Gryczan

Zusammenfassung

Die objektive Bewertung der Leistungsfähigkeit von Systemen ist in allen Bereichen der Wissenschaft und Wirtschaft relevant. In einer Zeit, in der die zunehmende Komplexität von Anwendungssystemen immer höhere Anforderungen an Hard- und Softwareplattformen bedingt, besteht ein entsprechender Bedarf an Kriterien und Konzepten der qualitativen und quantitativen Leistungsbewertung solcher Systeme.
Die vorliegende Diplomarbeit behandelt die Leistungsbewertung von Server-basierten Anwendungen am Beispiel der *Java 2 Platform Enterprise Edition*. Der besondere Fokus gilt hierbei den Möglichkeiten, Leistungskennzahlen für einzelne Anwendungsschichten zu ermitteln, um erkannte Schwachstellen innerhalb einer Anwendung zu lokalisieren. Um auch die Bewertung kommerzieller Produkte zu ermöglichen, wurde bei der Entwicklung der benötigten Konzepte die Rahmenbedingung eingehalten, bei Analyse und Bewertung eines Anwendungssystems auf die Verwendung des Quelltextes zu verzichten.
Im Rahmen dieser Arbeit werden zunächst Grundlagen für die Leistungsbewertung vorgestellt und diese später in Beziehung zu den Eigenschaften der Mehrschichtarchitekturen gesetzt. Nach der Erläuterung der entwickelten Konzepte für die Leistungsbewertung von J2EE-Anwendungen wird eine prototypische Umsetzung vorgestellt, um die entwickelten Konzepte zu validieren.

Abstract

The objective evaluation of performance is of relevance in all areas of science and economy. In times of software system's rising complexities demanding more and more performance of the hardware they run on, there is an according need of criteria and concepts to evaluate the quality and performance of such systems.
This diploma thesis covers the evaluation of server based applications that use the *Java 2 Platform Enterprise Edition*. In particular the possibilities of evaluating single application tiers will be addressed so that detected weaknesses in an application system can be located. For being able to evaluate commercial products the necessary concepts were developed to analyze and evaluate software systems without usage of source code.
Within the bounds of this thesis, first basic concepts of performance evaluation will be introduced, and later the possible relations to the properties of multi tier architectures will be discussed. After explaining the developed concepts for evaluating J2EE applications a prototypic implementation will be introduced to validate the elaborated concepts.

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Arbeit befasst sich mit der Leistungsbewertung Server-basierter Anwendungen am Beispiel der *Java 2 Platform Enterprise Edition* (J2EE). Der Begriff der Leistungsbewertung meint im Umfeld der elektronischen Datenverarbeitung die Ermittlung und Interpretation leistungsbezogener Kennzahlen. Die Leistungsbewertung der Hardwaresysteme ist in diesem Zusammenhang bereits intensiv untersucht worden. Obwohl die Leistungsfähigkeit der verfügbaren Hardwareplattformen bezüglich Rechenleistung sowie flüchtigem und nichtflüchtigem Speicher immer weiter zunimmt, bewegen sich moderne Betriebs- und Anwendungsplattformen und -systeme in ihren Anforderungen an die Hardware stets in den oberen Bereichen des technisch Machbaren. Die daraus ersichtliche Relevanz von Maßnahmen zur Leistungsbewertung der entsprechenden Softwarekomponenten führte zu einer Vielzahl von Lasttest- und Profilingkonzepten, die es ermöglichen, den Leistungsbedarf einer Anwendung während der Entwicklung zu verfolgen und Hardwareplattformen für die einzusetzenden Softwareprodukte zu optimieren. Die Idee für diese Arbeit entstand ursprünglich aus einer Anfrage im Rahmen einer angehenden Industriekooperation mit dem Arbeitsbereich Verteilte Systeme und Informationssysteme (VSIS) der Fakultät für Mathematik, Informatik und Naturwissenschaften der Universität Hamburg. Es ging in dieser Anfrage darum, zu ermitteln, welche Möglichkeiten bestehen, eine neu entwickelte Web-Schnittstelle einer mehrschichtigen, Server-basierten Anwendung bezüglich ihrer Leistungsfähigkeit mit der Vorgängerversion zu vergleichen. In Zusammenarbeit mit Professor Ritter haben Martin Husemann und Christian Kunze daraufhin ein Grundkonzept zur isolierten Bewertung von Anwendungsschichten in bestimmten Anwendungssystemen entwickelt. Dieses Grundkonzept dient als Ausgangspunkt für die vorliegende Diplomarbeit. Die Arbeit untersucht, welche (teilweise zu entwickelnden) Konzepte geeignet sind, einzelne Anwendungsschichten eines Anwendungssystems auf möglichst generische Weise isoliert und ohne Rückgriff auf den Quelltext bezüglich ihrer Leistungsfähigkeit zu untersuchen. Hierfür ist es erforderlich, Kennzahlen zu definieren und deren Ermittlung konzeptionell zu erfassen, die es zulassen, Anwendungssysteme sowie einzelne Schichten dieser Systeme zu bewerten und innerhalb einer gemeinsamen Domäne miteinander zu vergleichen.
Im Rahmen dieser Arbeit wurden Konzepte zur generischen Leistungsbewertung von Server-basierten Anwendungen mit Mehrschichtarchitektur geschaffen. Der besondere wissenschaftliche Anreiz lag hierbei darin, zu ermitteln wie weit die Analyse deskriptiver Elemente sowie des strukturellen Aufbaus eines Anwendungssystems Einblicke in solch ein System zulassen. Darüber hinaus galt es zu ergründen, ob sich die notwendigen Schritte hin zu der Leistungsbewertung eines Anwendungssystems automatisieren lassen. Abschließend waren die entstandenen Konzepte durch eine prototypische programmatische Umsetzung zu validieren.

1.1 Motivation

Nach allgemeiner Auffassung handelt es sich bei der Leistungsfähigkeit eines Produktes um ein wichtiges Qualitätsmerkmal. Häufig wird die Leistungsfähigkeit sogar höher bewertet als andere Qualitätsmerkmale wie Langlebigkeit oder Funktionsumfang. Die Bewertung von Leistung ist normalerweise umständlich und mit hohem Aufwand verbunden. Darüber hinaus sind die Ergebnisse einer Leistungsbewertung teilweise wenig aussagekräftig, da nicht immer abschließend geklärt ist, wie ermittelte Leistungskennzahlen zu interpretieren sind. Außerdem geben die Leistungskennzahlen vieler Bewertungsmethoden keinen Aufschluss über ihre strukturelle

Zusammensetzung. Auf Anwendungssysteme der elektronischen Datenverarbeitung bezogen bedeutet das, dass die Ermittlung von Leistungskennzahlen eine nicht triviale Aufgabe ist, die im Allgemeinen viel Wissen über den Aufbau einer Anwendung erfordert. Außerdem ist die Interpretation der ermittelten Kennzahlen wie Durchsatz und Antwortzeit häufig schwierig, da für verschiedene Ansätze unterschiedliche oder im schlimmsten Fall gar keine Definitionen für diese Kennzahlen vorliegen. Im günstigsten Fall sind anhand von Leistungskennzahlen Aussagen über Schwachstellen zu treffen, eine Lokalisierung dieser Schwachstellen innerhalb der Struktur des Anwendungssystems ist aber normalerweise nicht möglich.
Die beschriebene Situation begründet den Bedarf an wohldefinierten formalen Konzepten und Kennzahlen für die Leistungsbewertung von Anwendungssystemen. Entsprechend der genannten Anforderungen müssen diese Konzepte die Analyse und Bewertung von Anwendungssystemen automatisieren oder wenigstens teilautomatisieren. Um auch kommerzielle Systeme analysieren und bewerten zu können, ist es darüber hinaus erforderlich, auf die Verwendung von Anwendungsquelltexten zu verzichten. Außerdem müssen die entwickelten Konzepte in der Lage sein, einzelne Anwendungsschichten isoliert zu betrachten und zu bewerten, um aufgedeckte Schwachstellen zu lokalisieren.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist, zunächst einen formalen Zugang zur Leistungsbewertung von Anwendungen, insbesondere solchen mit Mehrschichtarchitektur, zu schaffen. Darüber hinaus werden die Möglichkeiten der Leistungsbewertung mehrschichtiger, Server-basierter Anwendungssysteme insbesondere innerhalb der *Java 2 Platform Enterprise Edition* bezüglich einzelner Anwendungsschichten sowie der Gesamtsysteme ohne Zugang zum Quelltext ergründet und in entsprechende Konzepte gefasst. Abschließend werden wichtige Aspekte der entwickelten Konzepte programmatisch umgesetzt.

1.3 Thematische Eingrenzung

Die vorliegende Arbeit soll dem Leser eine möglichst breite konzeptionelle Basis für die generische Leistungsbewertung von Server-seitigen Anwendungen mit Mehrschichtarchitektur zur Verfügung stellen. Aufgrund der bislang relativ unvollständigen formalen Erfassung werden daher die Grundlagen der Leistungsbewertung zwar formal sehr präzise aber bezüglich der Anwendbarkeit möglichst allgemein formuliert. Der Anspruch auf universelle Anwendbarkeit kann in der Kürze der Zeit einer Diplomarbeit allerdings nicht umfassend für die ganze Arbeit aufrecht erhalten werden. Aus diesem Grund verengt sich der Fokus in den späteren Kapiteln immer stärker auf die Betrachtung einer speziellen Anwendungsklasse. Die Ermittlung von Leistungskennzahlen Server-basierter J2EE-Anwendungen mit Mehrschichtarchitektur bildet in dieser Hinsicht die Kernthematik dieser Arbeit. Insbesondere gilt der Anspruch der Quelltext-losen Analyse und der isolierten Betrachtung einzelner Anwendungsschichten. Der akademische Anspruch einer Diplomarbeit gilt in erster Linie der Ergründung eines Themenbereiches sowie der Entwicklung neuer Konzepte, die das wissenschaftliche Verständnis dieses Bereichs erhöhen. Aufgrund der Komplexität der Thematik ist im Rahmen dieser Arbeit kein Leistungsbewertungssystem mit vollständigem Funktionsumfang als Umsetzung aller entwickelten Konzepte entstanden. Vielmehr wurden exemplarisch einige wichtige Konzepte prototypisch implementiert, um einerseits eine programmatische Grundlage für Folgeprojekte zu bilden, die sich aber andererseits auch zu einem funktionierenden Demonstrationssystem zusammenfügen lassen.

1.4 Aufbau der Arbeit

In dieser Diplomarbeit wird zunächst die Thematik der Leistungsbewertung relativ allgemein ergründet. Mit dem Fortschreiten der Arbeit verengt sich der Fokus immer weiter auf die eigentliche Kernthematik, die Leistungsbewertung von Server-basierten Anwendungssystemen mit Mehrschichtarchitektur am Beispiel der *Java 2 Platform Enterprise Edition*. Dieser Abschnitt dient dem Leser als Orientierungshilfe und soll gemeinsam mit den vorangegangenen Abschnitten einen präzisen Gesamteindruck von der vorliegenden Arbeit vermitteln.

Im folgenden Kapitel 2 werden die Grundlagen der Leistungsbewertung und die Konzepte von Mehrschichtarchitekturen vorgestellt, sowie bereits umgesetzte Konzepte der Leistungsbewertung beschrieben und von dieser Arbeit abgegrenzt. Darüber hinaus wird auf die Möglichkeiten der Analyse Java-basierter Anwendungen eingegangen, die eine wichtige Rolle in den späteren Kapiteln dieser Arbeit spielen. Zunächst werden die unterschiedlichen Möglichkeiten des Einsatzes von Leistungsbewertungskonzepten in den verschiedenen Systemen der elektronischen Datenverarbeitung erläutert und die Leistungsbewertung von Anwendungssystemen fokussiert. Im Weiteren wird das Konzept der Leistungsbewertung von dem des Lasttests abgegrenzt und darauf hin eine vierstufige Methodik der Leistungsbewertung vorgestellt, die für diese Arbeit von grundlegender Bedeutung ist. Der Kernteil dieses Abschnitts befasst sich mit der Definition und Erläuterung von Kennzahlen der Leistungsbewertung, die die formale Basis des später vorgestellten Leistungsbewertungskonzepts bilden. An dieser Stelle werden bereits einige besondere Möglichkeiten der Leistungsbewertung von Anwendungssystemen mit Mehrschichtarchitektur gegenüber monolithischen Architekturen herausgearbeitet. Abschließend werden einige Messergebnisse vorgestellt, die mit einer für diese Arbeit entwickelten Anwendung ermittelt wurden, um das Verhalten der wesentlichen, vorher definierten Kennzahlen unter zunehmenden Lastverhältnissen darzustellen und wichtige Zusammenhänge aufzuzeigen. Darüber hinaus werden die Konzepte der Mehrschichtarchitekturen vorgestellt und deren Entwicklung aus der Client-Server-Architektur verfolgt. Außerdem wird der Begriff der Schicht für die weitere Verwendung in dieser Arbeit definiert sowie die populärsten Vertreter der Mehrschichtarchitekturen vorgestellt und erläutert. Darauf aufbauend werden die beiden wichtigsten Plattformen für die Entwicklung Server-basierter Anwendungen mit Mehrschichtarchitektur, die *Java 2 Platform Enterprise Edition* von Sun und das *.NET-Rahmenwerk* von Microsoft beschrieben und miteinander verglichen, um Konzeption und Entwicklung einer Java basierten Lösung in Kapitel 3 und 4 argumentativ zu untermauern. Außerdem werden zwei Open-Source-Rahmenwerke zur Bewertung Java-basierter Anwendung vorgestellt, deren Konzepte für diese Arbeit teilweise übernommen und erweitert wurden. Abschließend werden Ansätze zur Quelltext-losen Analyse von Java-Klassen erläutert, die insbesondere für die Analyse von einzelnen Modulen einer J2EE-Anwendung von Bedeutung sind.

In Kapitel 3 wird das für diese Arbeit entwickelte Konzept eines Leistungsbewertungssystems für Server-basierte J2EE-Anwendungen mit Mehrschichtarchitektur vorgestellt. Dazu werden zunächst die Anforderungen an solch ein Leistungsbewertungssystem herausgearbeitet, um im Weiteren die gekapselte Betrachtung einzelner Anwendungsschichten mit Hilfe von Driver-, Stub- und Filter-Komponenten zu erläutern. Darauf hin werden die möglichen Kompositionen dieser Komponenten mit den Schichten eines typischen Server-basierten Anwendungssystems vorgestellt und die Aufgaben und Anforderungen sowie mögliche Umsetzungen und Lösungen der genannten Anforderungen bezüglich der einzelnen Kombinationen erläutert und deren Relevanz für diese Arbeit ermittelt. Außerdem werden Methoden zur Analyse von Anwendungsschichten einer J2EE-Anwendung behandelt. Hierbei wird insbesondere auf die Analyse von Web- und Enterprise-Java-Beans-Modulen eingegangen, da diese für die beispielhafte Implementierung der vorgestellten Konzepte in Kapitel 4 von grundlegender Bedeutung sind. Darüber hinaus werden auch die Möglichkeiten der Analyse der Datenhaltungsschicht beziehungsweise der Komponenten, die den Zugriff auf die Datenhaltung ermöglichen vorgestellt. Im weiteren Verlauf werden

Konzepte der Erstellung und Konfiguration von Anfragefolgen für die Durchführung einer Leistungsbewertung vorgestellt. Insbesondere wird hierbei auf das bereits angesprochene Konzept der Filter-Komponenten zurückgegriffen, das geeignet ist, tatsächliche Arbeitsabläufe eines Anwendungssystems aufzuzeichnen. In den letzten Abschnitten von Kapitel 3 werden die Möglichkeiten und Grenzen der Durchführung von Testläufen und der Ergebnisaufbereitung sowie -auswertung aufgezeigt. Dazu wird beschrieben, wie sich Driver-, Filter- und Stub-Komponenten in Server-basierten J2EE-Anwendungen mit Mehrschichtarchitektur zu Testumgebungen verknüpfen lassen, die in der Lage sind, die anfangs vorgestellten Kennzahlen der Leistungsbewertung sowohl für die Gesamtkomposition der Anwendung als auch für ihre einzelnen Anwendungsschichten zu ermitteln.
Kapitel 4 beschreibt die programmatische Umsetzung der in Kapitel 3 vorgestellten Konzepte und Ansätze. Hierfür wird zunächst die Gesamtarchitektur erläutert und auf diese Weise dargestellt, wie die einzelnen Komponenten zusammenhängen. Außerdem werden die Komponenten jeweils vorgestellt und Besonderheiten hervorgehoben sowie Entwurfsentscheidungen begründet. Der entstandene Prototyp ist in der Lage, Leistungskennzahlen für Server-basierte J2EE-Anwendungssysteme mit Web-Schnittstelle zu ermitteln und erlaubt insbesondere die isolierte Betrachtung und Bewertung der Web-Benutzungsschnittstelle. Er dient als Verifikation für die prinzipielle Umsetzbarkeit der in Kapitel 3 vorgestellten Konzepte.
In Kapitel 5 erfolgt eine kurze Zusammenfassung der Arbeit. Die erarbeiteten Ergebnisse werden bewertet und mit der ursprünglichen Zielsetzung verglichen. Aus den entwickelten Konzepten und der Umsetzung gefolgerte Schlüsse werden präsentiert und neu aufgetretene Fragestellungen dargestellt. Abschließend wird beschrieben, welche möglichen Erweiterungen denkbar sind, und an welchen Stellen weitere wissenschaftliche Projekte und Ausarbeitungen ansetzen könnten.

2 Grundlegende Konzepte und Technologien

In den folgenden Abschnitten werden Konzepte und Technologien erläutert, die für den Entwurf des Leistungsbewertungssystems in Kapitel 3 von grundlegender Bedeutung sind. Zunächst werden hier bestehende Leistungsbewertungskonzepte kurz erläutert, deren Bedeutung diskutiert und einige grundlegende Begriffe zur Leistungsbewertung von Anwendungssystemen festgelegt. Anschließend werden die historische Entwicklung und die Konzepte der Mehrschichtarchitektur untersucht. In diesem Zusammenhang werden auch Eigenschaften der Middlewaresysteme J2EE und .NET, soweit sie für diese Arbeit von Bedeutung sind, angesprochen und einander gegenübergestellt. Im weiteren Verlauf werden Konzepte der Leistungsbewertung in Java sowie der Java 2 Platform Enterprise Edition vorgestellt, sowie die Mittel der Quelltext-losen Analyse von Anwendungssystemen in diesem Umfeld geprüft.

2.1 Leistungsbewertungskonzepte

Bevor man sich der Fragestellung zuwendet, wie Leistungsbewertung von Hard- oder Software konzeptionell durchzuführen ist, gilt es zunächst festzustellen, auf welcher Ebene die Bewertung stattfinden soll. In den folgenden Abschnitten wird jeweils kurz erläutert, wie sich eine Leistungsbewertung auf verschiedene Systeme der elektronischen Datenverarbeitung beziehen kann. Des Weiteren wird auf Konzepte der Leistungsbewertung von Anwendungssystemen eingegangen. Insbesondere werden die für eine Leistungsbewertung relevanten Kennzahlen formal entwickelt und vorgestellt.

2.1.1 Leistungsbewertung von Hardwareplattformen und -komponenten

Die Bewertung der Leistungsfähigkeit von Hardwarekomponenten und -plattformen ist die älteste Disziplin der Leistungsbewertung in der elektronischen Datenverarbeitung. Bereits für den als erster frei programmierbarer Computer geltenden Zuse Z1, der von Konrad Zuse im Jahr 1938 fertiggestellt wurde, existieren Messungen über seine Rechenleistung. So wird die mittlere Rechengeschwindigkeit mit einer Multiplikation in etwa fünf Sekunden bei einer Taktfrequenz von einem Hertz angegeben [Zus99]. Die Leistungsfähigkeit einer Hardwareplattform ergibt sich aus der Leistungsfähigkeit und dem Zusammenspiel der einzelnen Komponenten. Die zentralen Komponenten einer Hardwareplattform sind die Verarbeitungseinheit (CPU), der flüchtige Arbeitsspeicher (RAM) und der nichtflüchtige Festspeicher, normalerweise als Magnetspeicher in Form von Festplatten. Die Leistung einer CPU wird in Form ihres Durchsatzes, das heißt in diesem Fall der Anzahl von Operationen in einem bestimmten Zeitraum angegeben. Die wichtigsten Kennzahlen sind in diesem Zusammenhang die geleisteten MIPS (Million Instructions per Second) und FLOPS (Floating Point Operations Per Second). Im Linux-Kernel wird zusätzlich das auf MIPS basierende Maß BogoMips zu Kalibrierungszwecken beim Booten ermittelt [Dor04]. Die wichtigsten industriellen Standards zur Messung der CPU-Leistung stammen von der Standard Performance Evaluation Company (SPEC), deren aktueller CPU2000-Benchmark Version 1.3 entworfen wurde, um unterschiedliche Computersysteme hinsichtlich ihrer Rechenleistung vergleichbar zu machen. Bei den Messungen werden Ganzzahl- und Gleitpunktoperationen getrennt betrachtet [SPEC06]. Für die Leistung von Arbeitsspeicher sind die Speicherkapazität, der Durchsatz und die Zugriffszeit von Bedeutung, wobei der Durchsatz außer von der Architektur des Speicherbausteins zusätzlich von der Taktung des Datenbusses abhängt. Festplatten werden im

Wesentlichen ebenfalls anhand ihrer Speicherkapazität, der maximalen Übertragungsrate und der Zugriffszeit bewertet. Die maximale Übertragungsrate hängt hierbei aufgrund der Architektur von der Rotationsgeschwindigkeit und der Datendichte sowie der Zylinderanzahl des Datenträgers ab. Zusätzlich ergibt sich aus dem maximalen Durchsatz der Datenanbindung eine obere Grenze für die tatsächliche Übertragungsrate von Festplatten. Dies spielt insbesondere dann eine Rolle, wenn mehrere Festplatten über den selben Datenbus angebunden sind. Bei der Bewertung von nichtflüchtigen Speicherkomponenten sind außerdem Datensicherheit und Lebensdauer von besonderer Bedeutung. Bei Festplatten führt beispielsweise das mechanische Aufsetzen des Schreib-Lesekopfes (Head Crash) in aller Regel zu irreparablen Fehlern und damit zu Datenverlust. Die durchschnittliche Anzahl von Betriebsstunden einer Festplatte vor dem Ausfall wird als *MTTF* (MeanTime To Failure) bezeichnet.
Da jede auf einem Computersystem erbrachte Leistung unabhängig davon, in welcher Ebene sie angefordert wird, letztlich in den zu Grunde liegenden Hardwarekomponenten erbracht wird, bezieht sich jede Leistungsbewertung in höheren Systemebenen stets auf die verwendete Hardwareplattform.

2.1.2 Leistungsbewertung von Betriebssystemen

Die Hauptaufgabe eines Betriebssystems besteht darin, eine Abstraktionsschicht zwischen den Anwendungen und der Hardwareplattform zu schaffen und Anforderungen von Anwendungen gegebenenfalls durch die Nutzung von Hardwaretreibern an die Hardwarekomponenten zu leiten. Moderne Betriebssysteme leisten darüber hinaus die Verwaltung von Prozessen und Threads sowie des Arbeitsspeichers und virtuellen Arbeitsspeichers. Die Leistungsfähigkeit eines Betriebssystems bemisst sich deshalb darin, wie effizient die angebotenen Leistungen bezüglich einer bestimmten Hardwareplattform erbracht werden. Wichtige Eckdaten sind in diesem Zusammenhang die maximale Prozess- und Threadanzahl und die maximale Größe des adressierbaren Arbeitsspeichers. Da Hardwaretreiber als Erweiterung des Betriebssystems normalerweise vom Hardwarehersteller zur Verfügung gestellt werden, lässt die Effizienz der Umsetzung von Anwendungsanforderungen in Hardwareanforderungen nicht unbedingt Rückschlüsse auf die Leistungsfähigkeit eines Betriebssystems zu.
Ähnlich wie bei der Hardwareplattform spielen die Faktoren des Betriebssystems bei der Leistungsbewertung höherer Systemebenen häufig eine Rolle, beispielsweise profitiert eine Anwendung, die auf dem massiven Einsatz von Threads basiert, stark von einer effizienten Threadverwaltung des Betriebssystems. Daher sind auch hier die entsprechenden Messwerte nur im bestimmten Kontext, das heißt mit auf der selben Plattform ermittelten Werten, vergleichbar.

2.1.3 Leistungsbewertung von Middleware-Systemen

Middleware-Systeme erweitern ein Betriebssystem um komplexe funktionale Aspekte, die für die Entwicklung verteilter Anwendungen notwendig oder hilfreich sind. Sie stellen häufig Laufzeitumgebungen für die entsprechenden Anwendungssysteme zur Verfügung und erbringen darüber hinaus normalerweise Namens- und Kommunikationsdienste und ermöglichen entfernte Methodenaufrufe. Häufig gehören auch Präsentationsdienste (zum Beispiel in Form eines Webservers, der in der Lage ist, dynamisch HTML-Dokumente zu erzeugen) und Persistenzdienste zum Funktionsumfang von Middleware-Systemen. Die Leistungsfähigkeit von Middleware-Systemen objektiv einzuschätzen, ist eine nicht triviale Aufgabe. Zusätzlich zu der Effizienz der von einem Middleware-System angebotenen Dienste, spielt auch der programmiertechnische

Aufwand, der nötig ist um bestimmte Aufgaben umzusetzen, bei der Bewertung eines Middleware-Systems eine große Rolle. Dass Aussagen über notwendige Quelltextlänge und Effizienz im Vergleich unterschiedlicher Middleware-Systeme aber nicht immer objektiv sind, zeigt der Versuch von Microsoft, die Überlegenheit seiner .NET-Architektur gegenüber Suns Java 2 Plattform Enterprise Edition mit einer besonders effizienten und Quelltext-sparsamen Version der Petstore-Anwendung zu demonstrieren [Mic06].

2.1.4 Leistungsbewertung von Anwendungsystemen

Die Leistungsbewertung einer Anwendung dient der Ermittlung von Kennzahlen, die Aussagen über Durchsatz, Speicherbedarf, Aufrufpotenzierung und Skalierbarkeitsschwellen einer Anwendung in verschiedenen Situationen erlauben, indem diese bezüglich der Umgebung, in der sie ermittelt wurden, interpretiert werden. Dabei sind grundsätzliche Unterschiede bei der Bewertung von Backend-Anwendungen einerseits und interaktiven Anwendungen andererseits zu beachten. Während im Stapelverarbeitungsbetrieb mit Leistung in erster Linie der Durchsatz, das heißt, die Anzahl der Transaktionen in einem bestimmten Zeitraum, gemeint ist, gilt das Interesse bei interaktiven Anwendungen vorrangig der Antwortzeit im Mehrbenutzerbetrieb. Unabhängig von der Art des Anwendungssystems ist bei einer in Schichten oder Komponenten organisierten Architekturform die jeweilige Verweilzeit in den Schichten beziehungsweise Komponenten von besonderer Bedeutung.

2.2 Methoden der Leistungsbewertung

Die Leistung eines Systems erschließt sich durch die Beobachtung dieses Systems während seiner normalen Aktivitäten. Da eine nicht-invasive Beobachtung eines Systems nicht möglich ist, beeinflusst jeder derartige Prozess das Verhalten und damit auch die Leistung solch eines Systems. Die Kunst einer Leistungsbewertungmethodik besteht folglich zunächst darin, die Beobachtung eines Systems zu ermöglichen, ohne seine Leistung nennenswert zu verändern oder wenigstens die Einflüsse, die die Leistungsbewertung auf das System hat, zu bemessen.
In den folgenden Abschnitten werden zunächst Lasttests und Leistungsbewertung voneinander abgegrenzt, um später eine Methodik der Leistungsbewertung vorzustellen, die im Rahmen dieser Arbeit aus einem Lasttestkonzept entwickelt wurde.

2.2.1 Lasttests und Leistungsbewertung

In der Literatur werden Lasttests und Leistungsbewertung nicht strikt unterschieden. Für eine klare Abgrenzung werden für diese Arbeit Lasttests und Leistungsbewertung wie folgt definiert. Ein Lasttest ist die Prüfung, ob ein System unter definierten Bedingungen eine geforderte Mindestleistung erbringt, indem bestimmte Kennzahlen ermittelt und diese dann mit dem gewünschten Ergebnis verglichen werden. Ein typisches Beispiel für Lasttests in diesem Sinne sind die TPC-Performanztests [Smi01]. Bei der Leistungsbewertung hingegen geht es primär darum, Kennzahlen zu ermitteln, die ein Anwendungssystem bezüglich seiner Leistungsfähigkeit in unterschiedlichen Situationen charakterisieren. Obwohl die Leistungsbewertung der allgemeinere Ansatz ist, lassen sich die Konzepte zur Ermittlung von Kennzahlen in Lasttests für die Leistungsbewertung anpassen.

2.2.2 Methodik der Leistungsbewertung

In [ZAO02] wird eine vierstufige Methodik für Lasttests vorgestellt. Sie umfasst die Spezifikation der relevanten Leistungskriterien, die Analyse und Simulation des Anwendungseinsatzes, das Erstellen der Prüfmethode und die Durchführung der gewählten Tests. Diese Methodik eignet sich konzeptionell für den Einsatz der Leistungsbewertung im Sinne dieser Arbeit. Die angepassten vier Stufen werden in den folgenden Abschnitten jeweils kurz erläutert.

Spezifikation der relevanten Leistungskriterien

Im ersten Schritt müssen die Kriterien wie Durchsatz und Antwortzeit festgelegt werden, die für die Leistungsbewertung einer spezifischen Anwendung relevant sind. Diese Kennzahlen der Leistungsbewertung werden in Abschnitt 2.3 vorgestellt. Im Rahmen von Lasttests werden für die relevanten Kennzahlen normalerweise akzeptable minimale oder maximale Werte festgelegt, auf deren Einhaltung eine Prüfmethode das Anwendungssystem testet.

Analyse und Simulation des Einsatzes

Nachdem die relevanten Leistungskriterien festgelegt wurden, ist es erforderlich, den Einsatz eines Anwendungssystems zu Analysieren, um realistische Simulationsabläufe zu ermöglichen. Simulationsabläufe werden in *Testplänen* definiert. Ein solcher Testplan enthält alle für einen bestimmten Testlauf notwendigen Informationen. Dabei handelt es sich im Rahmen dieser Arbeit um eine Folge von Leistungsanforderungen, die einen Anwendungsfall eines Anwendungssystems darstellt. Die funktionale Abdeckung spielt bei der Erstellung von Testplänen eine besondere Rolle, das heißt, dass sich nach Möglichkeit alle Anwendungsfälle, die bei der Benutzung eines zu testenden Anwendungssystems auftreten, in entsprechenden Testplänen wiederfinden sollten. Da eine vollständige funktionale Abdeckung vollautomatisch normalerweise nicht erreicht werden kann, und sich eine manuelle Erfassung im Aufwand proportional zu der Anzahl der möglichen Anwendungsfälle verhält, bietet sich für die Generierung von Testplänen ein Filtermechanismus an, der in der Lage ist, den Umgang eines Anwenders mit einer Anwendung zur späteren (gegebenenfalls parametrisierten) Verwendung aufzuzeichnen.

Erstellung der Prüfmethode

Die Erstellung der Prüfmethode umfasst die Auswahl, Konfiguration und Komposition der durchzuführenden Tests. Hier werden Wiederholungsanzahlen bestimmt und Denk- und Erholungszeiten festgelegt, das heißt die Zeitintervalle zwischen einzelnen Prüfschritten und Prüfungsdurchgängen. Im Wesentlichen werden in diesem Schritt also relevante Testpläne ausgewählt, soweit es nötig ist konfiguriert und zu einem ausführbaren Gesamttest zusammengefügt.

Durchführung der Prüfung

Bei der Durchführung der Prüfung werden die im vorigen Schritt zusammengestellten und konfigurierten Testpläne ausgeführt, um die gewünschten Leistungskennzahlen zu ermitteln. Während einer Prüfung sollte der normale Anwendungsbetrieb eingestellt werden. Ist aufgrund der Prüfung mit Seiteneffekten in der Datenbasis des zu prüfenden Anwendungssystems zu

rechnen, müssen vorher entsprechende Sicherungsmaßnahmen getroffen werden. Alternativ ist die Einrichtung einer Testumgebung möglich, in der das Anwendungssystem von der Produktivanwendung entkoppelt erneut aufgesetzt wird. Hierbei ist allerdings darauf zu achten, dass realistische Bedingungen bezüglich der beeinflussenden Systemebenen geschaffen werden, damit die Ergebnisse Rückschlüsse auf das eigentliche Produktivsystem zulassen.

2.3 Kennzahlen der Leistungsbewertung

In der Literatur werden die Antwortzeit und der Durchsatz als die wesentlichen Kennzahlen von Lasttests genannt. Die Antwortzeit ist für interaktive Anwendungen von besonderer Bedeutung. Bei ihr handelt es sich um den Zeitraum, der vom Senden einer Anfrage durch eine Client-Komponente bis zum vollständigen Erhalt einer Antwort von einer Server-Komponente verstreicht. Insbesondere also auch der Zeitraum, der nach dem Auslösen einer Aktion durch einen Benutzer einer interaktiven Anwendung verstreicht, bis ein Ergebnis dieser Aktion vorliegt. Dem gegenüber steht der Durchsatz, der normalerweise angibt, wie viele Transaktionen pro Sekunde eine Anwendung leistet. Hierbei ist hervorzuheben, dass der Durchsatz nicht die Geschwindigkeit einer Anwendung misst. Vielmehr handelt es sich um ein Maß für die Kapazität einer Anwendung. [ZAO02]
Ausgehend von den Definitionen der beiden Kennzahlen für Lasttests wurden für diese Arbeit fünf Kennzahlen der Leistungsbewertung abgeleitet, die im Folgenden vorgestellt werden. Dabei soll von Transaktionen sowie synchronen und asynchronen Methodenaufrufen zunächst in Form von *Leistungen* abstrahiert werden.

2.3.1 Ressourcenbedarf

Der Ressourcenbedarf einer Leistung bezeichnet die Menge der zu ihrer Erbringung erforderlichen Ressourcen. Diese umfassen unter anderem Hardwareressourcen wie Rechenleistung, Arbeits- und Festplattenspeicher oder Anforderungen an das Netzwerk sowie Softwareressourcen wie Dateisperren oder die Anzahl benötigter Threads. Insofern handelt es sich bei dem Ressourcenbedarf einer Leistung nicht um eine einzelne Kennzahl, sondern um eine Menge von Kennzahlen, die diese Leistung bezüglich ihrer Anforderungen charakterisieren.
In dieser Arbeit wird zwischen dem einfachen und dem kumulierten Ressourcenbedarf einer Leistung unterschieden. Der kumulierte Ressourcenbedarf oder Gesamtressourcenbedarf umfasst dabei alle Ressourcen, die verwendet werden, um eine Leistung in ihrer Gesamtheit zu erbringen. Dem gegenüber stellt der einfache Ressourcenbedarf nur die Menge an Ressourcen dar, die von der aufgerufenen Komponente zur Erbringung der untersuchten Leistung angefordert wird. Es gilt also, dass der einfache Ressourcenbedarf einer Leistung bezogen auf die aufgerufene Komponente deren Gesamtressourcenbedarf abzüglich der Menge der Ressourcen, die Leistungen anderer Komponenten beanspruchen, entspricht.

2.3.2 Antwortzeit

Entsprechend der Definition der Antwortzeit in einer Client-Server-Umgebung ist die Antwortzeit (R_P) einer Leistung, die von einer Komponente angeboten wird, die Zeit, die benötigt wird um diese Leistung zu erbringen, das heißt der Zeitraum zwischen Beginn (t_{req}) und vollständiger Erbringung (t_{res}) dieser Leistung.

$$R_P = t_{res} - t_{req}$$

Die Antwortzeit ist somit ein Maß für die Ausführungsgeschwindigkeit einer angebotenen Leistung. Je kleiner die Antwortzeit ist, desto effizienter wird die Leistung erbracht. In Komponentenumgebungen setzt sich eine Leistung (P) häufig aus der Teilleistung, die in der Komponente erbracht wird (p_0), sowie den Teilleistungen, die durch die Nutzung anderer Komponenten erbracht werden (p), zusammen.

$$P = p_0 \cup \bigcup_{p \in P \setminus p_0} p$$

In solchen Fällen ist insbesondere die *anteilige Antwortzeit* der Teilleistung, die in der eigenen Komponente erbracht wird von Interesse. Dies ist die Dauer, für die der Kontrollfluss zur Erbringung einer Leistung tatsächlich in der aufgerufenen Komponente verweilt.

$$R_{p_0} = R_P - \sum_{p \in P \setminus p_0} R_p$$

Abbildung 2.1 zeigt die genannten Zusammenhänge anhand einer einfachen Anwendung mit Web-Schnittstelle. Eine Anfrage von einem Browser löst die dynamische Erzeugung von HTML-Text im Web Container aus, welcher wiederum ein dahinter liegendes Datenbanksystem (DBS) nutzt. Die Antwortzeit des Web Containers zur Erbringung seiner Leistung ist in diesem Beispiel die Zeitspanne zwischen dem Senden der HTTP-Anfrage und dem Erhalt der HTTP-Antwort.

$$R_W = t_3 - t_0$$

Die anteilige Antwortzeit des Web Containers entspricht dieser Zeitspanne abzüglich der anteiligen Antwortzeit der Leistung des Datenbanksystems.

$$R_{w_0} = R_w - (t_2 - t_1) = t_3 - t_0 - (t_2 - t_1)$$

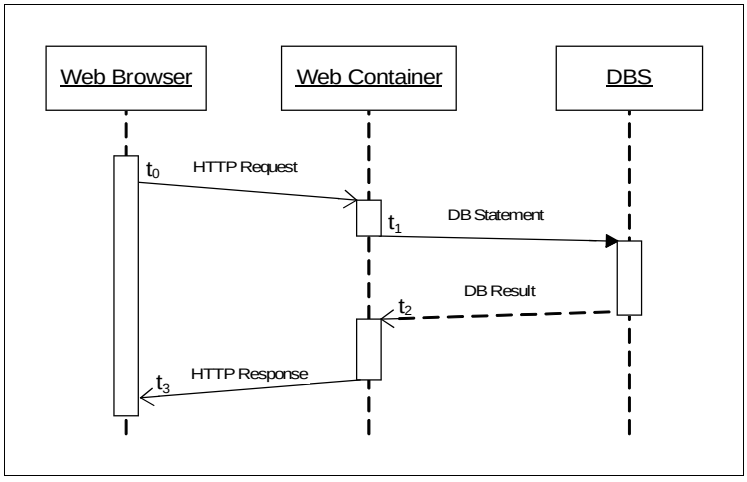

Abbildung 2.1: Interaktion zwischen Komponenten einer Web-Anwendung

2.3.3 Aufrufpotenzierung

Die Aufrufpotenzierung einer von einer Komponente angebotenen Leistung ist die Anzahl der für die Erbringung dieser Leistung erforderlichen Inanspruchnahmen von Leistungen anderer Komponenten. In dieser Arbeit wird die Aufrufpotenzierung (f_P^{pot}) als *Potenzierungsfaktor* einer

Leistung verstanden, der angibt, auf wie viele Aufrufe sich die Inanspruchnahme dieser Leistung vervielfacht. Im einfachsten Fall ist der Potenzierungsfaktor einer Leistung konstant Null.

$$f_P^{pot} = 0$$

Das bedeutet, dass beliebig parametrisierte Anforderungen dieser Leistung ausschließlich durch die anbietende Komponente erbracht werden; es werden keine Leistungen weiterer Komponenten in Anspruch genommen. Wird für unterschiedlich parametrisierte Anforderungen einer Leistung immer dieselbe Anzahl von Leistungen in anderen Komponenten beansprucht, hat diese Leistung einen konstanten Potenzierungsfaktor größer als Null.

$$f_P^{pot} \in \mathbb{N} \setminus \{0\}$$

Dieser Fall ist in Abbildung 2.2 beispielhaft für eine HTTP-Anfrage „Login“ dargestellt. Jeder Versuch eines Benutzers, sich an der Web-Schnittstelle anzumelden, führt zu genau einer Anfrage an das Datenbanksystem, um zu überprüfen, ob die angegebenen Anmeldedaten korrekt sind. Das heißt, in diesem Beispiel ist der Potenzierungsfaktor der Anmeldung an der Web-Schnittstelle konstant Eins.

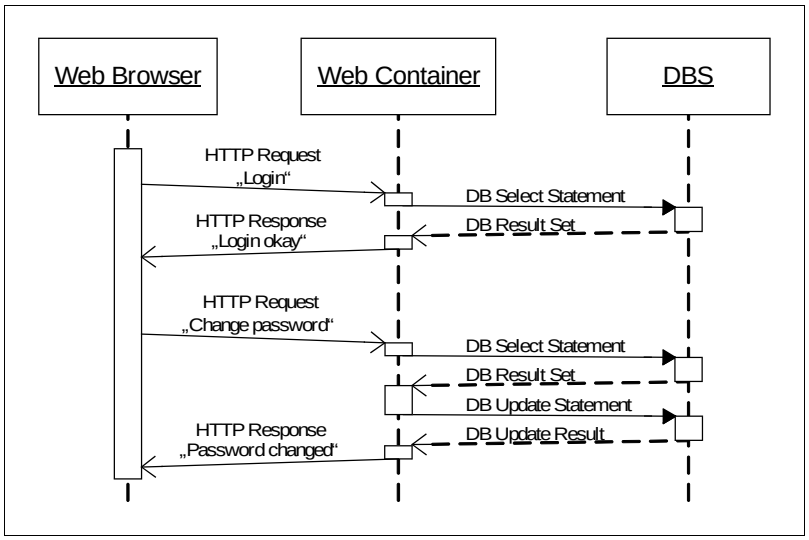

Abbildung 2.2: Konstante und dynamische Aufrufpotenzierung

Darüber hinaus ist es auch möglich, dass die Anzahl der in anderen Komponenten zu erbringenden Leistungen für eine Leistung direkt oder indirekt von den Parametern der Leistungsanfrage abhängt, zum Beispiel gilt dies für Leistungen anderer Komponenten, die in Schleifen angefordert werden, deren Zyklenanzahl von den Aufrufparametern abhängt. In diesem Fall hat die Leistung einen dynamischen Potenzierungsfaktor, von dem gegebenenfalls das Minimum, das Maximum und der Mittelwert abzuschätzen sind, falls die Ermittlung der eigentlichen Funktion zu komplex oder mangels Einsicht in die Funktionsweise der Leistungserbringung nicht möglich ist.

$$f_P^{pot} : par(P) \rightarrow \mathbb{N}$$

Die HTTP-Anfrage „Change password“ in Abbildung 2.2 hat im dargestellten Fall eine Potenzierung um den Faktor Zwei, da zum Ändern eines Kennwortes zunächst geprüft wird, ob das angegebene aktuelle Kennwort korrekt ist und danach erst das neue Kennwort an das Datenbanksystem übergeben wird. Der Potenzierungsfaktor der Kennwortänderung ist aber nicht konstant, sondern hängt von der Korrektheit des angegebenen aktuellen Kennwortes ab. Wird ein falsches aktuelles Kennwort angegeben, wird nach der Überprüfung kein neues Kennwort an das Datenbanksystem übergeben. Damit ist der Potenzierungsfaktor für diesen Fall Eins.

$$f_{chpw}^{pot} : STRING \rightarrow \{1, 2\}$$

Die Aufrufpotenzierung ist ein Komponenten-bezogenes Maß der Vermaschung einer angebotenen Leistung. Mit ihrer Hilfe lassen sich gegebenenfalls vorhandene Engpässe ermitteln. Der Ressourcenbedarf und die Antwortzeit einer Leistung hängen stark von Art und Grad der Aufrufpotenzierung dieser Leistung ab.

2.3.4 Durchsatz

Aufbauend auf der Definition des Durchsatzes als Anzahl der Transaktionen, die in einem bestimmten Zeitraum durchgeführt werden, wird der Durchsatz $T_{P,t}$ in dieser Arbeit allgemein definiert als die Anzahl erbrachter Leistungen pro Zeiteinheit. Es handelt sich genauer um die Anzahl von Ergebnissen $|res_{P,t}|$, die die mehrfache Anforderung einer Leistung in einem bestimmten Beobachtungsintervall t pro Zeiteinheit generiert.

$$T_{P,t} = \frac{|res_{P,t}|}{t}, \quad t > 0$$

Wie in [ZAO02] erläutert wird, hängt der Durchsatz davon ab, in welcher Form die Anforderungen gestellt werden. Bei einer Sequenz von Anforderungen, in der jede neue Anforderung erst abgesetzt wird, wenn die vorige Anforderung abgearbeitet wurde, ist ein Durchsatz entsprechend der Antwortzeit der Leistung, im Bereich $R_P < t$ weitgehend unabhängig von der Größe des Beobachtungsintervalls zu erwarten. In dieser Arbeit wird diese Größe als sequenziell ermittelter Durchsatz bezeichnet.

$$T^{seq}_{P,t} \approx \frac{1}{R_P}, \quad 0 < t < R_P$$

Bei einer nebenläufig abgesetzten Menge von Anforderungen hängt der Durchsatz, in dieser Arbeit als nebenläufig ermittelter Durchsatz $T^{par}_{P,t}(n)$ bezeichnet, von dem Skalierungsverhalten der angeforderten Leistung und damit auch vom *Parallelitätsgrad* n also der Kardinalität der Anforderungsmenge ab. Die entsprechende Differenz zum Durchsatz bei sequenzieller Anforderung lässt sich durch den höheren Verwaltungsaufwand, insbesondere die unter Umständen notwendige Synchronisation von Ressourcenzugriffen, sowie den höheren Speicherbedarf, aber auch die positiven Effekte in Multiprozessorumgebungen erklären. Der Durchsatz einer Leistung unterliegt einer oberen Schranke, die dem entsprechenden sequenziell ermittelten Durchsatz entspricht, korrigiert um einen Faktor C, der die eventuellen Auswirkungen einer Mehrprozessorumgebung auf die nebenläufige Ausführung berücksichtigt.

$$T^{par}_{P,t}(n) = \frac{|res_{P,t}|}{t} \leqslant T^{seq}_{P,t} \cdot C \leqslant \frac{n}{t}, \quad t > 0$$

2.3.5 Skalierungsverhalten und Kapazität

Skalierungsverhalten und Kapazität einer Leistung lassen sich anhand der Verläufe von Antwortzeit und Durchsatz bei zunehmendem Parallelitätsgrad ermitteln. Der maximale Durchsatz eines bestimmten Beobachtungsintervalls t ist immer bei dem Parallelitätsgrad zu erwarten, bei dem $t = max(R_p)$ gilt, das heißt an der Stelle, an der die Länge des Beobachtungsintervalls der maximalen Antwortzeit entspricht. Die maximale Antwortzeit ist in diesem Zusammenhang, die Dauer, nach der alle Leistungen einer Anforderungsmenge erbracht wurden. Zur Veranschaulichung dieser Faktoren ist im Rahmen dieser Arbeit eine Anwendung entwickelt worden, die in der Lage ist, die Verläufe von Durchsatz und Antwortzeit beliebiger Leistungen bei unterschiedlichen Parallelitätsgraden zu ermitteln und aufzuzeichnen. Hier werden unterschiedliche Parallelitätsgrade durch die Verwendung von Threads realisiert. Zunächst wird dazu jeder Thread für die Erbringung einer Leistung vorbereitet. Daraufhin werden alle vorbereiteten Threads möglichst gleichzeitig gestartet (in den durchgeführten Messungen lagen die Startverzögerungen bei rund 6.000 Threads zwischen einer und zwei Millisekunden). Abbildungen 2.3 und 2.4 zeigen typische Verläufe von Antwortzeit und Durchsatz, die auf einem Centrino Notebook mit einem Intel Pentium 4 1,8 GHz Prozessor und 512 MB RAM unter Windows XP (Service Pack 2) mit einer Sun Java 1.5.0_06-b05 Laufzeitumgebung ermittelt wurden. Bei der gemessenen Leistung handelt es sich in diesem Fall um das Allozieren und Befüllen von 8 MB Arbeitsspeicher. Messungen der Leistungsverläufe verschiedener Operationen (CPU bezogene Leistungen, Arbeitsspeicher bezogene Leistungen und Leistungen bezüglich des Festplattenzugriffs) auf unterschiedlichen Hardware- und Betriebssystemplattformen ergaben, dass die Antwortzeit einer Leistung mit zunehmendem Parallelitätsgrad überproportional ansteigt während der maximale Durchsatz entsprechend abnimmt. Dies ist aufgrund des bei zunehmender

Parallelität ansteigenden Verwaltungsaufwands nachvollziehbar. Die Messungen belegen außerdem den oben genannten Zusammenhang zwischen maximaler Antwortzeit und maximalem Durchsatz. Dies ist auch in Abbildungen 2.3 und 2.4 gut zu erkennen, wo der Durchsatz bei einem Beobachtungsintervall von einer Sekunde dort maximal ist, wo auch die maximale Antwortzeit eine Sekunde beträgt, nämlich bei etwa 1700 parallelen Anfragen. Für die Ermittlung repräsentativer Daten ist die Wahl eines hinreichend großen Parallelitätsintervalls von grundlegender Bedeutung, da bei einem zu klein gewählten Intervall stets auf einen linearen Verlauf geschlossen werden könnte.

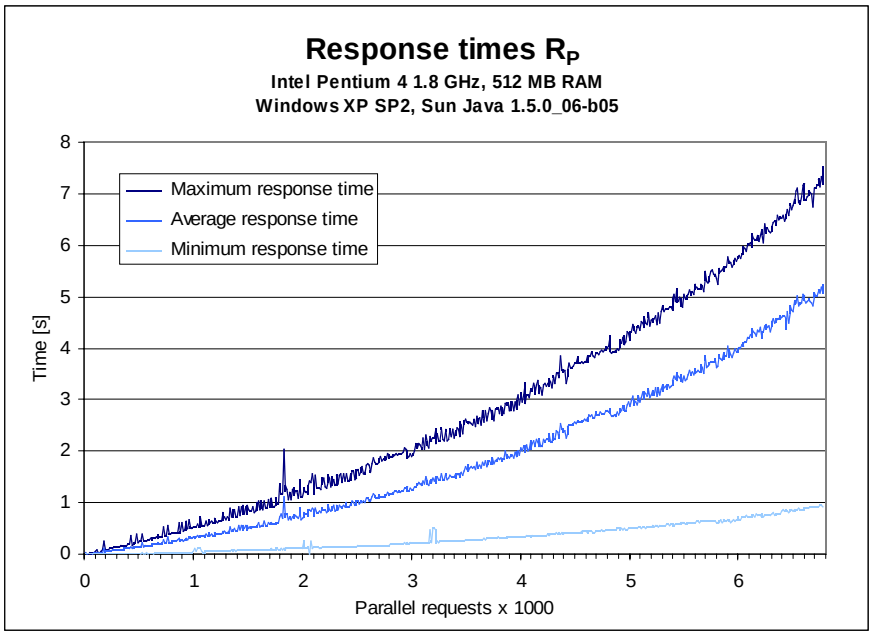

Abbildung 2.3: Verlauf der Antwortzeit von P=„Füllen von 8MB Arbeitsspeicher“

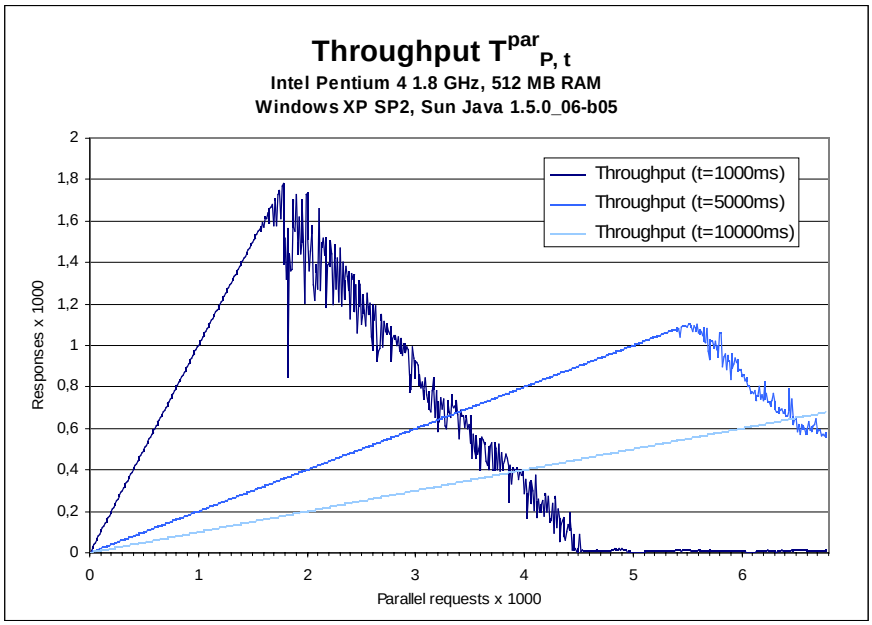

Abbildung 2.4: Verlauf des Durchsatzes von P=„Füllen von 8 MB Arbeitsspeicher“

2.4 Mehrschichtarchitekturen

Bei dem Konzept der Mehrschichtarchitektur handelt es sich um eine moderne Softwarearchitektur mit besonders günstigen Eigenschaften für den Entwurf und die Entwicklung komplexer Anwendungssysteme. Darüber hinaus wird im weiteren Verlauf dieser Arbeit gezeigt, dass sich Anwendungssysteme, die in einer Mehrschichtarchitektur entwickelt wurden, auf sinnvolle Weise bezüglich ihrer Leistungsfähigkeit prüfen lassen, indem Kennzahlen für ihre einzelnen Schichten ermittelt werden. Ein Anwendungssystem in Mehrschichtarchitektur ist in möglichst lose gekoppelte Einzelschichten partitioniert. In jeder Schicht sind Komponenten und Funktionen zusammengefasst, die verwandte Funktionalität zur Verfügung stellen oder sich auf der selben Abstraktionsebene befinden. Die Funktionalität einer Schicht wird an den Schichtgrenzen mittels einer klar definierten Schnittstelle angeboten, die nach Möglichkeit in deskriptiver Form vorliegt. In strikten Mehrschichtarchitekturen ist zudem der Zugriff nur auf Komponenten und Funktionen direkt benachbarter Schichten erlaubt.
In den folgenden Abschnitten wird zunächst die Entwicklung von Client-Server-Systemen hin zur Mehrschichtarchitektur dargestellt. Im Weiteren wird der heute populärste Vertreter der Mehrschichtarchitekturen, die Drei-Schicht-Architektur, vorgestellt. Abschließend werden die Konzepte und Grundlagen der beiden Middleware-Architekturen J2EE und .NET erläutert und deren Eigenschaften bezüglich der Entwicklung von Anwendungen mit Mehrschichtarchitektur verglichen.

2.4.1 Schichten und Ebenen

Die Begriffe Schicht und Ebene (Layer und Tier) werden in der Literatur nicht konsistent verwendet. In [Lan04] wird die „Web-basierte 4-Tier-Architektur“ vorgestellt und zwischen vier einzelnen Ebenen unterschieden. [TaSt03] benutzt die Begriffe Schicht und Ebene einerseits synonym bei der Erläuterung der Anwendungsschichten („Ebene der Benutzeroberfläche“, „Verarbeitungsebene“ und „Datenebene“), andererseits werden bei der „Zwei-Schichten-Architektur (two-tiered)“ Client und Server eines Anwendungssystems als dessen Schichten bezeichnet.
Um Inkonsistenzen zu vermeiden, wird in dieser Arbeit auf die Verwendung des Begriffes Ebene in diesem Zusammenhang verzichtet. Der Begriff Schicht bezeichnet im Rahmen dieser Arbeit ein Element einer vertikal partitionierten oder vertikal verteilten Anwendung. Die vertikale Partitionierung bezeichnet dabei den stapelartigen Aufbau einer Anwendung aus disjunkten Komponenten, die nach anwendungslogischen Gesichtspunkten zusammengefasst sind, während die vertikale Verteilung [TaSt03, S. 72] darüber hinaus die lose Kopplung zwischen diesen Komponenten und damit die Möglichkeit der räumlichen Verteilung meint. Die Funktionsweise einer Schicht ist durch die Implementierung ihrer Komponenten bestimmt. In dieser Arbeit werden die Komponenten einer Schicht weitgehend als *Black Boxes* behandelt, das heißt Implementierungsdetails, insbesondere der Quelltext bleiben verborgen.

2.4.2 Das Client-Server-Modell

In verteilten Systemen ist das Client-Server-Modell die klassische Architekturform. Bei dieser Architekturform wird die Funktionalität zwei Komponenten zugeordnet. Ein Server ist eine Komponente, die einen bestimmten Dienst implementiert. Ein Client ist eine Komponente, die einen Dienst von einem Server anfordert, indem sie eine Anfrage sendet und dann auf eine Antwort vom Server wartet. Dabei besteht zwischen Client und Server lediglich eine lose

Kopplung, so dass die Kommunikation auch über Kommunikationsnetzwerke realisiert werden kann. [TaSt03, S. 62ff]

2.4.3 Die Drei-Schicht-Architektur

Bei der Drei-Schicht-Architektur wird zwischen Präsentationsschicht, Verarbeitungsschicht und Datenhaltungsschicht unterschieden. Diese vertikale Partitionierung der Anwendung muss nicht unbedingt der vertikalen Verteilung ihrer Komponenten entsprechen. Dies hat zur Folge, dass eine lose Kopplung zwischen den Schichten nicht zwingend gefordert ist.

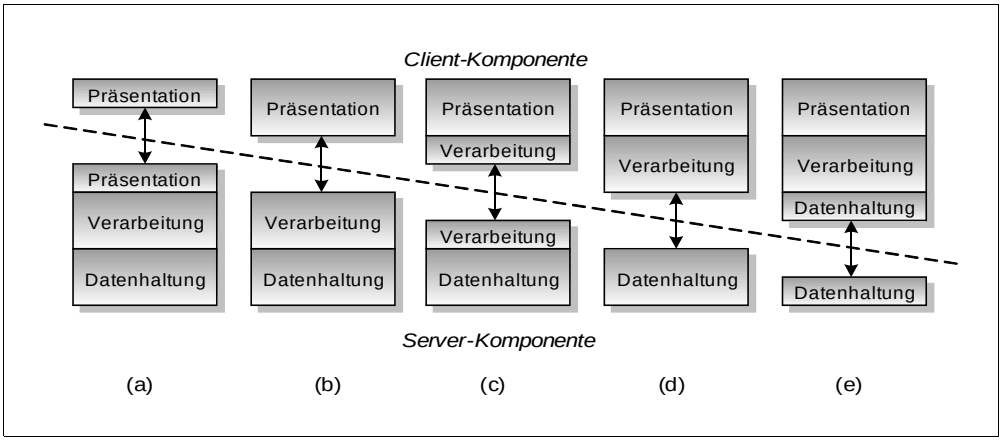

Abbildung 2.5: Alternative Client-Server-Anordnungen

Häufig wird die Drei-Schicht-Architektur mit dem Client-Server-Modell kombiniert. Die sich hieraus ergebenden alternativen Anordnungsmöglichkeiten der einzelnen Schichten sind in Abbildung 2.5 [TaSt03, S.71] dargestellt. Neben den dargestellten Anordnungen im Client-Server-Modell versteht man unter der Drei-Schicht-Architektur auch den in Abbildung 2.6a dargestellten Fall, in dem alle drei Schichten lose gekoppelt und gegebenenfalls räumlich verteilt sind. Bezüglich Abbildung 2.5 handelt es sich hierbei um eine Kombination aus (b) und (d), da hier eine lose Kopplung sowohl zwischen Präsentation und Verarbeitung als auch zwischen Verarbeitung und Datenhaltung besteht. Die Web-basierte Vier-Schicht-Architektur [Lan04, S. 22], dargestellt in Abbildung 2.6b, bezeichnet im Wesentlichen solch eine Drei-Schicht-Architektur, bei der die Präsentationsschicht in eine Client- und eine Webserver-Schicht aufgespalten ist. Die häufigste Umsetzung dieser Architektur ist eine Server-basierte Drei-Schicht-Anwendung, die über eine Web-Schnittstelle mit einem Browser zugänglich ist. Bezüglich Abbildung 2.5 handelt es sich um eine Kombination aus (a), (b) und (d).

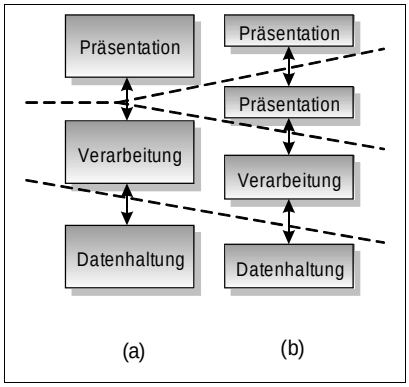

Abbildung 2.6: Drei- und Vier-Schicht-Architekturen mit loser Kopplung

Die Präsentationsschicht

Bei der Präsentationsschicht handelt es sich um die Ebene der Benutzungsoberfläche [TaSt03, S.67]. Hier wird es dem Benutzer ermöglicht, mit der Anwendung zu interagieren. Die Möglichkeiten der Realisierung von Benutzungsschnittstellen reichen von Zeichen-basierten Bildschirmen [Dun94, S. 107ff] über Browser-basierte Steuerung [Ash04, S. 199ff] bis hin zu komplexen grafischen Benutzungsoberflächen, die die Möglichkeiten moderner Fenstersysteme nutzen. Ein zeitgemäßer Ansatz für die Entwicklung von Benutzungsschnittstellen ist dabei die Model-View-Controller-Architektur [SJPC02; EKT02, S. 66ff], bei der Darstellung, Eingabevalidierung, Datenverarbeitung, Navigation und Sicherheitsaspekte [Ash04, S. 201ff] explizit unterschieden werden.

Die Verarbeitungsschicht

Die Verarbeitungsschicht enthält die Kernfunktionalität eines Anwendungssystems [TaSt03, S. 68ff]. Sie wird auch als Mittelschicht zwischen Präsentation und Datenhaltung bezeichnet. Entsprechend bietet die Verarbeitungsschicht von der Präsentationsschicht benötigte Dienste an, verarbeitet deren Aufrufe und nutzt dafür wo nötig Dienste der Datenhaltungsschicht.

Die Datenhaltungsschicht

Die Datenhaltungsschicht einer Drei-Schicht-Architektur enthält die Komponenten, die die eigentlichen Daten verwalten, mit denen das Anwendungssystem arbeitet [TaSt03, S. 69ff]. Häufig ist die Datenhaltungsschicht in Form von relationalen Datenbanksystemen realisiert, da diese zum Einen die Datenunabhängigkeit fördern, da der Zugriff auf die Daten über die generische Anfragesprache SQL [KBL05, S. 127ff] ermöglicht wird, und sie zum Anderen in aller Regel wenigstens ACID-Transaktionen unterstützen [KBL05, S. 763ff].

2.4.4 Die N-Schicht-Architektur

Die Erweiterung von drei auf eine beliebige Anzahl von Schichten führt von der Drei-Schicht-Architektur zur allgemeinen Mehrschichtarchitektur oder N-Schicht-Architektur. Im Gegensatz zur Drei-Schicht-Architektur existieren bei dieser Form des Anwendungsentwurfs prinzipiell keine strikten Richtlinien, welche Funktionalität in welcher Schicht gebündelt werden sollte. Vielmehr gelten hier die oben erläuterten allgemeinen Eigenschaften der Mehrschichtarchitektur.
Die verbreitetsten Vertreter der N-Schicht-Architektur sind die Datenbanksysteme. Sie sind normalerweise als Fünf-Schicht-Architektur konzeptioniert und umgesetzt. Die Einzelschichten sind hier als Abstraktionsebenen der Hardware anzusehen. Beim Entwurf von Datenbanksystemen wird aus Performanzgründen häufig auf Striktheit verzichtet, das heißt an wohldefinierten Stellen ist der direkte Zugriff über die Grenzen von mehr als einer Schicht aus möglich.

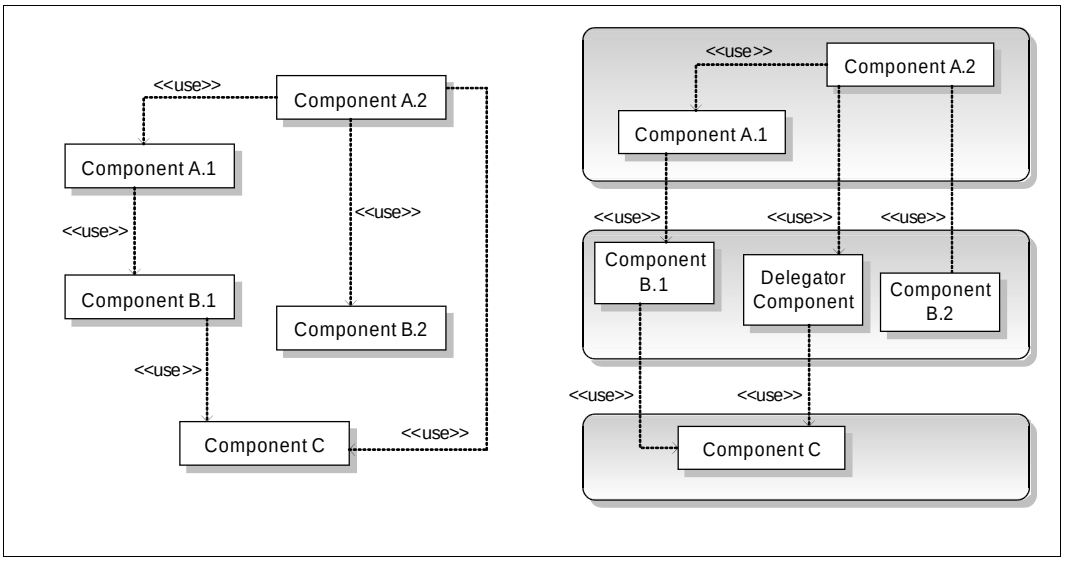

Abbildung 2.7: Von der Komponentenarchitektur zur Mehrschichtarchitektur

Während die N-Schicht-Architektur historisch gesehen aus den Drei-Schicht-Modellen hervorgegangen ist, kann sie formal als Spezialfall der Komponentenarchitektur betrachtet werden. In einem Anwendungssystem mit Komponentenarchitektur ist zusammengehörige Funktionalität in Komponenten gebündelt, die prinzipiell beliebig untereinander vermascht sein können. Der spezielle Fall, in dem Komponenten so zu Schichten gruppiert werden, dass nur noch Aufrufe zwischen direkt benachbarten Schichten erforderlich sind, führt zu einer strikten Mehrschichtarchitektur. Eine Anwendung mit Komponentenarchitektur kann in eine Mehrschichtarchitektur überführt werden, indem die genannte Schichteneinteilung durchgeführt wird. Sollte die Vermaschung zwischen den Komponenten so ausgeprägt sein, dass Aufrufe über mehr als eine Schichtgrenze hinweg auftreten, und ist darüber hinaus eine strikte Mehrschichtarchitektur gefordert, so müssen diese Aufrufe entsprechend über die Zwischenschichten delegiert werden. Abbildung 2.7 illustriert die Transformation von einer Komponentenarchitektur zu einer strikten Mehrschichtarchitektur. Dafür werden zusammengehörige Komponenten in Schichten zusammengefasst. Zusätzlich wird die ursprüngliche Nutzungsbeziehung zwischen den Komponenten A.2 und C über eine Zwischenkomponente realisiert, um eine strikte Schichteneinteilung zu erhalten.

2.5 Die Java 2 Platform Enterprise Edition

Neben Corba und .NET gehört die Java 2 Platform Enterprise Edition (J2EE) heute zu einem der wesentlichen Integrationsstandards für große Unternehmen [EKT02]. Zum Zeitpunkt der Fertigstellung der vorliegenden Arbeit (zweites Quartal 2006) war J2EE 1.4 die aktuelle Version. Der Nachfolger (Java EE 5) befand sich im späten Beta-Stadium. Soweit nicht anders gekennzeichnet, beziehen sich in dieser Arbeit alle Aussagen über J2EE auf die Version 1.4. Die Java 2 Platform Enterprise Edition ist ein Standard für die Entwicklung von Anwendungsservern und Server-seitigen Anwendungen, die typischerweise in großen Unternehmen eingesetzt werden. Entwicklern von Servern und Anwendungen wird mit J2EE ein gemeinsamer Spezifikationssatz zur Verfügung gestellt, auf den sie sich bei dem Entwurf von Server-Komponenten und Anwendungsmodulen berufen können. Die Spezifikation von J2EE 1.4 [Sha03] fordert, dass ein Server, der den vollständigen J2EE-Stack anbietet, neben diversen technischen Funktionalitäten wie dem Verzeichnisdienst *JNDI* (Java Naming and Directory Interface) [SDN06] und der

Unterstützung der Konfiguration und Installation von Anwendungen auf einem Server (*Deployment*) mindestens einen Enterprise-Java-Beans-Container (EJB-Container) [DeMi03], einen Servlet-Container [CoYo03] und einen J2EE-Connector-Architecture-Container (JCA-Container) [Sun03] zur Verfügung stellt. Hierbei ergibt sich die Forderung nach dem JCA-Container implizit, durch Sicherheitseinschränkungen der anderen Container. Jeder J2EE-Container ist eine logische Komponente eines J2EE-Servers, die eine Laufzeitumgebung für einen J2EE-Modultyp implementiert. Abbildung 2.8 verschafft einen strukturellen Überblick über die Komponenten von J2EE.

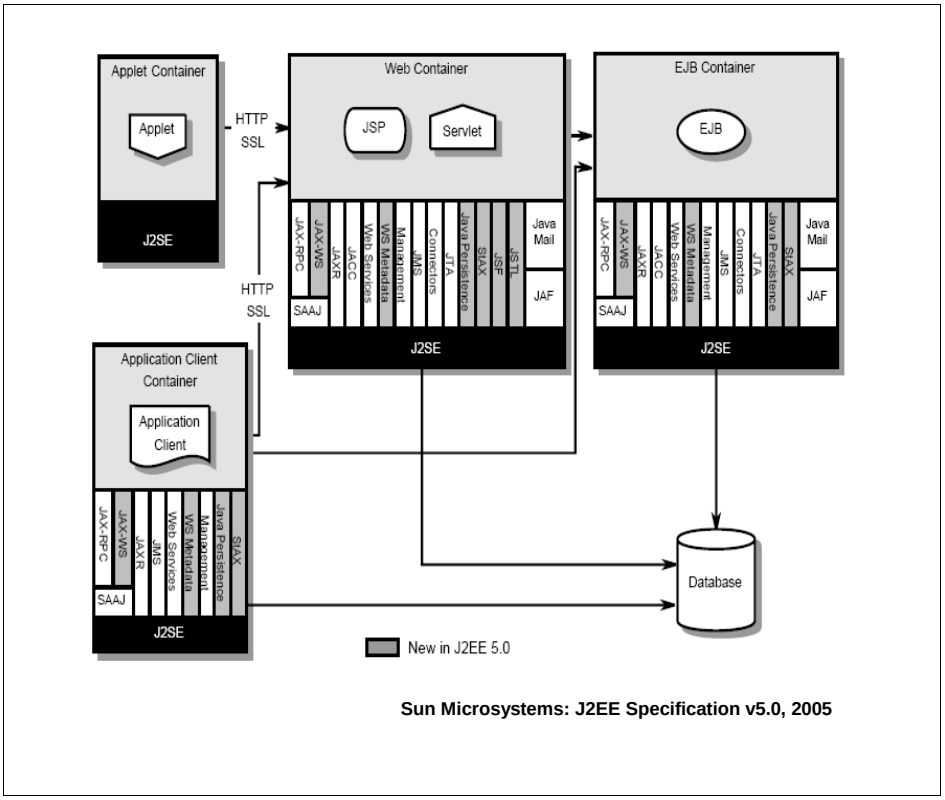

Abbildung 2.8: Die Komponenten von J2EE

Neben den Anforderungen, denen ein J2EE-Server genügen muss, existiert auch ein Spezifikationssatz für die Entwicklung von J2EE-Anwendungen. Es ist zwar in der Praxis durchaus üblich, eine Anwendung, die eine oder mehrere der Technologien von J2EE verwendet, als J2EE-Anwendung zu bezeichnen, in der vorliegenden Arbeit meint dieser Begriff aber nur eine J2EE-Anwendung im engeren Sinne, das bedeutet eine J2EE-Server-basierte Anwendung. Solch eine J2EE-Anwendung wird normalerweise in Form eines J2EE-Anwendungsarchivs erstellt, welches alle notwendigen Module als Modularchive und Installationsinformationen in Form von *Deployment-Deskriptoren* beinhaltet. Der typische Aufbau einer J2EE-Anwendung mit Web-Schnittstelle wird in Abbildung 2.9 dargestellt. Von der Abbildung abweichend wäre der Zugriff der verarbeitenden Komponenten auf die Datenquellen prinzipiell auch über ein oder mehrere JCA-Module realisierbar. Dies hätte gegenüber dem mehr oder weniger direkten Datenbankzugriff den Vorteil, dass einerseits eine striktere Trennung von Verarbeitung und Datenzugriff durchgesetzt und sich andererseits der Aufwand zur Integration unterschiedlicher Datenquellen und Applikationsserver insgesamt verringern würde (theoretisch würde eine N-mal-M-Problematik auf eine N-plus-M-Problematik reduziert werden). Da der zusätzliche Aufwand für die Entwicklung eigener JCA-Module aufgrund fehlender Verbreitung allerdings vergleichsweise hoch ist, hat diese Art der Umsetzung nur eine geringe praktische Relevanz. Aus diesem Grund

wird in der vorliegenden Arbeit auf die Verwendung der J2EE Connector Architecture nicht näher eingegangen. Die Technologien, die bei der Umsetzung der einzelnen Module einer typischen J2EE-Drei-Schicht-Anwendung mit Web-Schnittstelle zum Einsatz kommen, werden im Folgenden erläutert.

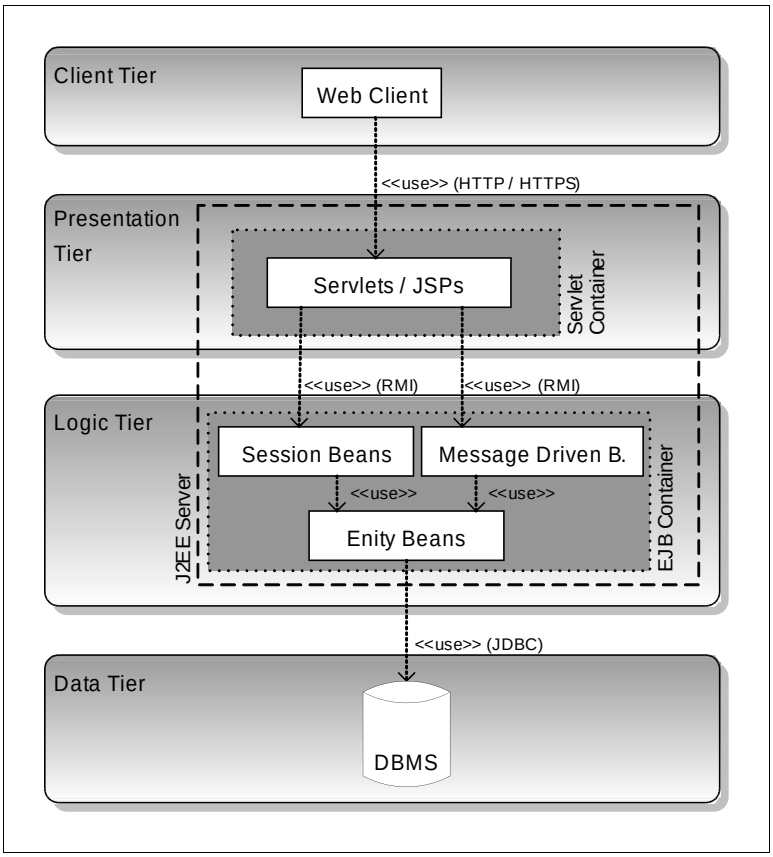

Abbildung 2.9: Aufbau einer typischen Web-basierten J2EE-Anwendung

2.5.1 Umsetzung der Präsentationsschicht mit Servlet-Technologie

Für die Umsetzung Web-basierter Benutzungsschnittstellen ist in J2EE das Servlet-Konzept vorgesehen. Zur Version 1.4 des J2EE-Standards gehört die Servlet-Spezifikation in der Version 2.4. Das Servlet-Konzept stellt die Technologien Servlet und JSP (JavaServerPages) für die Entwicklung dynamischer Webseiten zu Verfügung. Servlets sind Klassen, die in der Lage sind HTML-Ausgaben zu produzieren. JSPs sind HTML-Dokumente, die Java-Code enthalten können. Alle Servlet-Klassen sowie der Einstiegspunkt in die Web-Schnittstelle, das heißt die Seite, die zuerst angezeigt werden soll, sind in dem Deployment-Deskriptor des Web-Moduls festgelegt.

2.5.2 Umsetzung der Verarbeitungsschicht als EJB-Modul

Die Enterprise-Java-Beans-Technologie bildet das Komponentenmodell von J2EE. Die Version 2.1 der EJB-Spezifikation gehört zum J2EE-Standard in der Version 1.4. Trotz der namentlichen Ähnlichkeit von Enterprise Java Beans und *Java Beans*, die das Komponentenmodell von Java bilden [Ham97], handelt es sich um zwei sehr unterschiedliche Konzepte. Während der Java-Beans-Ansatz im Wesentlichen aus einem Regelsatz besteht, der beschreibt, wie die Schnittstelle eines

Java Beans aufgebaut sein muss, um einen automatisierten Zugriff auf seine Eigenschaften und Operationen zu ermöglichen, modellieren Enterprise Java Beans die Funktionalität und Datenstrukturen einer J2EE-Anwendung. Die EJB-Spezifikation sieht drei Klassen von Enterprise Java Beans vor. Dies sind Entity Beans, Session Beans und Message Driven Beans. Enterprise Java Beans können von lokalen und entfernten Clients genutzt werden. Die lokale oder entfernte Zugreifbarkeit von Entity Beans und Session Beans wird durch das *Home Interface* festgelegt. Hier wird spezifiziert, ob und auf welche Weise EJB-Objekte zugegriffen werden können. Dieser Mechanismus ermöglicht der EJB-Laufzeitumgebung die Vorhaltung von EJB-Pools sowie die Wiederverwendbarkeit einzelner EJB-Objekte. Die Methoden, die ein Enterprise Java Bean zur Verfügung stellt, sind in seinem *Component Interface* spezifiziert. Je nachdem, ob das Home Interface den lokalen oder den entfernten Zugriff auf Objekte einer EJB-Klasse vorsieht, müssen ein *Local Interface* oder ein *Remote Interface* zur Verfügung gestellt werden. [DeMi03, S. 45ff]

Entity Beans

Entity Beans sind die Klassen eines EJB-Moduls, die die persistenten Daten repräsentieren, mit denen die J2EE-Anwendung arbeitet, sie bilden das Datenkomponentenmodell von J2EE. In den meisten Umsetzungen entspricht jeder Entity-Bean-Typ einer (logischen) Tabelle der Datenbasis der Anwendung und jedes Entity-Bean-Exemplar einer Zeile der entsprechenden Tabelle. Prinzipiell lässt der J2EE-Standard einen Zugriff auf Entity-Bean-Exemplare von beliebigen Modulen aus zu, im Sinne einer sauberen Trennung zwischen Datenhaltung und Präsentation sollte aber im Allgemeinen auf solche Zugriffe verzichtet werden, vielmehr werden alle Manipulationen und Auskünfte bezüglich Entity Beans über entsprechende Methoden der Session Beans durchgeführt. Der aktuelle J2EE-Standard sieht zwei unterschiedliche Typen von Entity Beans vor. Dies sind *Container Managed Persistence* (CMP) und *Bean Managed Persistence* (BMP) Entity Beans. Im Fall von Container Managed Persistence werden alle Änderungs- und Aktualisierungsoperationen der Entity Beans ohne Zutun des Entwicklers vom EJB-Container durchgeführt. Der Entwickler der Entity Beans kümmert sich also im Wesentlichen um den Aufbau der Klassen und die Bereitstellung einer entsprechenden Datenbankstruktur. Wird dagegen Bean Managed Persistence genutzt, müssen die notwendigen Datenbankoperationen vom Entwickler in die Entity Beans eingefügt werden.

Session Beans

Neben Message Driven Beans bilden Session Beans das Funktionskomponentenmodell von J2EE. Sie kapseln die Funktionalität eines Anwendungssystems und stellen diese anderen Komponenten, Anwendungsschichten oder Client-Anwendungen lokal oder per RMI beziehungsweise RMI/IIOP über entfernte Methodenaufrufe zur Verfügung. Die besondere Eigenschaft von Session Beans gegenüber Message Driven Beans ist ihre synchrone Aufrufsemantik, das bedeutet, dass der Kontrollfluss nach einer Leistungsanforderung an ein Session Bean erst dann zu der anfordernden Instanz zurückkehrt, wenn die Leistung erbracht wurde. Session Beans können zustandsbehaftet oder zustandslos sein.

Message Driven Beans

Wie Session Beans sind Message Driven Beans Bestandteil des Funktionskomponentenmodells von J2EE. Message Driven Beans nutzen allerdings eine asynchrone Aufrufsemantik. Das bedeutet,

dass der Aufruf einer Methode eines Message Driven Beans über das Senden einer Nachricht erfolgt. Nach Bearbeitung des Auftrags sendet das Message Driven Bean eine Antwortnachricht zurück. Während der Bearbeitung im Message Driven Bean verbleibt der Kontrollfluss im aufrufenden Prozess beziehungsweise Thread.

2.5.3 Zugriff auf die Datenhaltungsschicht mit JDBC

Es wurde bereits diskutiert, dass der J2EE-Standard für den Zugriff auf die Datenhaltungsschicht die J2EE Connector Architecture vorsieht, dass diese aber in der Praxis nur selten zum Einsatz kommt. Tatsächlich findet der Zugriff auf die Datenbasis durch eine J2EE-Anwendung normalerweise unter Nutzung der *JDBC-API* (Java Data Base Connectivity) statt [And05, Sun06]. Die JDBC-API befindet sich zum Zeitpunkt der Fertigstellung dieser Arbeit in der Version 3, Version 4 ist im späten Beta-Stadium. Sie ist Bestandteil der Java 2 Platform Standard Edition. Die JDBC-API stellt Anwendungsentwicklern einen standardisierten Satz von Klassen und Operationen zur Verfügung, die prinzipiell den Zugriff auf beliebige SQL-Datenquellen erlauben. Die Anbindung der einzelnen Datenbanksysteme wird über JDBC-Treiber realisiert, die die Kommunikationsprotokolle der jeweiligen Datenbanksysteme beherrschen. Im Rahmen eines JDBC-Treibers werden zwei wesentliche Klassen zur Verfügung gestellt, die Data-Source-Klasse und die Connection-Klasse. Es handelt sich hierbei um Implementierungen der JDBC-Schnittstellen `javax.sql.DataSource` und `java.sql.Connection`. Die Connection-Klasse stellt hierbei die Verbindung zur Datenbasis dar. Sie stellt Funktionen zur Verfügung, die es ermöglichen, SQL-Anfragen an die Datenbasis zu senden und entsprechende Antworten beziehungsweise Ergebnismengen zu empfangen. Die Aufgabe der Data-Source-Klasse besteht darin, Connection-Objekte zu einer Datenbasis zur Verfügung zu stellen. Hierbei können verschiedene Ansätze verwendet werden. Das genutzte Data-Source-Objekt kann bei jeder Anforderung eines Connection-Objekts ein neues Exemplar der Connection-Klasse erzeugen, es kann aber auch ein Pool von Connection-Objekten vorgehalten werden, aus dem Anforderungen bedient werden. Der Ansatz der Pooled Connections wird im J2EE-Umfeld relativ häufig genutzt, da die meisten J2EE-Server Mechanismen unterstützen, Connection Pools anzulegen, die dann als JNDI-Ressourcen zur Verfügung gestellt werden.

2.6 Die .NET-Plattform

Microsoft verwendet das Label .NET für eine Reihe von Produkten, die einer gemeinsamen Strategie folgen. Laut [Gre02, S. 39] versteht man unter .NET eine Plattform, auf der Code ausgeführt werden kann und die über eine Klassenbibliothek verfügt, die bezüglich ihres Codes von allen unterstützen Sprachen verwendet werden kann. Außerdem werden auch Entwicklungswerkzeuge wie das Visual Studio und viele Serverprodukte zu .NET gezählt. Abbildung 2.10 [Gre02, S. 41] verschafft einen groben Überblick über das Rahmenwerk, die Sprachen und die Werkzeuge der .NET Entwicklungs- und Laufzeitumgebung. Ein großer Vorteil der .NET-Plattform ist die Möglichkeit, beliebige Progammiersprachen zu verwenden, solange diese der Common Language Specification genügen. Dies wird durch die Übersetzung in eine Zwischensprache ermöglicht, die *Intermediate Language* (IL).

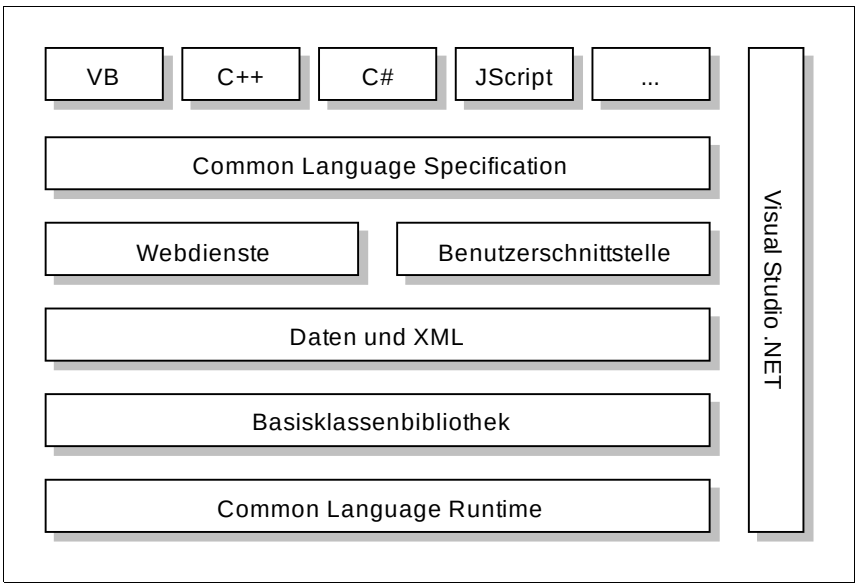

Abbildung 2.10: .NET-Rahmenwerk, Sprachen und Werkzeuge

Das .NET-Rahmenwerk ist für diese Arbeit grundsätzlich relevant, da sich seine Komponenten als Middleware für die Entwicklung Web-basierter Anwendungen mit Drei-Schicht-Architektur nutzen lassen. Die Technologien, die bei der Umsetzung der einzelnen Schichten zum Einsatz kommen, werden im Folgenden kurz erläutert.

2.6.1 Umsetzung der Präsentationsschicht mit ASP.NET

ASP.NET (Active Server Pages .NET) ermöglicht mit der Technologie ASP.NET *WebForms* die Entwicklung dynamischer Webseiten [Lan04, S. 160ff]. Bei ASP.NET WebForms handelt es sich um Dokumente im ASPX-Dateiformat. Im Wesentlichen sind dies HTML-Dokumente, die die Benutzung zusätzlicher Tags ermöglichen. Darüber hinaus ist es möglich, ASPX-Dokumente um ausführbaren Programmtext anzureichern (so genannter Inline Code), der beim Aufruf der Seite Server-seitig ausgeführt wird. Hierfür kommen alle Sprachen der .NET-Plattform in Frage, da diese jeweils in die gemeinsame Intermediate Language übersetzt werden. Im Wesentlichen entspricht das Konzept der ASP.NET WebForms dem JSP-Ansatz der J2EE-Plattform. Ein entsprechender Servlet-Ansatz fehlt der .NET-Welt jedoch. Auch der Aufbau von ASP.NET WebForms mithilfe von Controls-Objekten erinnert an ein Konzept der J2EE-Plattform, die JSF-Technologie (Java Server Faces).

2.6.2 Umsetzung der Verarbeitungsschicht mit COM+

COM+ ist das .NET-Komponentenmodell. COM+ wurde aus den Technologien *COM* (Component Object Model), DCOM (Distributed COM) und MTS (Microsoft Transaction Server) zusammengefasst. COM+ unterstützt lokale und entfernte Methodenaufrufe, Transaktionen, Rollen-basierte Sicherheit, Ressourcen-Pooling, Object-Pooling, Komponenten mit Warteschlangen und Ereignisse [Gre02]. Insgesamt ähnelt das heutige COM+ dem Enterprise-Java-Beans-Konzept der J2EE-Plattform. Zusätzlich unterstützt COM+ die Ereignisverwaltung und ermöglicht Komponenten auf diese Weise an bestimmten Kontrollflüssen teilzuhaben. Allerdings findet sich in COM+ kein Ansatz, der mit den Entity Beans der J2EE-Plattform vergleichbar wäre. Es handelt sich also bei COM+ um ein Funktionskomponentenmodell, ähnlich den Session und Message Driven Beans bei J2EE.

2.6.3 Zugriff auf die Datenhaltungsschicht mit ADO.NET

ADO.NET stellt .NET-Anwendungen Methoden für den Datenzugriff zur Verfügung. Bei ADO.NET handelt es sich um Microsofts .NET-Erweiterung der *ADO*-Technologie (ActiveX Data Objects). ADO.NET unterscheidet zwischen Inhaltskomponenten und Managed-Provider-Komponenten. Inhalte werden in Objekte der Klasse `DataSet` und deren Hilfsklassen `DataTable`, `DataRow`, `DataColumn` und `DataRelation` abgebildet. Die Aufgabe der Managed-Provider-Komponenten besteht darin, Inhalte zwischen der Datenbasis und den Inhaltskomponenten auszutauschen. Die wesentliche Komponente hierfür ist die Klasse `DataAdapter`, die für die einzelnen Datenbasen spezialisiert werden muss. Für den Datenaustausch zwischen Datenbasis und Inhaltskomponenten wird ein Microsoft-spezifisches Verfahren basierend auf dem Checkout/Checkin-Ansatz verwendet. Konkret bedeutet dies, dass eine Datenmenge mittels der Load-Methode der Data-Adapter-Klasse in ein Data-Set-Objekt geladen wird. Auf diesem Data-Set-Objekt können lokal Änderungen vorgenommen werden. Der Aufruf der Update-Methode am Data-Adapter-Objekt führt zu der Prüfung, ob zwischenzeitlich in der Datenbasis Änderungen vorgenommen wurden, die inkonsistent mit den Änderungen an dem lokalen Auszug sind. Ist dies der Fall, wird der Update-Vorgang mit einer Ausnahme abgebrochen. Andernfalls werden die Änderungen in der Datenbasis manifestiert.

2.7 Gegenüberstellung von J2EE und .NET

Wie Tabelle 2.1 [Lan04, S. 138] entnommen werden kann, unterscheiden sich J2EE und .NET relativ stark in der Anzahl der verfügbaren Programmiersprachen, der Plattform- und Herstellerunabhängigkeit sowie der Performanz. Die Konzepte, die in J2EE und .NET verwendet werden, um Server-basierte Mehrschichtanwendungen zu entwickeln, sind jedoch weitgehend ähnlich. Aufgrund dieser konzeptionellen Ähnlichkeiten sind die Ansätze zur Leistungsbewertung, die in Kapitel 3 vorgestellt werden, prinzipiell auf beide Architekturformen anwendbar. Da sich die J2EE-Plattform aufgrund ihrer Offenheit und Herstellerunabhängigkeit in den letzten Jahren als Industriestandard durchgesetzt hat und wegen des freien Zugangs zu Java und J2EE-Server-Implementierungen sowie entsprechenden Werkzeugen und Entwicklungsumgebungen wurden die weiterführenden Konzepte dieser Arbeit mit Fokus auf die Java 2 Platform Enterprise Edition entwickelt.

Tabelle 2.1: Allgemeiner Vergleich von J2EE und .NET

	Programmier-sprachen	**Plattform-unab-hängigkeit**	**Hersteller-unab-hängigkeit**	**Performanz**
J2EE	Java	gut	gut	schlecht
.NET	C#, VB.NET, C++, Cobol .NET, etc. (insgesamt ca. 22)	schlecht	mäßig	gut

2.8 Leistungsbewertung in Java und J2EE

Lasttests und Leistungsbewertungsprozesse werden in Java und J2EE durch eine Vielzahl von Mechanismen und Werkzeugen unterstützt. In den folgenden Abschnitten werden exemplarisch zwei Rahmenwerke vorgestellt, deren Ansätze für die Konzeption und Implementierung des Leistungsbewertungssystems in dieser Arbeit Verwendung finden.

2.8.1 The Grinder

The Grinder ist die Implementierung eines Leitstellenkonzepts, das in dem Buch „J2EE Performance Testing“ [ZAO02] vorgestellt wird. Es wurde für den BEA WebLogic Server [BEA06] umgesetzt. Unter anderem werden Werkzeuge zur Verfügung gestellt, die das Aufzeichnen von HTTP-Anfragen ermöglichen und es zulassen, daraus Anfragen an eine Web-Anwendung zu generieren. Auf diese Weise ist es mit The Grinder möglich, Web-Anwendungen Lasttests zu unterziehen und Leistungskennzahlen wie Antwortzeiten und Durchsätze zu ermitteln.

2.8.2 JMeter

JMeter ist ein Leistungsmessungswerkzeug, das als Jakarta-Projekt im Rahmen der Apache Software Foundation entwickelt wird [Apa05]. JMeter wurde vollständig in Java entwickelt; es handelt sich um ein Open-Source-Projekt. Mit JMeter ist es möglich, unterschiedlichste Anwendungsfälle zu testen. Hierfür müssen so genannte *Sampler* für die zu testenden Objekte entwickelt werden. Zur Zeit existieren Sampler für HTTP-Ressourcen, FTP-Ressourcen, JDBC-Ressourcen, SOAP/XML-RPC-Ressourcen und Java-Ressourcen. Bei der Auswertung der Ergebnisse kann JMeter nicht nur die ermittelten Antwortzeiten sondern auch die entsprechenden Rückgabewerte in textueller Hinsicht bewerten. JMeter nimmt keine internen Messungen vor, das heißt es betrachtet alle zu vermessenden Objekte und Domänen nur von außen und macht keine Aussagen über innere Strukturen oder die Speichernutzung. Wie The Grinder verfügt auch JMeter für das Messen der Leistung von Web-basierten Anwendungen über einen Mechanismus zum Aufzeichnen von HTTP-Anfragen.

2.8.3 Grenzen der Werkzeuge zur Leistungsbewertung

The Grinder und JMeter bieten Möglichkeiten, Server-basierte Anwendungen bezüglich ihrer Leistungsfähigkeit zu untersuchen, ohne Eingriffe in diese Anwendungen vornehmen zu müssen. Allerdings werden in beiden Fällen keinerlei Analysen bezüglich des strukturellen Aufbaus von Anwendungen vorgenommen. Dies ist typisch für Leistungsbewertungswerkzeuge, die auf Quelltextnutzung verzichten. Es existieren Werkzeuge, die in der Lage sind, die Struktur und teilweise auch den Kontrollfluss von Anwendungen zu ermitteln, indem der Quelltext einer eingehenden Analyse unterzogen wird. In diesem Zusammenhang sei zum Beispiel der *JProfiler* von EJ-Technologies [EJT06] genannt. Solche Ansätze zielen aber eher auf die Unterstützung während der Anwendungsimplementierung ab als auf die Bewertung fertiger Anwendungssysteme.

2.9 Analyse von Java-Anwendungen

Um die Leistung eines Anwendungssystems zu bewerten, ist es nötig, dieses in seinen Komponenten sowie seiner Gesamtheit eingehend zu analysieren, um darauf aufbauend Testmuster zu generieren. Ziel dieser Arbeit ist die Quelltext-unabhängige Leistungsbewertung von Anwendungssystemen. Aus diesem Grund werden in den nachfolgenden Abschnitten bestehende Konzepte zur Analyse von Java-Bytecode vorgestellt. Außerdem wird auf Methoden zur Ermittlung des Funktionsumfangs eingegangen. Auf die Darstellung Quelltext-abhängiger Analyse- und Profilingkonzepte und -werkzeuge wird verzichtet, da sie den Fokus dieser Arbeit verlassen würde.

2.9.1 Analyse von Java-Klassen mit Hilfe der Java-Reflection-API

Die *Java-Reflection-API* [SDN05] ist Teil der Java-System-Bibliothek. Sie befindet sich im Paket `java.lang.reflect`. Die Java-Reflection-API umfasst Klassen und Schnittstellen, die den Zugang zu strukturellen Informationen von Klassen und Objekten zur Laufzeit ermöglichen. Sie erlaubt darüber hinaus den programmatischen Zugriff auf Informationen über Felder, Methoden und Konstruktoren geladener Klassen, sowie den generischen Zugang zu diesen Feldern, Methoden und Konstruktoren innerhalb der geltenden Sicherheitsbeschränkungen. Das bedeutet, dass Konstruktoren und Methoden von Klassen und Objekten aufgerufen und Felder zugegriffen werden kann, deren Struktur zum Übersetzungszeitpunkt nicht fest stehen oder nicht bekannt sind. Zusätzlich erlaubt die Klasse `AccesibleObject` den Zugriff auf Konstruktoren, Methoden und Felder die nicht als `public` deklariert wurden, falls die ausführende virtuelle Maschine über die entsprechenden Rechte (`ReflectPermission`) verfügt. Die Klasse `Arrays` bietet Methoden an, die es erlauben, Arrays dynamisch zu erzeugen und zuzugreifen. Gemeinsam mit der Klasse java.lang.Class bietet die Java-Reflection-API eine Fülle an Möglichkeiten, die Struktur von Klassen und Objekten sowie deren Aufbau zur Laufzeit zu ermitteln.

2.9.2 Analyse von Java-Klassen mittels BCEL

Die Entwicklung der *Byte Code Engineering Library* (BCEL) ist ein Unterprojekt des Apache-Jakarta-Projekts. Das Ziel von BCEL besteht darin, dem Benutzer eine komfortable Möglichkeit zur Analyse und Manipulation von Java-Klassen anzubieten. Die BCEL-API besteht aus einer Reihe von Werkzeugen, zur statischen Analyse und dynamischen Erzeugung oder Transformation von Java-Class-Dateien [Apa03]. Damit ist BCEL prinzipiell in der Lage, vorhandene Java-Klassen um zusätzliche Funktionalität zu erweitern. Im Laufe dieser Arbeit hat sich allerdings herausgestellt, dass die Erzeugung und Übersetzung von Quelltext besser handhabbar ist als als die Manipulation fertiger Klassen. Daher wird BCEL in den hier vorgestellten Konzepten und der prototypischen Implementierung nicht weiter betrachtet.

2.10 Fazit

Im vorangegangenen Kapitel wurden allgemeine Konzepte und formale Grundlagen der Leistungsbewertung von Anwendungssystemen untersucht. Zusätzlich wurde die Klasse der Server-basierten Anwendungen mit Mehrschichtarchitektur und die beiden wichtigsten Plattformen für deren Entwicklung J2EE und .NET vorgestellt. Es wurde hierbei gezeigt, dass die Konzepte von J2EE und .NET sehr ähnlich sind. Abschließend wurden die aktuellen Möglichkeiten

der Leistungsbewertung im Umfeld von Java und J2EE ermittelt. Die formalen Grundlagen der Leistungsbewertung, die im vorangegangenen Kapitel vorgestellt wurden, bilden die Basis für die Entwicklung von Konzepten eines Leistungsbewertungssystems für Server-basierte Anwendungen mit Mehrschichtarchitektur. J2EE und .NET wurden als wichtige Vertreter für Plattformen zur Entwicklung solcher Anwendungen vorgestellt. Die im nächsten Kapitel vorgetragenen Konzepte der Leistungsbewertung wurden speziell für die Nutzung in J2EE-Umgebungen entwickelt. Aufgrund der starken konzeptionellen Ähnlichkeit von J2EE und .NET sollte es aber durchaus möglich sein, die Konzepte für .NET anzupassen und umzusetzen.

3 Entwurf des Leistungsbewertungssystems

Dieses Kapitel geht auf die Konzepte ein, die im Rahmen dieser Arbeit für den Entwurf eines Leistungsbewertungssystems für J2EE-Anwendungen mit Mehrschichtarchitektur entwickelt wurden. Hierbei werden zuerst die Anforderungen an ein derartiges Leistungsbewertungssystem behandelt. Darauf folgend werden zunächst die Möglichkeiten der Analyse von J2EE-Anwendungen ergründet und darauf aufbauend ein Konzept für die gekapselte Betrachtung einzelner Anwendungsschichten mit Hilfe von Driver-, Stub- und Filter-Komponenten, in dieser Arbeit als *Prüfkomponenten* bezeichnet, vorgestellt. Außerdem wird der Bezug zur dreischichtigen J2EE-Anwendungsarchitektur hergestellt. Im weiteren Verlauf werden Konzepte zur Analyse der Anwendungsschichten einer J2EE-Anwendung bezüglich des Funktionsumfangs sowie der Generierung realistischer Anfragen diskutiert. Darauf aufbauend werden Konzepte zur Erstellung von Testplänen, die unterschiedlichen Möglichkeiten der Durchführung einer Leistungsbewertung sowie die Auswertung von Testergebnissen behandelt. Abschließend wird der Bezug der entwickelten Konzepte zu der in Kapitel 2 vorgestellten vierstufigen Methodik der Leistungsbewertung hergestellt.

3.1 Anforderungen

Die Anforderungen an die Konzeption eines Leistungsbewertungssystems von J2EE-Anwendungen legen Ansprüche und Einschränkungen der entwickelten Konzepte fest. Sie ergeben sich zum Teil aus der Zielsetzung der generischen Leistungsbewertung, die die Forderung nach Automatisierung birgt sowie aus der angestrebten Möglichkeit, einzelne Schichten eines Anwendungssystems isoliert zu betrachten. Um eine möglichst breite Anwendbarkeit der Konzepte zu erreichen war es außerdem notwendig, diese in Hinblick auf weitgehende Unabhängigkeit von fertigen Implementierungen (beispielsweise unterschiedlicher J2EE-Server) zu entwickeln und nur die Vorgaben des eigentlichen J2EE-Standards zu verwenden.

3.1.1 Teilautomatisierte Erstellung parametrisierbarer Testpläne

Ein wichtiger und arbeitsintensiver Vorgang bei der Leistungsbewertung ist die Erstellung von Testplänen. Diese Testpläne dienen später als Grundlage für die Durchführung der Leistungsbewertung. Es wurde bereits angesprochen, dass eine vollautomatische Generierung solcher Testpläne normalerweise nicht möglich ist, da ohne intensive Analyse des Quelltextes im Allgemeinen keine Möglichkeit besteht, eine vollständige funktionale Abdeckung des zu bewertenden Anwendungssystems durch automatisch generierte Testpläne zu erreichen. Da aber gerade die Erstellung von Testplänen normalerweise sehr aufwändig ist und tief gehende Kenntnisse über das betreffende Anwendungssystem erfordert, andererseits aber im Rahmen dieser Arbeit auf die Nutzung von Anwendungsquelltext verzichtet wird, werden in den folgenden Abschnitten Konzepte vorgestellt, die wenigstens eine Teilautomatisierung der Testplanerstellung erlauben. Dabei werden im Wesentlichen alle tatsächlich relevanten Arbeitsabläufe innerhalb eines Anwendungssystems während der Benutzung aufgezeichnet, ohne dass dafür der Quelltext des Anwendungssystems untersucht werden muss. Die zusätzliche Parametrisierbarkeit dieser Aufzeichnungen resultiert in einem mächtigen Instrument zur Bewertung von Arbeitsabläufen in einer Anwendung bei unterschiedlichen Lastverhältnissen.

3.1.2 Unabhängigkeit von J2EE-Server-spezifischen Eigenschaften

Es wird in den folgenden Abschnitten gezeigt, dass Konzepte zur Leistungsbewertung von J2EE-Anwendungen an vielen Stellen die Analyse von J2EE-Anwendungen erfordern. Um eine J2EE-Anwendung auf einem spezifischen J2EE-Server zu installieren, ist es notwendig, diese Anwendung den Anforderungen des J2EE-Servers entsprechend zu konfigurieren. Der Vorgang der Konfiguration und Installation bis hin zur Einsatzbereitschaft wird als *Deployment* bezeichnet. Bestimmte Aspekte dieser Anwendungskonfiguration sind nicht im J2EE-Standard festgelegt (wie beispielsweise die Festlegung und Zuweisung von JNDI-Namen) und werden entsprechend Server-spezifisch gehandhabt. Auf die Nutzung solcher Server-spezifischen Konfigurationsaspekte wird bei der Entwicklung von Konzepten für die Analyse von J2EE-Anwendungen verzichtet, damit diese Konzepte für alle J2EE-Server umsetzbar und nutzbar sind.

3.1.3 Berücksichtigung dynamischer Aspekte

Die Konzepte, die in dieser Arbeit entstanden sind, wurden mit Hinblick auf möglichst universelle Einsetzbarkeit entwickelt. Sie dienen in erster Linie der Leistungsbewertung von Server-basierten Anwendungen mit Web-Schnittstelle. Daher waren beim Entwurf der Konzepte zur Leistungsbewertung insbesondere dynamische Aspekte von Web-Schnittstellen zu berücksichtigen. Web-Schnittstellen ermöglichen die Nutzung einer Server-basierten Anwendung mittels HTTP. Da HTTP einerseits als zustandsloses Protokoll entwickelt wurde, die Web-Schnittstellen Server-basierter Anwendungen andererseits aber in aller Regel auf Kontext-bezogene Zustandsinformationen angewiesen sind, verfügt die J2EE-Plattform Mechanismen zur Sitzungsverwaltung mit HTTP. Diese Mechanismen basieren auf der Übertragung von Sitzungsidentifikationsnummern zwischen Server und Client mittels Cookies im HTTP Header oder Parametern, die in die Anfrage-URL eincodiert werden. Auf diese Weise kann ein Server Anfragen eines Clients einer Sitzung und damit einem Anwendungskontext zuordnen. Da sich die Sitzungsidentifikationsnummer bei jeder neuen Sitzung ändert, müssen solche dynamischen Aspekte bei der Ausführung vorher aufgezeichneter Testpläne beachtet werden.

3.1.4 Betrachtung anteiliger Leistungserbringung

Diese Arbeit grenzt sich neben dem Verzicht auf Quelltextnutzung von anderen Arbeiten und Konzepten dadurch ab, dass die hier entwickelten Konzepte die Betrachtung anteiliger Leistungserbringung in Anwendungssystemen mit Mehrschichtarchitektur ermöglichen. Dies bezieht sich insbesondere auf die Antwortzeit einer Leistung, für die häufig interessant ist, in welchen Schichten des Anwendungssystems sie zu welchen Anteilen erbracht wird, dies drückt sich in den anteiligen Antwortzeiten einer Leistung bezüglich der einzelnen Anwendungsschichten gemäß der Definition in Abschnitt 2.3.2 aus. Da die Mechanismen zur Betrachtung anteiliger Leistungserbringung eine Anreicherung des zu bewertenden Anwendungssystems um gewisse Funktionalität erfordern, müssen an dieser Stelle unter Umständen zusätzliche Maßnahmen getroffen werden, damit nachträglich eingefügte Zusatzfunktionalität keinen, wenigstens aber nur einen möglichst geringen Einfluss auf die Leistung des Anwendungssystems hat.

3.1.5 Automatisierte Testdurchführung

Leistungsbewertung dient dem Zweck, Kennzahlen eines Anwendungssystems zu ermitteln, anhand derer es möglich ist, Aussagen über die Leistungsfähigkeit dieses Systems zu treffen. Insbesondere für die Ermittlung von Leistungskennzahlen, die vom Parallelitätsgrad abhängen, die also eine Funktion in Abhängigkeit von der Anzahl gleichzeitiger Leistungsanforderungen definieren, ist eine große Zahl von Wiederholungen einzelner Abläufe notwendig. Dies führt dazu, dass das eigentliche Ermitteln der gewünschten Leistungskennzahlen sehr zeitaufwändig sein kann. Daher wurde für die entwickelten Konzepte gefordert, eine automatisierte Durchführung vorab konfigurierter Leistungstests zu ermöglichen.

3.1.6 Unterstützung bei der Auswertung der Bewertungsergebnisse

Nach der Ermittlung der Kennzahlen gestaltet sich die Interpretation der ermittelten Werte häufig schwierig, da einzelne Kennzahlen für sich genommen wenig Aussagekraft besitzen. Neben der reinen Leistungsmessung, die als Hauptanforderung der vorliegenden Arbeit anzusehen ist, sind also auch Konzepte erforderlich, die bei der Interpretation von Bewertungsergebnissen unterstützend eingesetzt werden können. Dies kann beispielsweise durch eine Visualisierung von Kennzahlen in Form von Verläufen in Abhängigkeit vom Parallelitätsgrad oder durch die Darstellung anteiliger Antwortzeiten in Form von Interaktionsdiagrammen geschehen.

3.2 Analyse von J2EE-Anwendungen

Die Analyse einzelner Anwendungsschichten von J2EE-Anwendungen erfordert eine genaue Kenntnis über den strukturellen Aufbau und die deskriptiven Elemente der Anwendungen und ihrer Einzelschichten. In diesem Abschnitt wird der Aufbau von J2EE-Anwendungen erläutert und ermittelt, welche Möglichkeiten des Zugriffs auf deren einzelne Komponenten bestehen.

3.2.1 Das Java-Archivformat

Das Java-Archivformat [Som98] bildet die Grundlage für alle Java-Anwendungsarchive. Es erlaubt das wahlweise unkomprimierte oder komprimierte Bündeln unterschiedlicher Dateien zu einem Archiv. Das Java-Archivformat baut auf dem ZIP-Dateiformat auf [PKW06] und nutzt dessen ZLIB-Kompression [DeGa96]. Typischerweise enthalten Java-Archivdateien (JAR-Dateien) die zu einer Java-Anwendung oder einem Java-Applet gehörigen Klassen-Dateien und Ressourcen. JAR-Dateien können mit einer digitalen Signatur versehen werden, um ihre Herkunft und Integrität zuzusichern.

3.2.2 Das J2EE-Anwendungsarchiv

Das Format der J2EE-Anwendungsarchive basiert auf dem Java-Archivformat. J2EE-Anwendungsdateien (EAR-Dateien) enthalten alle Modularchive und gegebenenfalls auch Bibliotheken, aus denen sich eine J2EE-Anwendung zusammensetzt. Zusätzlich enthält jede EAR-Datei mindestens einen J2EE-konformen Anwendungs-Deployment-Deskriptor, eine XML-Datei, die Angaben zu Konfigurationsaspekten beinhaltet, die die gesamte Anwendung betreffen. Insbesondere gibt dieser Deployment-Deskriptor Aufschluss über Art und Verwahrungsort der

einzelnen Anwendungsmodule. Abbildung 3.2 stellt den strukturellen Aufbau eines J2EE-Anwendungsarchivs dar.

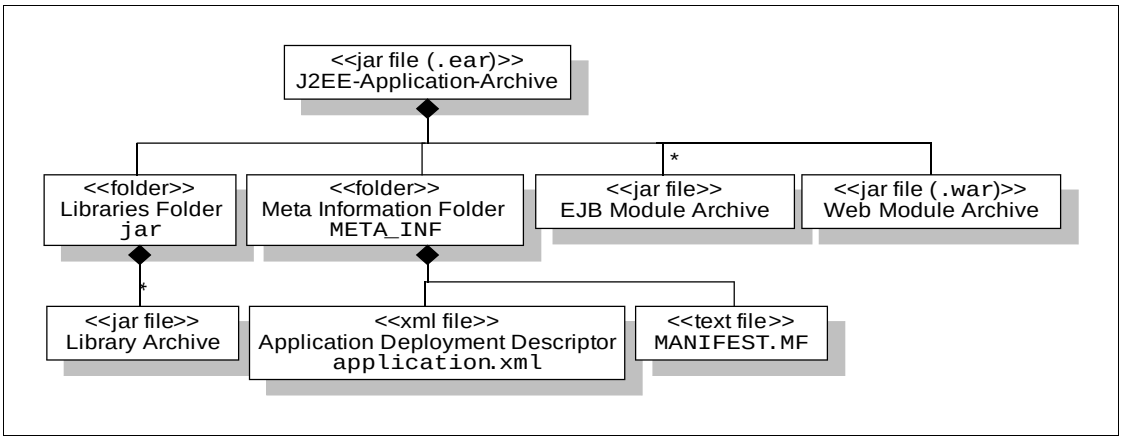

Abbildung 3.1: Aufbau eines J2EE-Anwendungsarchivs

3.2.3 Die wichtigsten J2EE-Modularchive

Die einzelnen Komponenten von J2EE-Anwendungen sind in Modulen organisiert. Solche Module sind wie die J2EE-Anwendung selbst in jeweils spezialisierten Modularchiven abgelegt. Im Rahmen der Konzepte dieser Arbeit sind Web-Module und EJB-Module die wichtigsten Komponenten Server-basierter J2EE-Anwendungen. Das Format und der Aufbau der Modularchive dieser beiden Modultypen wird im Folgenden jeweils kurz erläutert.

Das Web-Modularchiv

Die Komponenten eines J2EE-Web-Moduls werden in WEB-Modularchiven zusammengefasst. Wie bei J2EE-Anwendungsdateien handelt es sich bei Web-Modularchiven ebenfalls um spezialisierte Java-Archive. Ein Web-Modularchiv enthält alle Komponenten eines Web-Moduls. Hierbei handelt es sich um Java-Klassen-Dateien der Servlets und Session Beans sowie zusätzlicher Hilfsobjekte und die Java Server Pages. Darüber hinaus enthält jedes Web-Modularchiv wenigstens einen *Web-Deployment-Deskriptor*, eine XML-Datei, die unter Anderem Auskunft über die zum Web-Modul gehörigen Servlet-Klassen sowie den Einstiegspunkt in die Webanwendung gibt.

Das EJB-Modularchiv

EJB-Module werden in EJB-Modularchiven organisiert. Auch hierbei handelt es sich um ein spezialisiertes Java-Archivformat. Das EJB-JAR-Dateiformat ist ein Standardformat, das dazu dient, Enterprise Java Beans mit den zugehörigen deklarativen Informationen in einem gemeinsamen Archiv zusammenzufassen. EJB-JAR-Dateien beinhalten zum Einen die Java-Klassen-Dateien der Enterprise Beans sowie der dazu gehörigen Schnittstellen und zum Anderen einen *EJB-Deployment-Deskriptor*. Dieser EJB-Deployment-Deskriptor dient der Konfiguration der einzelnen Enterprise Java Beans, insbesondere der Festlegung der zugehörigen deklarierenden und implementierenden Java-Klassen.

3.2.4 Deployment-Deskriptoren

Ein Deployment-Deskriptor ist eine Konfigurationskomponente einer J2EE-Anwendung oder eines J2EE-Moduls. Er enthält Informationen darüber, wie eine Anwendung in einem J2EE-Anwendungsserver oder gegebenenfalls ein Modul in einem J2EE-Container zu installieren ist. Deployment-Deskriptoren befinden sich in Form von XML-Dokumenten in einem vorgegebenen Verzeichnis des jeweiligen Anwendungs- oder Modularchivs. Es wird zwischen J2EE-Deployment-Deskriptoren und J2EE-Server-spezifischen Deployment-Deskriptoren unterschieden. Die Form der J2EE-Deployment-Deskriptoren ist durch den J2EE-Standard vorgegeben. Diese standardisierten Konfigurationskomponenten enthalten allgemeine Informationen beispielsweise zu implementierenden Klassen der jeweiligen Komponente. J2EE-Server-spezifische Deployment-Deskriptoren verfügen über Konfigurationsdaten, die von der tatsächlichen Server-Umgebung abhängen. Dazu gehören zum Beispiel die JNDI-Namen einzelner Komponenten. In den folgenden Abschnitten werden die einzelnen J2EE-Deployment-Deskriptoren für J2EE-Anwendungen, EJB-Module und Web-Module jeweils bezüglich der für diese Arbeit wichtigen Analyseaspekte untersucht. Die J2EE-Server-spezifischen Deployment-Deskriptoren werden in dieser Arbeit nicht weiterführend behandelt, da diese gerade dem Anspruch dieser Arbeit widersprechen, die Konzepte zur Leistungsbewertung von J2EE-Anwendungen so zu entwickeln, dass sie für alle verfügbaren J2EE-Server umsetzbar sind. Darüber hinaus hat sich ergeben, dass die Informationen der J2EE-standardisierten Deployment-Deskriptoren für die Analyse im Rahmen einer Leistungsbewertung ausreichend sind.

Der Anwendungs-Deployment-Deskriptor

Der Deployment-Deskriptor einer J2EE-Anwendung befindet sich innerhalb des Anwendungsarchivs im Archiveintrag `META-INF/application.xml`. Es handelt sich um eine XML-Datei, die Konfigurationsaspekte festlegt, die eine J2EE-Anwendung als Ganzes betreffen. In Quelltext 3.1 wird der prinzipielle Aufbau eines Anwendungs-Deployment-Deskriptors exemplarisch dargestellt. Neben allgemeinen Informationen zu der J2EE-Anwendung gibt ein Anwendungs-Deployment-Deskriptor insbesondere Auskunft zu den Speicherorten der einzelnen, zur Anwendung gehörigen Module.

```
1:  <?xml version="1.0" encoding="UTF-8" ?>
2:  <application version="1.4" ...>
3:    <display-name>HelloJ2EE</display-name>
4:    <module>
5:      <web>
6:        <web-uri>HelloJ2EE-WebModule.war</web-uri>
7:        <context-root>/HelloJ2EE-WebModule</context-root>
8:      </web>
9:    </module>
10:   <module>
11:     <ejb>HelloJ2EE-EJBModule.jar</ejb>
12:   </module>
13: </application>
```

Quelltext 3.1: Exemplarischer Aufbau eines Anwendungs-Deployment-Deskriptors

Im Beispiel weist der Deployment-Deskriptor zwei Module aus. Ein Web-Modul im Archiv `HelloJ2EE-WebModule.ear` (Zeilen 4 bis 9) und ein EJB-Modul im Archiv `HelloJ2EE-EJBModule.jar` (Zeilen 10 bis 12). Solche Angaben sind für diese Arbeit von besonderem Interesse, da sie das automatisierte Auffinden von Modulen innerhalb eines Anwendungsarchivs ermöglichen.

Der EJB-Deployment-Deskriptor

Der Deployment-Deskriptor eines EJB-Modularchivs ist eine XML-Datei, die strukturelle Informationen über die im Archiv enthaltenen Enterprise Java Beans enthält [DeMi03, S. 48]. Der EJB-Deployment-Deskriptor befindet sich innerhalb des EJB-Modularchivs in dem Archiveintrag `META-INF/ejb-jar.xml`. Dieser Deployment-Deskriptor spezifiziert insbesondere die Typen der Entprise Java Beans und legt darüber hinaus die Klassen der Home und Component Interfaces sowie die Klassen mit der Implementierung der eigentlichen Funktionalität fest. Quelltext 3.2 stellt den generellen Aufbau eines EJB-Deployment-Deskriptors exemplarisch dar. Das dort spezifizierte Session Bean ist sowohl lokal als auch entfernt zugreifbar, da die entsprechenden Home und Component Interfaces angegeben sind (Zeilen 8 bis 11). Es wird von der Klasse `session.SimpleHelloBean` implementiert (Zeile 12). Solche Angaben dienen im Rahmen dieser Arbeit zunächst zum Auffinden aller EJB-Komponenten eines Anwendungsarchivs. Darüber hinaus können die entsprechenden Einträge angepasst werden, um statt auf die ursprünglich implementierenden Java-Klassen auf eine eigene Komponente mit gewünschter Zusatzfunktionalität zu verweisen.

```
1:  <?xml version="1.0" encoding="UTF-8" ?>
2:  <ejb-jar ...>
3:    <display-name>HelloJ2EE-EJBModule</display-name>
4:    <enterprise-beans>
5:      <session>
6:        <display-name>SimpleHelloSB</display-name>
7:        <ejb-name>SimpleHelloBean</ejb-name>
8:        <home>session.SimpleHelloRemoteHome</home>
9:        <remote>session.SimpleHelloRemote</remote>
10:       <local-home>session.SimpleHelloLocalHome</local-home>
11:       <local>session.SimpleHelloLocal</local>
12:       <ejb-class>session.SimpleHelloBean</ejb-class>
13:       <session-type>Stateless</session-type>
14:       <transaction-type>Container</transaction-type>
15:     </session>
16:       ...
17:   </enterprise-beans>
18:   ...
19: </ejb-jar>
```

Quelltext 3.2: Exemplarischer Aufbau eines EJB-Deployment-Deskriptors

Der Web-Deployment-Deskriptor

Konfigurationsaspekte eines Web-Moduls werden im Web-Deployment-Deskriptor festgelegt. Der Web-Deployment-Deskriptor ist eine XML-Datei, die sich im Archiveintrag `WEB-INF/web.xml` im Web-Modul-Archiv befindet. In Quelltext 3.3 wird der Aufbau eines Web-Deployment-Deskriptors exemplarisch dargestellt. Neben der Konfiguration der Sitzungsverwaltung und den Einstiegspunkten in die Web-Schnittstelle werden im Web-Deployment-Deskriptor die Servlet-Komponenten des Web-Moduls spezifiziert. Darüber hinaus ist es möglich, ein sogenanntes *Servlet Mapping* vorzunehmen. Damit wird veranlasst, dass durch das Anfragen eines anzugebenden URL-Musters immer ein bestimmtes Servlet angefordert wird. In Quelltextauszug 3.3 wird in den Zeilen 13 bis 16 solch eine Zuordnung vorgenommen. Der Aufruf von URLs der Form `http://<server>/<context-root>/action/<any-document>` wird immer an das `ControlServlet` geleitet. Dieser Mechanismus kann prinzipiell genutzt werden, um eine Filter-Komponente direkt in eine Web-Schnittstelle zu integrieren.

```
1:  <?xml version="1.0" encoding="UTF-8" ?>
2:  <web-app xmlns="http://java.sun.com/xml/ns/j2ee"  ...>
3:    <session-config>
4:      <session-timeout>30</session-timeout>
5:    </session-config>
6:    <welcome-file-list>
7:      <welcome-file>index.jsp</welcome-file>
8:    </welcome-file-list>
9:    <servlet>
10:     <servlet-name>ControlServlet</servlet-name>
11:     <servlet-class>servlet.Controller</servlet-class>
12:   </servlet>
13:   <servlet-mapping>
14:     <servlet-name>ControlServlet</servlet-name>
15:     <url-pattern>/action/*</url-pattern>
16:   </servlet-mapping>
17: </web-app>
```

Quelltext 3.3: Exemplarischer Aufbau eines Web-Deployment-Deskriptors

3.3 Analyse von Anwendungsschichten

Aufbauend auf den im Grundlagenkapitel vorgestellten Methoden zur Analyse von Java-Anwendungen, die ausschließlich im Bytecode vorliegen, werden in den folgenden Abschnitten in dieser Arbeit entwickelte Konzepte vorgestellt, die der Analyse einzelner Anwendungsschichten einer J2EE-Anwendung dienen. Insbesondere wird dabei auf die Ermittlung des Funktionsumfangs einer Anwendungsschicht sowie auf die Generierung möglichst realistischer Anfragen eingegangen, um die Grundlagen für die Konzeption und Implementierung entsprechender Prüfkomponenten zu schaffen.

3.3.1 Analyse einer Web-Schicht

Für die Analyse einer Web-Schicht bieten sich prinzipiell zwei mögliche Vorgehensweisen an. Einerseits besteht die Möglichkeit, das Anwendungsmodul, das die Web-Schnittstelle implementiert, zu untersuchen. Bei J2EE handelt es sich hierbei um ein Web-Modul, das entweder einzeln zur Verfügung gestellt wird oder als Bestandteil einer J2EE-Anwendung in eine J2EE-Anwendungsarchiv-Datei eingebunden ist. Alternativ zur statischen Untersuchung ist es auch möglich, das Laufzeitverhalten einer Web-Schnittstelle zu beobachten, indem HTTP-Anfragen und -Antworten während der Benutzung der Web-Schnittstelle mit Hilfe einer Web-Browser-Anwendung von einem integrierten Filter oder einer eigenständigen Proxy-Komponente aufgezeichnet werden. Beide Ansätze werden im Folgenden erläutert.

Analyse von Web-Modulen

Durch die Analyse eines Web-Moduls lassen sich statische Eigenschaften dieses Anwendungsmoduls ermitteln, die zum Zeitpunkt der Erstellung der Moduldatei feststanden. Es wurde bereits angesprochen, dass sich die Moduldatei eines J2EE-Web-Moduls (WAR-Datei) normalerweise innerhalb einer J2EE-Anwendungsarchiv-Datei (EAR-Datei) befindet. Um die WAR-Datei zu analysieren, muss sie zunächst im J2EE-Anwendungsarchiv aufgefunden und daraus extrahiert werden. Sowohl EAR- als auch WAR-Dateien sind prinzipiell aufgebaut wie ein Java-Archiv. Insbesondere handelt es sich also um Dateien, die im ZIP-Dateiformat vorliegen. Ein J2EE-Anwendungsarchiv enthält zusätzlich zu den Modulen der J2EE-Anwendung den Anwendungs-Deployment-Deskriptor, in dem unter anderem die Namen und Speicherungsorte der Moduldateien innerhalb des Anwendungsarchivs angegeben sind. Genau wie ein J2EE-Anwendungsarchiv enthält auch eine Web-Moduldatei einen Deployment-Deskriptor. In diesem sind unter anderem Informationen über Sitzungsdauer, Servlets und den Einstiegspunkt in die Web-Schnittstelle enthalten. Leistungsanforderungen an Web-Schnittstellen werden prinzipiell durch HTTP-Anfragen gestellt, die bestimmte Aktionen in Servlets und JSPs nach sich ziehen. Diese Aktionen hängen dabei häufig von Aufbau und Inhalt der URL oder auch von POST- oder GET-Daten ab, die Bestandteile einer HTTP-Anfrage sein können. Ein früher Ansatz dieser Arbeit war, den Funktionsumfang eines Web-Moduls zu ergründen, indem alle Servlets und JSPs ermittelt und bezüglich ihrer Formulare analysiert werden. Diesem Gedanken liegt zu Grunde, dass Anfragen an eine Web-Schnittstelle normalerweise in Form von HTTP-Anfragen gestellt werden, die um die Eingaben in Formularen angereichert sein können. Dieser Ansatz musste allerdings verworfen werden, da grundsätzlich Formulare zum Einen auch in bereits übersetzten Servlets vorkommen können und zum Anderen selbst JSP-basierte Formulare von Java-Code durchzogen sein können. Aus diesem Grund bezieht sich die Analyse der Web-Schicht im Rahmen dieser Arbeit im Wesentlichen auf die Analyse des Web-Deployment-Deskriptors und die Aufzeichnung der Kommunikation zwischen Webbrowser und Web-Schnittstelle.

Beobachten der Kommunikation mit einer Web-Schnittstelle

Dynamische Aspekte eines Web-Moduls lassen sich nicht durch eine Analyse des Anwendungsmoduls ermitteln, wenn kein Quelltext dieses Moduls zur Verfügung steht. Um trotzdem Aussagen über das Laufzeitverhalten einer Web-Schnittstelle treffen zu können ist es möglich, die Kommunikation eines Webbrowsers mit einer Web-Schnittstelle zu beobachten und gegebenenfalls aufzuzeichnen. Dieses Konzept ermöglicht die Erfassung funktionaler Abläufe während der eigentlichen Anwendungsnutzung. Solche Abläufe können als Testpläne für die

Leistungsbewertung beliebiger Server-basierter Anwendungen mit Web-Schnittstelle genutzt werden. Dieser Ansatz wird später in diesem Kapitel mit dem Konzept der Web-Filter-Komponente vertieft.

3.3.2 Analyse einer EJB-Schicht

Die EJB-Schicht einer J2EE-Anwendung wird durch EJB-Module realisiert. Auch bei der Analyse einer EJB-Schicht bestehen prinzipiell zwei mögliche Vorgehensweisen. Zum Einen ist es möglich, die EJB-Module zu untersuchen, aus denen sich die EJB-Schicht zusammensetzt, um statische Aspekte zu ermitteln, zum Anderen besteht die Möglichkeit, die EJB-Schicht zur Laufzeit zu beobachten, um dynamische Aspekte zu ermitteln. Beide Ansätze werden im Folgenden kurz erläutert.

Analyse von EJB-Modulen

Die statische Analyse von EJB-Modulen bezieht sich im Wesentlichen auf die Analyse des EJB-Deplyoment-Deskriptors, der Angaben über Typen und Schnittstellen der im EJB-Modul enthaltenen Enterprise Java Beans macht. Anhand der im EJB-Deployment-Deskriptor angegebenen Component Interfaces werden die Schnittstellen-Klassen der Enterprise Java Beans ermittelt. Diese können mit den Mitteln der Java-Reflection-API untersucht werden, um zu ergründen, welche Funktionen die Enterprise Java Beans anbieten. Dieser statische Ansatz wird vor allem in der prototypischen Implementierung angewendet, die im nächsten Kapitel vorgestellt wird.

Beobachten der EJB-Schicht zur Laufzeit

Um EJB-Komponenten während der Laufzeit beobachten zu können, ist es erforderlich, die entsprechenden EJB-Module um Filter-Funktionalität anzureichern. Diese Filter-Funktionalität stellt Aufrufe an den Enterprise Java Beans fest, ohne die Verarbeitung zu beeinflussen. Auf diese Weise ist es möglich, Laufzeitinformationen zu ermitteln, ohne den Anwendungsablauf zu verfälschen. Dieser Ansatz wird im Rahmen der Prüfkomponenten, insbesondere der EJB-Filter-Komponenten, vertieft, die in den nachfolgenden Abschnitten vorgestellt werden.

3.3.3 Analyse einer Datenhaltungsschicht

Wie bei der Analyse von Web- und EJB-Schichten existieren auch zwei Ansätze der Analyse von Datenhaltungsschichten. Die strukturelle Analyse einer Datenhaltungsschicht bezieht sich auf die Analyse der Datenstrukturen. Dem gegenüber steht wiederum die Ermittlung des Laufzeitverhaltens. Das Laufzeitverhalten einer Datenhaltungsschicht lässt sich im Allgemeinen durch das Beobachten der JDBC-Kommunikation ermitteln, da J2EE-Anwendungen normalerweise die JDBC-API nutzen, um mit Datenhaltungssystemen zu kommunizieren.

Analyse von Datenstrukturen

Die Analyse der Datenstrukturen des Datenhaltungssystems bezieht sich auf die Ermittlung des verwendeten Datenschemas. Leider findet sich bislang im SQL-Standard bislang keine

Funktionalität, die den Zugriff auf solche Metadaten ermöglicht. Durch diese Einschränkung wird die Analyse des Datenschemas außerordentlich erschwert. Prinzipiell bietet zwar praktisch jedes Datenbanksystem Mechanismen an, mit denen es möglich ist, Schemadaten auszulesen, häufig indem SQL-Anfragen auf vordefinierten Systemtabellen aufgerufen werden, allerdings fehlt auch hier eine Standardisierung, und eine Applikation, die in der Lage sein soll, solche Daten auszulesen, muss den Typ des zugrunde liegenden Datenbanksystems kennen.

Beobachten der JDBC-Kommunikation

Für den Zugriff auf die Datenbasis benötigt eine Datenhaltungsschicht, die die JDBC-API nutzt, einen JDBC-Treiber, der die Kommunikation mit der Datenbasis ermöglicht und ein Data-Source-Objekt, das Verbindungen zur Datenbasis herstellt, auf der Anfragen gestellt werden können. Zur Beobachtung der JDBC-Kommunikation ergeben sich daraus zwei mögliche Ansätze. Zum Einen ist es möglich, den JDBC-Treiber durch eine Filter-Komponente zu ersetzen, die alle an sie gestellten Anfragen registriert und dann an den eigentlichen JDBC-Treiber weiterleitet. Zum Anderen besteht die Möglichkeit, eine eigene Data-Source-Implementierung zu erstellen, die Verbindungsobjekte zur Datenbasis zur Verfügung stellt, die ebenfalls alle an sie gestellten Anfragen registrieren und dann an eine eigentliche Verbindung zur Datenbasis weiterleiten. Beide Ansätze werden im Rahmen der Prüfkomponenten für Datenhaltungssysteme später in diesem Kapitel vertieft.

3.4 Prüfkomponenten zur Kapselung von Anwendungsschichten

Ein wichtiger Aspekt dieser Arbeit ist die isolierte Betrachtung einzelner Anwendungsschichten. Um Aussagen über das Laufzeitverhalten einer einzelnen Anwendungsschicht machen zu können, ist es notwendig, die betrachtete Schicht während der Anwendungsnutzung isoliert zu beobachten. Die isolierte Betrachtung einer Anwendungsschicht kann durch die Kapselung mit Prüfkomponenten ermöglicht werden. Die Aufgabe einer Prüfkomponente besteht darin, Leistungsanforderungen an eine Schicht zu stellen, eine Schicht zu vertreten oder sie um zusätzliche Funktionalität zu erweitern. Aus der Anforderung, Aussagen über einzelne Schichten eines Anwendungssystems machen zu können, ergeben sich drei Klassen von Prüfkomponenten. *Driver-Komponenten* fordern eine Menge von Leistungen von der zu testenden Anwendungsschicht an, *Stub-Komponenten* ersetzen die Funktionalität von Anwendungsschichten und *Filter-Komponenten* reichern eine Anwendungsschicht um zusätzliche Funktionalität an. Abbildung 3.2 stellt exemplarisch zwei Einsatzmöglichkeiten von Prüfkomponenten in einer Anwendung mit Mehrschichtarchitektur dar. Abbildung 3.2a zeigt hierbei die Ausgangssituation einer klassischen Vier-Schicht-Anwendung. In Abbildung 3.2b wurde der Web-Schicht der Anwendung eine Filter-Komponente vorgeschaltet, die alle Anfragen an die Web-Schnittstelle registriert und weiterleitet. Darüber hinaus wurde auch der Verarbeitungsschicht eine Filter-Komponente vorgelagert, die Anfragen an die Verarbeitungsschicht entgegen nimmt und weiterleitet. Durch eine Auswertung der einzelnen Anforderungs- und Fertigstellungszeitpunkte ist es in dieser Kombination möglich, Während der normalen Anwendungsnutzung Antwortzeiten für beliebige Leistungen zu ermitteln. Darüber hinaus bietet dieser Einsatz von Filter-Komponenten die Möglichkeit, die anteiligen Antwortzeiten der Web-Schicht sowie der Verarbeitungsschicht zusammen mit den Datenhaltungsschicht zu ermitteln. Ein zusätzlicher Nutzen ergibt sich durch die Möglichkeit, mittels der Web-Filter-Komponente Anfragen aufzuzeichnen, die während der Anwendungsnutzung an die Web-Schnittstelle gesendet werden, um eine repräsentative Anfragemenge zu erhalten. Abbildung 3.2c zeigt letztlich einen Aufbau, der in der Lage ist, Antwortzeiten der Web-Schicht sowie der Verarbeitungsschicht automatisch zu ermitteln, ohne

dass Seiteneffekte in der Datenhaltung des Anwendungssystems auftreten. Eine Web-Driver-Komponente ist in dieser Zusammenstellung dafür verantwortlich, Anfragen an die Web-Schnittstelle der Anwendung zu stellen und die Zeitintervalle bis zu den Antworten festzuhalten. Die Anfragemengen könnten in diesem Fall zuvor mit einer Filter-Komponente vor der Web-Schicht aufgezeichnet worden sein. Zusätzlich wurde der Verarbeitungsschicht wiederum eine Filter-Komponente vorgeschaltet, die die Zeitpunkte aller Anfragen und Ergebnisse festhält. Die Datenhaltungsschicht wurde durch eine Stub-Komponente ersetzt, wodurch Seiteneffekte auf die Datenhaltungsschicht ausgeschlossen werden. Durch die Kombination von Driver-Komponente für die Web-Schnittstelle, Filter-Komponente vor der Verarbeitungsschicht und Stub-Komponente anstelle der Datenhaltung werden Web-Schicht und Verarbeitungsschicht quasi von Prüfkomponenten gekapselt, was eine isolierte Betrachtung ermöglicht.

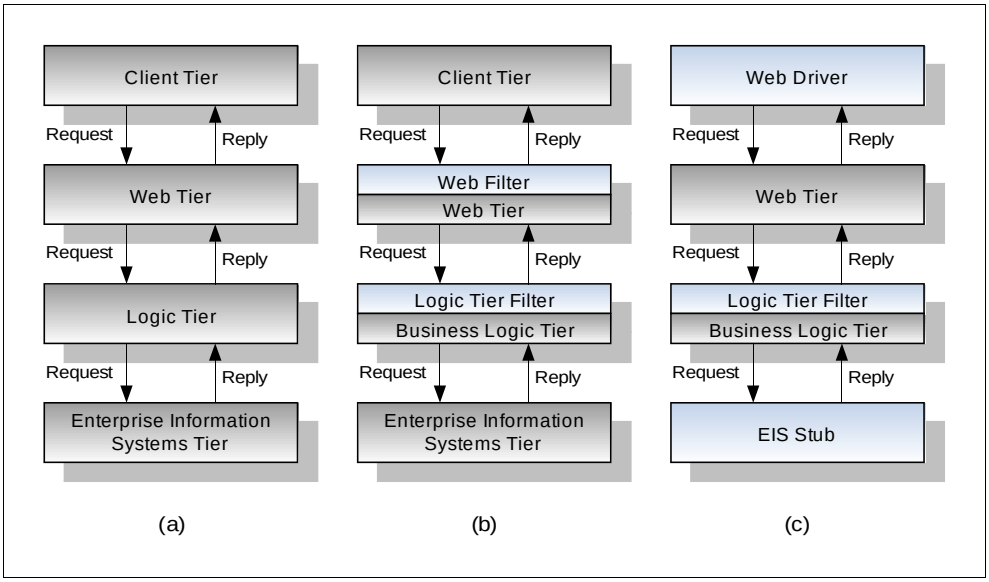

Abbildung 3.2: Exemplarische Einsatzmöglichkeiten von Prüfkomponenten

3.4.1 Driver-Komponenten

Eine Driver-Komponente stellt Leistungsanforderungen an eine bestimmte Schicht eines Anwendungssystems, misst die Antwortzeiten der angeforderten Leistungen und beurteilt in groben Zügen ob die Ergebnisse den Anforderungen entsprechen. Die Aufgabe einer Driver-Komponente besteht also zum Einen darin, das Verhalten des Benutzers beziehungsweise der übergeordneten Anwendungsschicht gegenüber der zu prüfenden Schicht eines Anwendungssystems nachzubilden, um Leistungen von dieser Anwendungsschicht automatisiert anfordern zu können. Zum Anderen muss eine Driver-Komponente in der Lage sein, die Antwortzeiten der angeforderten Leistungen zu ermitteln und sollte aus dem Ergebnis der Anforderung Aussagen darüber ableiten können, ob die Leistung erfolgreich erbracht wurde, oder ob ein Verarbeitungsfehler aufgetreten ist. Abbildung 3.2c stellt einen Anwendungsfall dar, in dem die Anfragen an die Web-Komponente eines Anwendungssystems durch eine Driver-Komponente gestellt werden. Eine Web-Driver-Komponente muss dazu in der Lage sein, HTTP-Anfragen zu generieren und diese mittels des Hypertext-Transfer-Protokolls an die Web-Komponente der Anwendung zu stellen. Darüber hinaus muss die Web-Driver-Komponente sowohl die Fähigkeit haben, HTTP-Antworten der Web-Komponente zu empfangen, um die Antwortzeiten zu ermitteln, als auch in der Lage sein, die empfangenen HTTP-Antworten bezüglich ihrer Qualität zu

beurteilen, um Aussagen darüber treffen zu können, ob eine Leistungsanforderung erfolgreich verarbeitet wurde.

3.4.2 Stub-Komponenten

Eine Stub-Komponente bietet die Schnittstellen einer Komponente an, ohne deren Funktionalität zu beherrschen. Stub-Komponenten dienen also als Ersatz oder Stellvertreter für ihre Ursprungskomponenten. Prinzipiell sieht das Stub-Konzept dieser Arbeit vor, dass eine Stub-Komponente die Ergebnisse der an sie gestellten Leistungsanforderungen selbstständig ermittelt, ohne auf andere Komponenten des Anwendungssystems zurückzugreifen. Dieses Verhalten bringt den Vorteil mit sich, dass, je nach Architektur und Schicht, Seiteneffekte bei der isolierten Betrachtung einer Anwendungsschicht reduziert oder sogar ausgeschlossen werden können. Abbildung 3.2c illustriert eine Situation, in der das Verhalten der Web-Schicht und der Logik-Schicht eines Anwendungssystems während einer Testdurchführung beobachtet werden können, ohne dass der Datenbestand geändert wird, da das Datenhaltungssystem der Anwendung durch eine entsprechende Stub-Komponente ersetzt wurde. In diesem Fall ist die Stub-Komponente in der Lage, per JDBC-Schnittstelle erhaltene SQL-Ausdrücke zu verarbeiten, die dem Datenschema des ursprünglichen Datenhaltungssystems entsprechen, und passende SQL-Ergebnismengen an die aufrufende Komponente zurück zu liefern. Die Anforderung, Seiteneffekte zu vermeiden und trotzdem Ergebnisse zu produzieren, die den Ergebnissen der Ursprungskomponente hinreichend ähnlich sind, um das Verhalten der darüber liegenden Anwendungsschichten nur wenig zu verändern, macht den Entwurf und die Umsetzung von Stub-Komponenten zu einer ausgesprochen komplexen Aufgabe. Die Schwierigkeit besteht insbesondere bei Anwendungsbereichen, in denen der Anwendungszustand die Ergebnisse von Anforderungen an die Ursprungskomponente beeinflusst. In solchen Fällen ist es praktisch überhaupt nicht möglich, von Stub-Komponenten generische und trotzdem angemessene Ergebnisse zu erhalten. In Situationen, in denen der Anwendungszustand die Anforderungsergebnisse kaum beeinflusst oder diese im Rahmen einer Sitzung feststehen, ist es möglich, dieses Problem durch das vorangegangene Aufzeichnen von Anforderungsergebnissen zu lösen.

3.4.3 Filter-Komponenten

Ähnlich dem Konzept der Stub-Komponente bieten Filter-Komponenten die Schnittstellen ihrer Ursprungskomponenten an, ohne deren Funktionalität selbst zu beherrschen. Im Gegensatz zu einer Stub-Komponente bedient sich eine Filter-Komponente allerdings der Funktionalität ihrer Ursprungskomponente, um angeforderte Leistungen zu erbringen, so dass eine Filter-Komponente sich aus Sicht einer aufrufenden Komponente wie die Ursprungskomponente verhält. Filter-Komponenten nehmen also an Leistungsanforderungen teil, ohne diese zu beeinflussen. Filter-Komponenten können genutzt werden, um das Laufzeitverhalten einer Anwendung zu ermitteln. Im Rahmen dieser Arbeit dienen Filter-Komponenten insbesondere dazu, Vorgänge aufzuzeichnen, die später einer Driver-Komponente als Testpläne dienen können.

3.4.4 Komposition von Anwendungsmodulen und Prüfkomponenten

Tabelle 3.1 stellt die möglichen Kombinationen der vorgestellten Prüfkomponenten und den Modulen und Bestandteilen einer typischen Web-basierten J2EE-Anwendung dar und bewertet sie bezüglich ihrer Relevanz für das hier vorgestellte Leistungsbewertungskonzept für

Anwendungssysteme mit Hinblick auf die Möglichkeit isolierter Betrachtung einzelner Anwendungsschichten. Die einzelnen Kompositionen werden im Folgenden jeweils vorgestellt. Außerdem werden die Anforderungen an die jeweilige Prüfkomponente ermittelt und darüber hinaus die angegebene Relevanz für die Leistungsbewertung im Kontext dieser Arbeit argumentativ belegt. Vorbereitend auf die prototypische Implementierung, die in Kapitel 4 vorgestellt wird, werden für die Prüfkomponenten, die als besonders relevant eingestuft wurden, zusätzlich mögliche Konzepte der Umsetzung im Rahmen von J2EE vorgestellt.

Tabelle 3.1: Relevante Driver- und Stub-Komponenten

	Driver-Komponente	**Stub-Komponente**	**Filter-Komponente**
Client-Anwendung	geringe Relevanz	ohne Relevanz	geringe Relevanz
Web-Modul	hohe Relevanz	geringe Relevanz	hohe Relevanz
EJB-Modul	hohe Relevanz	hohe Relevanz	hohe Relevanz
EIS	ohne Relevanz (bzgl. dieser Arbeit)	hohe Relevanz	hohe Relevanz

Prüfkomponenten für Client-Anwendungen

Eine Client-Anwendung einer J2EE-Anwendung kann in unterschiedlichen Ausprägungen vorliegen. Zum Einen kann es sich um eine komplexe Anwendung handeln, die unter Umständen in einem entsprechenden J2EE-Container ausgeführt wird und die von J2EE angebotenen Technologien nutzt, um mit den anderen Komponenten zu interagieren. Andererseits kann die Client-Anwendung auch nur einen einfachen Web-Browser umfassen, dessen Aufgabe einzig darin besteht, HTTP-Anfragen an das Web-Modul der Anwendung zu senden, vom Web-Modul generierte HTML-Inhalte darzustellen und gegebenenfalls vorhandene JavaScript-Elemente zu verarbeiten. Dieser Ansatz hat gegenüber einer komplexen Client-Anwendung den Vorteil, dass auf den Zielrechnern außer dem normalerweise vorhandenen Web-Browser keine weitere Anwendung installiert und gewartet werden muss. Aufgrund der geringen Verbreitung komplexer Anwendungs-Clients werden diese hier nicht weiter betrachtet. Die in den folgenden Abschnitten vorgestellten Client-Prüfkomponenten beziehen sich entsprechend auf den Einsatz eines Webbrowsers.

Web-Client-Driver

Eine Web-Client-Driver-Komponente hat die Aufgabe, einen Web-Browser so zu steuern, dass er sich verhält, als würde ein Anwender mit ihm eine Server-basierte Anwendung benutzen. Dies wäre zum Einen durch die Nutzung gegebenenfalls vorhandener Programmierschnittstellen des Browsers möglich. Ein anderer Ansatz wäre die Nutzung von Scripting-Mechanismen, die in vielen Betriebssystemen bereits vorhanden sind, indem die tatsächliche Benutzung des Web-Browsers durch einen Benutzer aufgezeichnet und später automatisiert wiederholt wird. Prinzipiell wäre es auf diese Weise möglich, den gesamten Funktionsumfang einer Server-basierten Anwendung abzudecken, allerdings besteht keine Möglichkeit, die Ergebnisse einer Aktion zu

beurteilen, hierfür müsste wiederum auf Programmierschnittstellen des Browsers zurückgegriffen werden. Da sich Web-Module mit erheblich einfacheren Mitteln mit Hilfe eines Web-Drivers ansprechen lassen, wurden Web-Client-Driver-Komponenten in dieser Arbeit als wenig relevant bewertet.

Web-Client-Stub

Die Aufgabe einer Web-Client-Stub-Komponente besteht darin, sich dem Benutzer gegenüber wie ein Web-Browser zu verhalten, ohne die Funktionalität einer Server-basierten Anwendung zu nutzen. In dieser Arbeit werden keine Ansätze zur Anwenderbewertung verfolgt. Daher sind Web-Client-Stub-Komponenten hier nicht relevant und werden nicht weiter betrachtet.

Web-Client-Filter

Eine Web-Client-Filter-Komponente hat die Aufgabe, die Aktionen eines Anwenders entgegen zunehmen und an einen Web-Browser weiterzuleiten, um die Ergebnisse dann wiederum dem Benutzer zu präsentieren. Tatsächlich ist der einzig denkbare Anwendungsfall die Aufzeichnung notwendiger Aktionen, für die Browser basierte Nutzung einer Anwendung. In dieser Hinsicht kann eine Web-Client-Filter-Komponente für eine entsprechende Web-Client-Driver-Komponente unter Umständen von gewisser Bedeutung sein und wurde daher insgesamt mit einer geringen Relevanz bewertet.

Prüfkomponenten für Web-Module

Im J2EE-Umfeld besteht die Aufgabe eines Web-Moduls darin, eine Web-Schnittstelle für eine Server-basierte Anwendung zur Verfügung zu stellen. Web-Prüfkomponenten dienen in diesem Zusammenhang der Aufzeichnung, Replikation oder Generierung und Verarbeitung von HTTP-Anfragen und -Antworten.

Web-Driver

Einer Web-Driver-Komponente kommt die Aufgabe zu, sinnvolle Anfragen an eine Web-Schnittstelle zu generieren und abzusetzen sowie die Antworten der Web-Schnittstelle zu empfangen und die Verarbeitungszeit zu berechnen. Die Erzeugung sinnvoller Anfragen erfordert eine vorangegangene Analyse der entsprechenden Web-Schnittstelle, damit eine möglichst vollständige funktionale Abdeckung erreicht wird. Um Anfragen an einen Web-Server zu senden und dessen Antworten empfangen zu können, muss eine Web-Driver-Komponente in der Lage sein, auf der Basis von HTTP zu kommunizieren. Darüber hinaus müssen dynamische Aspekte einer Web-Schnittstelle wie zum Beispiel die Sitzungsverwaltung oder die Cookie-Verwaltung behandelt werden. Um außerdem beurteilen zu können, ob angeforderte Leistungen erbracht wurden, muss eine Web-Driver-Komponente in der Lage sein, zumindest die empfangenen Statusmeldungen eines Webservers zu interpretieren. Web-Driver-Komponenten lassen sich nutzen, um Leistungsanforderungen an beliebige Web-Schnittstellen, für die sie konfiguriert wurden, zu senden und Leistungskennzahlen zu ermitteln. Web-Driver-Komponenten sind daher für diese Arbeit besonders relevant.
Eine Möglichkeit, eine Web-Driver-Komponente zu implementieren, ist die direkte Integration in den Web-Container eines J2EE-Servers. Dieser Ansatz ermöglicht es prinzipiell, Leistungsanforderungen mittels Java-Methodenaufrufen an eine Web-Schnittstelle abzusetzen. HTTP-Anfragen würden in diesem Fall also nicht per HTTP übertragen. Stattdessen würde die Kommunikation über Objekte abgewickelt, deren Klassen die Java-Schnittstellen

`HTTPServletRequest` und `HttpServletResponse` (beide aus dem Paket `javax.servlet.http`) implementieren. Eine weitere Möglichkeit der Implementierung einer Web-Driver-Komponente stellt die Umsetzung als eigenständige Anwendung dar, die Anfragen per HTTP an den Web-Container eines J2EE-Servers sendet und dessen Antworten empfängt. Die Umsetzung innerhalb des Web-Containers hat gegenüber einer eigenständigen Anwendung den Vorteil, dass die Sitzungsverwaltung nicht behandelt werden muss, da das zustandslose Protokoll HTTP ausgespart wird. Allerdings erfordert die Integration einer Testkomponente in einen J2EE-Container unter Umständen einen tiefen Eingriff in die Architektur des Containers. Dies bedeutet wiederum, dass die resultierende Testkomponente auf die Implementierung eines bestimmten Web-Containers abgestimmt sein müsste, was der Forderung nach der Unabhängigkeit von einzelnen J2EE-Server-Implementierungen widerspricht. Zudem hat die Umsetzung einer Web-Driver-Komponente als eigenständige, per HTTP angebundene Anwendung den Vorteil, dass aufgrund der losen Kopplung auch nicht-lokale Web-Container angesprochen werden können. Im Gegensatz zu einer integrierten Lösung würde in solch einem Fall die eigentliche Leistung einer Web-Schnittstelle durch die Web-Driver-Komponente nicht beeinflusst, da sie sich an der Web-Schnittstelle praktisch nicht von einem tatsächlichen Benutzer unterscheiden lässt.

Web-Stub

Der Anspruch an eine Web-Stub-Komponente besteht darin, HTTP-Anfragen eines Browsers entgegen zunehmen und diese zu beantworten, ohne eine Verarbeitungsleistung zu erbringen. Dieses Verhalten ließe sich prinzipiell nutzen, um die Verarbeitungsgeschwindigkeit von Webbrowsern zu ermitteln. Da dies aber nicht Gegenstand dieser Arbeit ist, sind Web-Stub-Komponenten in diesem Zusammenhang nur wenig relevant.

Web-Filter

Eine Web-Filter-Komponente nimmt HTTP-Anfragen eines Browsers entgegen, ermittelt den verantwortlichen Webserver und leitet Anfragen an diesen weiter. Vom Webserver empfangene HTTP-Antworten leitet eine Web-Filter-Komponente umgekehrt an den Browser weiter. Bis zu diesem Punkt entspricht das Verhalten einer Web-Filter-Komponente dem Konzept eines HTTP-Proxys. Die Aufgabe einer Web-Filter-Komponente besteht im Gegensatz zu einem HTTP-Proxy allerdings nicht darin, Aufrufe an einer Web-Schnittstelle durch Zwischenspeicherung von Anfrageergebnissen zu beschleunigen und die Netzlast dahinter liegender Netzwerkabschnitte zu verringern. Vielmehr nimmt eine Web-Filter-Komponente an der Kommunikation zwischen Webbrowser und -server teil, ohne diese zunächst zu beeinflussen. Das Konzept einer derartigen Komponente ist offensichtlich geeignet, Arbeitsabläufe an der Web-Schnittstelle einer Server-basierten Anwendung aufzuzeichnen. Solche Aufzeichnungen bilden die Grundlage für die Konfiguration von Web-Driver-Komponenten, daher haben Web-Filter-Komponenten für diese Arbeit eine entsprechend hohe Relevanz.
Genau wie Web-Driver-Komponenten lassen sich Web-Filter-Komponenten prinzipiell sowohl innerhalb des Web-Containers implementieren als auch in Form einer eigenständigen Anwendung. Eine Umsetzung innerhalb des Web-Containers hat zwar ebenfalls den Vorteil, dass die Sitzungsverwaltung nicht behandelt werden müsste, da diese bereits vom Container bereitgestellt wird. Andererseits besteht aber auch hier der Nachteil, dass unter Umständen Eingriffe in die Architektur des Containers vorgenommen werden müssen, die wiederum Auswirkungen auf das Leistungs- und Laufzeitverhalten der Web-Schnittstelle selbst haben können. Die Alternative, eine Web-Filter-Komponente als eigenständige Anwendung umzusetzen, die praktisch zwischen Webbrowser und Web-Schnittstelle steht könnte nahtlos an dem verbreiteten HTTP-Proxy-Konzept ansetzen. Jeder moderne Webbrowser unterstützt die Konfiguration eines HTTP-Proxys, an den

alle Anfragen gestellt und Antworten erwartet werden. Dies hat den Vorteil, dass eine als HTTP-Proxy implementierte Web-Filter-Komponente faktisch in der Lage ist, Anfragemengen an beliebige Web-Schnittstellen aufzuzeichnen. Die Einschränkung auf das J2EE-Umfeld gilt also bei dieser Alternative nicht.

Prüfkomponenten für Enterprise-Java-Beans-Module

Ein EJB-Modul kapselt die Verarbeitungslogik einer Server-basierten J2EE-Anwendung. Die Ausführung der Komponenten des EJB-Moduls, der Enterprise Java Beans, findet im EJB-Container des J2EE-Applikationsservers statt. Prinzipiell beziehen sich die in den folgenden Abschnitten vorgestellten Prüfkomponenten auf alle drei EJB-Typen. In der Praxis werden aber sehr häufig nur Session Beans und (seltener) Message Driven Beans als Schnittstelle für übergeordnete Schichten verwendet, während Entity Beans meist ausschließlich innerhalb des EJB-Containers für den Zugriff auf die Datenhaltung genutzt werden. Die Aufgabe von EJB-Prüfkomponenten besteht im Wesentlichen darin, Anfragen an EJB-Komponenten aufzuzeichnen und zu replizieren oder realistische Anfragen zu generieren, sowie solche Anfragen gegebenenfalls zu verarbeiten.

EJB-Driver

Eine EJB-Driver-Komponente muss in der Lage sein, sinnvolle Anfragen an EJB-Komponenten zu generieren, diese an die EJB-Komponenten zu stellen, Verarbeitungsergebnisse entgegen zunehmen und diese bezügliche Erfolg oder Misserfolg der Verarbeitung zu bewerten sowie die Verarbeitungsdauer zu berechnen. Da die zu generierenden Anfragen möglichst alle Anwendungsfälle eines dahinter stehenden Anwendungssystems abdecken sollen, stellt sich an dieser Stelle wiederum das Problem der automatischen Generierung sinnvoller Anfragen. Wie bereits bei dem Konzept des Web-Drivers ist auch hier eine vorangehende Analyse des entsprechenden Anwendungsmoduls notwendig. Web-Driver-Komponenten sind ein wichtiges Werkzeug für die isolierte Betrachtung der Verarbeitungsschicht einer J2EE-Anwendung und sind daher für diese Arbeit besonders relevant.
Prinzipiell ist es möglich, EJB-Driver-Komponenten sowohl für die Ausführung in einem J2EE-Container als auch als eigenständige Anwendung zu entwickeln. Der Aufwand für die Umsetzung als eigenständige Anwendung ist aber deutlich größer als eine Container-basierte Lösung, da in diesem Fall die Auffindung von Enterprise Java Beans und das Aufrufen von Methoden an diesen in der EJB-Driver-Komponente implementiert werden muss, während ein J2EE-Container diese Anforderungen bereits erfüllt. Die Umsetzung einer EJB-Driver-Komponente für die Ausführung in einem J2EE-Container erfordert den Einsatz eines entsprechend konfigurierten Containers, der insbesondere über die JNDI-Namen und die zugehörigen Enterprise Java Beans verfügen muss. Prinzipiell ist die Implementierung solch einer EJB-Driver-Komponente für jede der drei relevanten Container-Typen möglich. Die Umsetzung für einen Applikationscontainer ermöglicht das Darstellen einer Benutzungsschnittstelle, die alle diesbezüglichen Möglichkeiten von Java ausschöpfen kann. Eine Umsetzung für einen Web-Container bietet sich an, wenn die Steuerung der EJB-Driver-Komponente möglichst ohne vorbereitende Maßnahmen von beliebigen Arbeitsstationen aus möglich sein soll. Letztendlich besteht auch die Möglichkeit, eine EJB-Driver-Komponente direkt für einen EJB-Container zu implementieren. Der letzte Ansatz hat den Vorteil, dass es mit ihm auch möglich ist, Enterprise-Java-Bean-Objekte anzusprechen, die nicht über ein Remote Interface verfügen.

EJB-Stub

Eine EJB-Stub-Komponente hat die gleichen Schnittstellen wie das zugehörige Enterprise Java Bean, insbesondere handelt es sich bei ihr also offensichtlich selbst um ein Enterprise Java Bean. Ein Aufruf einer EJB-Stub-Komponente führt aber nicht zu einer entsprechenden Verarbeitung, sondern gegebenenfalls zur Rückgabe eines generischen oder vorab ermittelten Ergebniswertes. In diesem Sinne ist eine EJB-Stub-Komponente also in der Lage, ein Enterprise Java Beans zu ersetzen, ohne dass dies in den übergeordneten Schichten eines Anwendungssystems Auswirkungen hätte. Dies hat insbesondere den Vorteil, dass Seiteneffekte in untergeordneten Anwendungsschichten ausgeschlossen werden können, da beispielsweise Zugriffe auf Datenhaltungskomponenten nicht mehr durchgeführt werden. Um die Verarbeitungsschicht einer J2EE-Anwendung vollständig durch Stub-Komponenten darzustellen, ist es notwendig, jedes Enterprise Java Bean eines EJB-Moduls durch eine entsprechende EJB-Stub-Komponente zu ersetzen. Das resultierende Modul kann dann das ursprüngliche EJB-Modul im Anwendungssystem ersetzen. Ein solches EJB-Stub-Modul ermöglicht die isolierte Betrachtung übergeordneter Anwendungsschichten ohne das Auftreten von Seiteneffekten und mit sehr kurzen Störzeiten die durch die Verarbeitung (also gegebenenfalls das Generieren oder Abrufen eines passenden Rückgabewertes) in den Stub-Komponenten entstehen, die aber bei geschickter Implementierung exakt bestimmbar sind. Aus diesem Grund sind EJB-Stub-Komponenten für diese Arbeit ausgesprochen relevant.
Da es sich bei einer EJB-Stub-Komponente um ein Enterprise Java Bean handelt, muss sie offensichtlich für die Ausführung innerhalb eines EJB-Containers implementiert werden. Eine EJB-Stub-Komponente implementiert alle Schnittstellen der zu ersetzenden Enterprise-Java-Bean-Klasse, das bedeutet insbesondere auch, dass die verschiedenen EJB-Schnittstellen (`RemoteHome`, `Remote`, `Home` und `Local`) identisch sind. Aus diesem Grund lassen sich EJB-Stub-Komponenten recht einfach in ein EJB-Modul einfügen. Hierfür wird im Deployment-Deskriptor des EJB-Moduls die implementierende Klasse eines Enterprise Java Beans durch die Klasse der EJB-Stub-Komponente ersetzt. Die eigentliche Herausforderung bei der Umsetzung von EJB-Stub-Komponenten liegt darin, ihr Verhalten so zu gestalten, dass sie entsprechend dem ursprünglichen Enterprise Java Bean auf Anfragen reagieren, ohne eine Verarbeitung durchzuführen. Außerdem ist es im Sinne einer teilautomatisierten Leistungsbewertung erforderlich, passende EJB-Stub-Komponenten zu einer J2EE-Anwendung zu generieren.

EJB-Filter

Die Aufgabe einer EJB-Filter-Komponente besteht darin, Anfragen, die eigentlich an ein Enterprise Java Bean (das Ziel-EJB) gerichtet sind, entgegen zunehmen und gegebenenfalls an das Ziel-EJB weiterzuleiten. Wie eine EJB-Stub-Komponente verfügt auch eine EJB-Filter-Komponente zu diesem Zweck über die gleichen Schnittstellen wie ihr Ziel-EJB, es handelt sich also bei ihr selbst ebenfalls um ein Enterprise Java Bean. EJB-Filter-Module eignen sich einerseits als Werkzeug zum Aufzeichnen des Verhaltens der EJB-Komponenten eines Anwendungssystems, darüber hinaus ist es mit ihnen möglich, anteilige Antwortzeiten der übergeordneten Anwendungsschichten von denen der darunter liegenden Anwendungsschichten zu trennen, indem die Zeitintervalle in denen Verarbeitung in einem Enterprise Java Bean stattfindet aufgezeichnet und einer beobachtenden Komponente (beispielsweise einer Web-Driver-Komponente) zugänglich gemacht werden. EJB-Filter-Komponenten wurden dementsprechend für diese Arbeit als sehr relevant eingestuft.
Bei einer EJB-Filter-Komponente handelt es sich ebenso wie bei einer EJB-Stub-Komponente um ein Enterprise Java Bean. Folglich muss auch eine EJB-Filter-Komponente für die Ausführung innerhalb eines EJB-Containers implementiert werden. Dadurch, dass jede EJB-Filter-Komponente ebenfalls alle Schnittstellen der jeweils ursprünglichen Enterprise-Java-Bean-Klasse implementiert, lassen sich EJB-Filter-Komponenten ebenso einfach durch Anpassungen im Deployment-

Deskriptor in ein EJB-Modul einpflegen wie EJB-Stub-Komponenten. Wie bei den EJB-Stub-Komponenten liegt auch hier die eigentliche Herausforderung in der automatischen Generierung von EJB-Filter-Komponenten zu jedem Enterprise Java Bean. Tatsächlich ist es möglich, EJB-Stub-Komponenten und EJB-Driver-Komponenten mittels einer gemeinsamen Oberklasse umzusetzen. In diesem Ansatz wird Zusatzfunktionalität in die ursprünglichen EJB-Komponenten eingefügt, die es ermöglicht, das Verhalten bezüglich Weiterleiten von Anfragen oder eigenständigem Generieren von Rückgabewerten von Außen zu steuern. Auf diese Art der Umsetzung wird in Kapitel 4 genauer eingegangen.

Prüfkomponenten für Datenhaltungssysteme

Der Zugriff auf Datenhaltungssysteme wird in J2EE-Anwendungen normalerweise mittels der JDBC-Technologie realisiert. Prinzipiell ist auch die Nutzung der bereits angesprochenen JCA-Module möglich, um Datenhaltungssysteme mit J2EE-Anwendungen zuzugreifen, da dies in der Praxis aber nur sehr selten Verwendung findet, wird dieser Ansatz hier nicht verfolgt. Entsprechend beziehen sich die im Folgenden vorgestellten Prüfkomponenten für Datenhaltungssysteme auf die Verwendung der JDBC-API.

JDBC-Driver

Die Aufgabe einer JDBC-Driver-Komponente im Sinne dieser Arbeit besteht darin, Anfragemengen zu generieren und an JDBC-Verbindungen zu stellen, wobei auch hier wieder das Problem besteht, Anfragemengen zu generieren, die einen möglichst großen Teil der zu prüfenden Funktionalität abdecken. Das JDBC-Driver-Konzept dieser Arbeit ist trotz der namentlichen Ähnlichkeit nicht mit den JDBC-Treibern zu verwechseln, bei denen es sich um Java-Klassen handelt, die den JDBC-Zugriff auf Datenquellen ermöglichen. Das hier verwendete JDBC-Driver-Konzept bildet vielmehr eine Umgebung zur Durchführung von Benchmarks auf Datenbanken. Hierbei handelt es sich prinzipiell um einen bedeutenden Ansatz, da die Leistungsfähigkeit eines Datenhaltungssystems normalerweise starken Einfluss auf die Leistung eines darauf aufbauenden Anwendungssystems hat. Die Analyse und die Ermittlung der Leistungsfähigkeit von Datenbanksystemen wurde und wird allerdings bereits hinreichend behandelt. Dies gilt insbesondere im Rahmen der schon erwähnten TPC-Lasttests [Smi01]. Da sich diese Arbeit außerdem vorrangig mit der Bewertung eigentlicher Anwendungssysteme und nicht der zu Grunde liegenden Datenbanksysteme beschäftigt, wurden JDBC-Driver-Komponenten im Rahmen dieser Arbeit als nicht relevant eingestuft.

JDBC-Stub

Eine JDBC-Stub-Komponenten dient einem Anwendungssystem während einer Leistungsbewertung als Ersatz der eigentlichen JDBC-Datenquelle. Die Aufgabe der JDBC-Stub-Komponente besteht darin, an sie gestellte Anfragen zu beantworten, ohne diese an die tatsächliche Datenhaltungskomponente weiterzuleiten. JDBC-Stub-Komponenten könnten insbesondere eingesetzt werden, um Seiteneffekte in der Datenbasis während eines Leistungstests zu verhindern. Aus diesem Grund sind sie für diese Arbeit sehr relevant.
Es sind grundsätzlich drei Ansätze zur Implementierung einer JDBC-Stub-Komponente denkbar. Zum Einen ist es möglich, die JDBC-Stub-Komponente direkt als JDBC-Treiber zu entwickeln, der normalerweise dafür zuständig ist, JDBC-Anfragen an die eigentliche Datenbasis weiterzuleiten. Eine als JDBC-Treiber konzipierte Stub-Komponente würde Anfragen allerdings nicht an die Datenbasis weiterleiten, sondern selbst Antworten beziehungsweise Ergebnismengen generieren.

Eine zweite Möglichkeit ist die Anfertigung eigener Implementierungen der Schnittstellen `javax.sql.DataSource` und `java.sql.Connection`. Klassen, die die Data-Source-Schnittstelle implementieren sind dafür verantwortlich, der Anwendung Verbindungen zur Datenbasis zur Verfügung zu stellen. Bei diesen Verbindungen zur Datenbasis handelt es sich um Objekte von Klassen, die die Connection-Schnittstelle implementieren. Die Umsetzung einer eigenen Connection-Klasse würde nun ebenfalls Anfragen entgegennehmen, hätte aber keine tatsächliche Verbindung zur Datenbasis, sondern würde Antworten beziehungsweise Ergebnismengen selbst generieren. Die Aufgabe der eigenen Data-Source-Implementierung ist es, die Anwendung mit diesen Stub-Verbindungen zu versorgen. Die größere Schwierigkeit liegt auch bei JDBC-Stub-Komponenten wieder darin, realitätsnahe Daten zu gegebenen Anfragen zu generieren, ohne auf die tatsächliche Datenbasis zurückzugreifen. Hierfür existieren grundsätzlich drei Lösungswege. Als erstes ist es möglich, zu jeder Anfrage Zufallsdaten zu generieren. Auf diese Weise erhielte man Ergebnismengen, die bestenfalls noch dem ursprünglichen Schema der Datenbasis entsprächen. Zusätzlich ist es möglich, wiederum vorweg die Kommunikation mit der JDBC-Schnittstelle während der Anwendungsnutzung aufzuzeichnen, und diese Aufzeichnung als Basis für die Generierung vernünftiger Ergebnismengen zu verwenden. Mit diesem Verfahren wären die Ergebnismengen prinzipiell der Anwendungssituation angemessen und auch ein Sitzungszustand würde beachtet werden. Allerdings ist es nicht immer möglich, für alle Situationen Daten vor zuhalten, da auch die selben Anfragen in höheren Anwendungsschichten (zum Beispiel in Abhängigkeit von der Uhrzeit) prinzipiell zu unterschiedlichsten Anfragen an die Datenbasis führen können. In solchen Fällen müsste dann wiederum auf Zufallsdaten, beziehungsweise eine zufällige Auswahl aufgezeichneter Daten, zurückgegriffen werden. Die dritte Möglichkeit, eine JDBC-Stub-Komponente zu implementieren, ist eigentlich ein hybrider Ansatz zwischen Filter- und Stub-Komponente und weicht von dem Prinzip ab, dass eine Stub-Komponente die zu ersetzende Ursprungskomponente nicht nutzen darf. Eine JDBC-Stub-Komponente kann auch ohne die Nutzung von Zufallsdaten und die Notwendigkeit vorweg JDBC-Kommunikation aufzuzeichnen umgesetzt werden, indem zwischen lesenden und schreibenden JDBC-Anfragen unterschieden wird. Lesende JDBC-Anfragen werden einfach gemäß dem Filter-Ansatz an die Ursprungskomponente weitergeleitet. Nach Erhalt einer Antwortmenge von der Ursprungskomponente wird diese wiederum an die aufrufende Komponente zurückgegeben. Schreibende JDBC-Anfragen werden nicht an die Ursprungskomponente weitergeleitet, sondern stets mit einer Erfolgsmeldung quittiert. Dieser Ansatz liefert zu jeder lesenden Anfrage eine aktuelle Ergebnismenge, die offensichtlich maximal realistisch ist, da sie der tatsächlichen Ergebnismenge der Datenbasis entspricht. Trotzdem werden Seiteneffekte in der Datenbasis vermieden, da schreibende Zugriffe faktisch ignoriert werden. Allerdings gehen Anwendungszustände, die über die Datenbasis verfolgt werden, bei diesem Ansatz verloren.

JDBC-Filter

Eine JDBC-Filter-Komponente nimmt an der Kommunikation zwischen der Verarbeitungsschicht und der Datenhaltungsschicht teil, ohne diese zu beeinflussen. Sie nimmt alle JDBC-Anfragen der Verarbeitungsschicht entgegen und leitet diese an die Datenhaltungsschicht weiter, von welcher sie gegebenenfalls eine Ergebnismenge erhält, die sie wiederum an die Verarbeitungsschicht weiterleitet. JDBC-Filter-Komponenten spielen für die isolierte Betrachtung einzelner Anwendungsschichten bei der Leistungsbewertung eine ähnlich große Rolle, wie JDBC-Stub-Komponenten. Mit ihnen ist es zwar nicht ohne weiterführende Maßnahmen möglich, die Datenbasis während einer Leistungsbewertung vor Seiteneffekten zu schützen, wenigstens können aber deren Laufzeitinformationen ermittelt werden. Zusätzlich können mit JDBC-Filter-Komponenten JDBC-Anfragen aufgezeichnet werden, die während der Nutzung einer Anwendung

anfallen, um diese Anfragemengen später als Datengrundlage für eine JDBC-Stub-Komponente zu verwenden.
Die Implementierung einer JDBC-Filter-Komponente ist ähnlich einer JDBC-Stub-Komponente wiederum sowohl als JDBC-Treiber als auch als Umsetzung der Connection- und der Data-Source-Schnittstelle möglich. Allerdings würde eine JDBC-Stub-Komponente in beiden Fällen die ursprüngliche Implementierung nutzen, um an sie gestellte Anfragen entsprechend weiterzuleiten.

3.4.5 Grenzen der isolierten Beobachtung von Anwendungsschichten

Mit den in den vorangegangenen Abschnitten vorgestellten Prüfkomponenten wurde ein Satz von Konzepten entwickelt, der es ermöglicht, einzelne Anwendungsschichten einer Server-basierten J2EE-Anwendung gekapselt zu betrachten und hinsichtlich des anteiligen Antwortzeitverhaltens einer Leistung in den einzelnen Anwendungsschichten zu untersuchen. Durch die gekapselte Betrachtung einzelner Anwendungsschichten lassen sich bezüglich jeder Schicht anteilige Antwortzeiten von Leistungen ermitteln. Damit ist es möglich, die genaue Interaktion zwischen den einzelnen Anwendungsschichten bei der Erbringung von Leistungen nachzuvollziehen, und diese gegebenenfalls als Interaktionsdiagramm darzustellen. Auf diese Weise können nicht nur Rückschlüsse auf Entwurfsentscheidungen gezogen und diese bezüglich ihrer Umsetzung bewertet werden, auch die Lokalisierung von Leistungsengpässen und die quantitative Ermittlung der Aufrufpotenzierung zwischen einzelnen Schichten wird ermöglicht.
Das beschriebene Verfahren funktioniert allerdings nur bei exklusiver Nutzung der J2EE-Anwendung durch die Prüfkomponenten. Darüber hinaus darf auch die Prüfkomponente Anfragen ausschließlich seriell an die Anwendung stellen. Wie in Abbildung 4.1 dargestellt, würden parallele Anfragen an die Web-Schnittstelle einer J2EE-Anwendung unter Umständen zur gleichzeitigen Verarbeitung mehrerer Anforderungen im EJB-Container führen. Da die Prüfkomponenten keine Möglichkeit haben, eine Verarbeitung im EJB-Container einer HTTP-Anfrage zuzuordnen, kann die Interaktion zwischen den Anwendungsschichten nun nicht nachvollzogen werden.

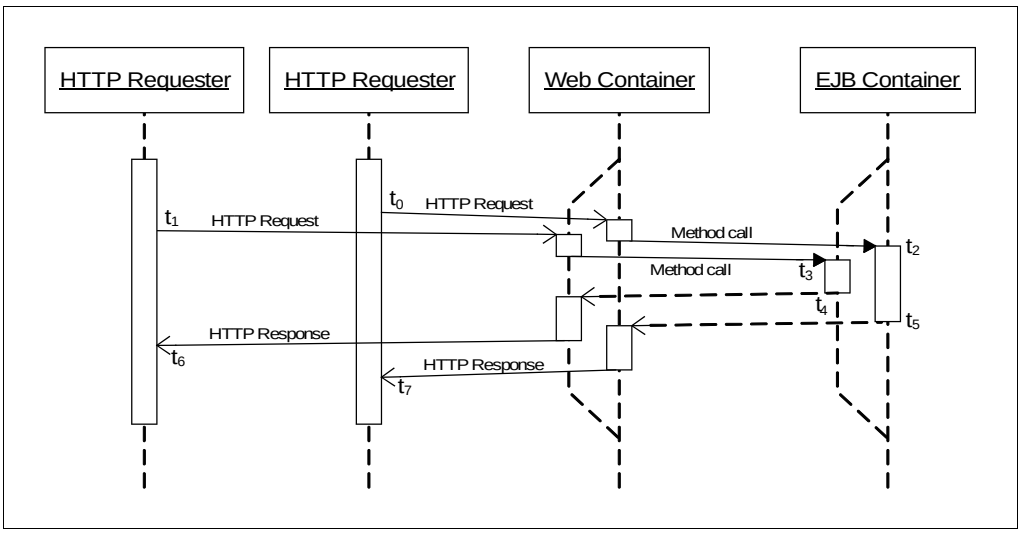

Abbildung 3.3: Betrachtung zweier paralleler Anfragen an eine Web-Schnittstelle

Abgesehen von der genannten Einschränkung ist es aber prinzipiell möglich, mit den vorgestellten Prüfkomponenten auch Prüfungen mit parallelen Anfragen durchzuführen. Auf diese Weise lassen sich dann entsprechende Aggregate der Leistungskennzahlen ermitteln. Zum Beispiel wäre es

möglich, eine Menge HTTP-Anfragen zu stellen. Aus den Zeitintervallen, in denen Leistungen im EJB-Container erbracht werden und insbesondere aus den Zeitintervallen, in denen im EJB-Container keine Leistung erbracht wird, kann die Kapazität des Web-Containers im Vergleich zur Kapazität des EJB-Containers bezüglich der Anzahl gleichzeitiger Anfragen ermittelt werden.

3.5 Erstellung von Testplänen

Es wurde bereits in den vorangegangenen Abschnitten darauf hingewiesen, dass es sich bei der Erstellung von Testplänen ganz und gar nicht um eine triviale Aufgabe handelt. Da die Erstellung von Testplänen für die Leistungsbewertung im Allgemeinen besonders tief gehende Kenntnisse über die zu prüfende Anwendung erfordert, ist die manuelle Erstellung unter Umständen die einzige Möglichkeit, einen angemessenen Ablaufplan zu erstellen. Eine automatische Generierung von Testplänen ist bestenfalls durch eingehende Analyse des Anwendungsquelltextes möglich. Da im Rahmen dieser Arbeit auf die Analyse von Anwendungsquelltext verzichtet wird, wird eine Teilautomatisierung der Erstellung von Testplänen angestrebt. Das Ziel ist hierbei, ein Konzept zu entwickeln, das im Gegensatz zu der manuellen Erstellung von Testplänen nicht auf Nutzer angewiesen ist, denen genaue strukturelle Zusammenhänge der zu testenden Anwendung bekannt sind. Dem gegenüber stehen Nutzer, die zwar über keinerlei Programmiererfahrung verfügen, die aber mit der zu testenden Anwendung vertraut sind und damit umgehen können. Idealerweise handelt es sich hierbei um die eigentlichen Nutzer der Anwendung. Mit diesen Anwendungsnutzern und dem Einsatz von Filter-Komponenten an den wesentlichen Schichtübergängen ist es möglich, sinnvolle und realistische Testpläne aufzuzeichnen, die den tatsächlichen Anforderungen an das zu testende Anwendungssystem entsprechen. Über die reine Aufzeichnung von Abläufen hinaus können diese Testpläne erweitert werden, indem für Nutzungsparameter nicht nur die ursprünglich aufgezeichneten Werte verwendet werden, sondern es darüber hinaus ermöglicht wird, Testpläne mit Wertmengen beziehungsweise Wertbereichen zu parametrisieren.

3.6 Ausführen von Testläufen und Auswertung der Ergebnisse

Um Testläufe nach vorangegangener Erstellung von Testplänen durchzuführen, ist es erforderlich, Prüfkomponenten, also eine Driver-Komponente und gegebenenfalls Filter- und Stub-Komponenten zu einem Prüfsystem zu verknüpfen. Je nach Art der Umsetzung der Filter- und Stub-Komponenten kann es hierfür notwendig sein, die zu prüfende Anwendung um diese Komponenten anzureichern. Die Driver-Komponente sollte in ein Werkzeug eingebettet sein, das ihre Steuerung ermöglicht und außerdem die Auswertung der Ergebnisse unterstützt. Die Steuerung einer Driver-Komponente kann das einfache Vermitteln eines Testplans und das Initiieren eines Prüfvorgangs bedeuten. Falls die Umsetzung der Driver-Komponente und der Typ des Testplans es zulassen, kann hier aber auch die Konfiguration mehrfacher Testläufe und die Parametrisierung des Testplans erfolgen. Die Unterstützung der Auswertung kann von der einfachen Ausgabe der ermittelten Kennzahlen bis zur Generierung von Graphen und Diagrammen reichen, die auf den erhobenen Daten basieren.

3.7 Bezug zur Methodik der Leistungsbewertung

In Kapitel 2 wurde eine Methodik vorgestellt, die eine Leistungsbewertung in vier Stufen vorsieht. In den vorangegangenen Abschnitten wurden Konzepte für die Quelltext-lose Leistungsbewertung

Server-basierter J2EE-Anwendungen mit Mehrschichtarchitektur entwickelt. In diesem Abschnitt werden die vorgestellten Konzepte den einzelnen Stufen der Leistungsbewertungsmethodik zugeordnet, um aufzuzeigen, an welchen Stellen dieser Methodik die entwickelten Konzepte einsetzbar sind.

Der erste Schritt der Methodik der Leistungsbewertung umfasst die Spezifikation der relevanten Leistungskriterien. Das heißt, dass zu diesem Zeitpunkt Kennzahlen festgelegt werden, deren Werte für die zu prüfende Anwendung von Interesse sind. Hierbei handelt es sich um die formalen Kennzahlen, die in Abschnitt 2.3 entwickelt und vorgestellt wurden. Die größte Bedeutung kommt in einer normalen Leistungsprüfung wahrscheinlich der Antwortzeit und dem Durchsatz zu. Für eine strukturelle Leistungsprüfung, die in der Lage ist, Performanz-Schwachstellen in einem Anwendungssystem zu lokalisieren, sind insbesondere anteilige Antwortzeiten der einzelnen Anwendungsschichten interessant, da sich hieraus anhand der in Abschnitt 2.3.2 vorgestellten Methode der Interaktionsverlauf und damit auch die Aufrufpotenzierung für diesen Ablauf bestimmen lassen. Zur Ermittlung der gewünschten Leistungskennzahlen ist es erforderlich, entsprechende Driver-Komponenten zu entwickeln, die den Konzepten der vorangegangenen Abschnitte folgen.

Im nächsten Schritt der Methodik der Leistungsbewertung wird die Analyse und Simulation des Einsatzes der zu prüfenden Anwendung behandelt. Dies umfasst vor allen Dingen die Erstellung von Testplänen. Im Rahmen dieser Arbeit wurde gezeigt, dass das Aufzeichnen von Anwendungsvorgängen eine gute Möglichkeit ist, realistische Testpläne zu erzeugen. Das Konzept der Filter-Komponenten eignet sich durch Einsatz an den interessanten Schichtübergängen zur Aufzeichnung von Anwendungsvorgängen. Die Entwicklung und Integration solcher Filter-Komponenten ist entsprechend notwendig. Im nächsten Kapitel wird gezeigt werden, dass wenigstens die Generierung und Integration von EJB-Filter-Komponenten in das EJB-Modul einer J2EE-Anwendung voll automatisiert werden kann. Es wird darüber hinaus ein Werkzeug benötigt, das die Filter-Komponenten steuert und es erlaubt, aufgezeichnete Vorgänge zu persistieren.

Der dritte Schritt der Methodik der Leistungsbewertung behandelt die Erstellung der Prüfmethode. An dieser Stelle werden zuvor aufgezeichnete Testpläne zusammengestellt und konfiguriert. Dies kann sowohl die Parametrisierung der Testpläne als auch die Einstellung von Wiederholungen, zu prüfenden Parallitätsgraden sowie Denk- und Erholungszeiten bedeuten. Hierfür wird ein Werkzeug benötigt, das in der Lage ist, zuvor persistierte Testpläne zu laden und vorzubereiten.

In der vierten und letzten Stufe der Methodik der Leistungsbewertung wird die Prüfung des Anwendungssystems schließlich durchgeführt. Dies geschieht durch die Nutzung der entsprechenden Driver-Komponenten. Um eine Prüfung durchzuführen ist es erforderlich, Driver-Komponenten mit vorbereiteten Testplänen zu versehen und den Prüfvorgang zu starten. Hierfür ist ein Werkzeug notwendig, das die Driver-Komponenten beinhaltet und zugänglich macht. Außerdem kommt diesem Werkzeug die Aufgabe zu, die Auswertung der Ergebnisse zu unterstützen, also wenigstens die Kennzahlen auszugeben, die ermittelt wurden.

3.8 Fazit

In dem vorangegangenen Kapitel wurden zunächst die Möglichkeiten der Analyse von J2EE-Anwendungen und deren Einzelschichten ermittelt. Darauf hin wurde das Konzept der Prüfkomponenten zur Kapselung von Anwendungsschichten vorgestellt und in Beziehung zu den Schichten einer J2EE-Anwendung untersucht. Daraus ergab sich eine Menge möglicher Prüfkomponenten für die isolierte Untersuchung und Betrachtung von Anwendungsschichten vor und während einer Leistungsbewertung. Auf den Konzepten der Prüfkomponenten aufbauend

wurde ermittelt, wie realistische Testpläne teilautomatisch generiert werden können und welche Möglichkeiten bestehen, solche Testpläne auszuführen, um Leistungskennzahlen zu ermitteln. Abschließend wurde der Bezug zu der im zweiten Kapitel vorgestellten Methodik der Leistungsbewertung hergestellt. Insgesamt hat das vorangegangene Kapitel einen konzeptionellen Komplex erstellt, der die Leistungsbewertung Server-basierter J2EE-Anwendungen ermöglicht und gleichzeitig Mechanismen anbietet, die es erlauben, einzelne Anwendungsschichten zu betrachten.

4 Implementierung der Leistungsbewertungsplattform

Dieses Kapitel beschäftigt sich mit der prototypischen Implementierung der Konzepte, die im vorangegangenen Kapitel vorgestellt wurden. Einerseits soll in diesem Kapitel belegt werden, dass sich die entwickelten Konzepte für eine programmatische Umsetzung und den praktischen Einsatz eignen, und andererseits soll dem Leser ein Zugang zu der prototypischen Implementierung geschaffen werden, um einen einfacheren Einstieg für gegebenenfalls nachfolgende Projekte zu ermöglichen. Im Rahmen dieser Arbeit ist eine prototypische Implementierung der Konzepte entstanden, die eine gekapselte Betrachtung der Web-Schicht einer Server-basierten J2EE-Anwendung ermöglichen. In diesem Kapitel werden zunächst die Anforderungen aufgezeigt, die an eine Implementierung gestellt wurden. Darauf hin wird ein Überblick über die Architektur vermittelt und die möglichen Anwendungsfälle aus Benutzersicht erläutert. Nach einer kurzen Vorstellung der verwendeten Entwurfsmuster, werden die einzelnen Komponenten der prototypischen Implementierung und deren Funktionsweise sowie die Besonderheiten bei der programmatischen Umsetzung untersucht.

4.1 Anforderungen

Das Ziel der prototypischen Implementierung ist zu zeigen, dass die gekapselte Betrachtung der Web-Schicht einer Server-basierten J2EE-Anwendung möglich ist, ohne den Quelltext dieser Anwendung zu nutzen. Gemäß dem vorangegangenen Kapitel ist es hierfür erforderlich, das Konzept des Web-Filters, des Web-Drivers und des EJB-Stubs oder des EJB-Filters programmatisch umzusetzen. Diese Prüfkomponenten ermöglichen die in Abbildung 4.1 dargestellten Konfigurationen. Die Anordnung in Abbildung 4.1a ermöglicht die Aufzeichnung von HTTP-Anfragefolgen, die während der Anwendungsnutzung auftreten. Die Anordnung in Abbildung 4.1b dient der nachfolgenden Prüfung der Web-Schicht. Die Web-Driver-Komponente nutzt dabei HTTP-Anfragefolgen als Testpläne, die mittels der Web-Filter-Komponente aufgezeichnet wurden.

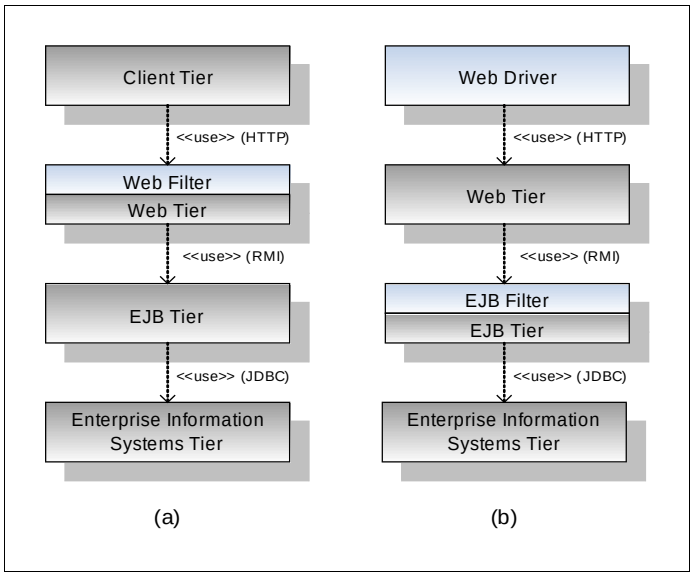

Abbildung 4.1: Mögliche Konfigurationen der prototypischen Prüfkomponenten

4.2 Architektur

Die prototypische Implementierung der Leistungsbewertungsplattform besteht aus drei Grundkomponenten. Es handelt sich hierbei um den HTTP-Werkzeugkasten, den J2EE-Anwendungsinjektor und den Laufzeitinformationsdienst. Der HTTP-Werkzeugkasten stellt die Web-Filter-Komponente und die Web-Driver-Komponente zur Verfügung. Die Aufgabe des J2EE-Anwendungsinjektors besteht darin, Zusatzfunktionalität in J2EE-Anwendungen zu injizieren, die dazu dient, mittels des Laufzeitinformationsdienstes Angaben zur Laufzeit von im EJB-Container erbrachten Leistungen an die Web-Driver-Komponente zu übertragen. Der HTTP-Werkzeugkasten und der Anwendungsinjektor stellen Prüfkomponenten zur Verfügung, während es sich beim Laufzeitinformationsdienst um eine Hilfskomponente handelt.

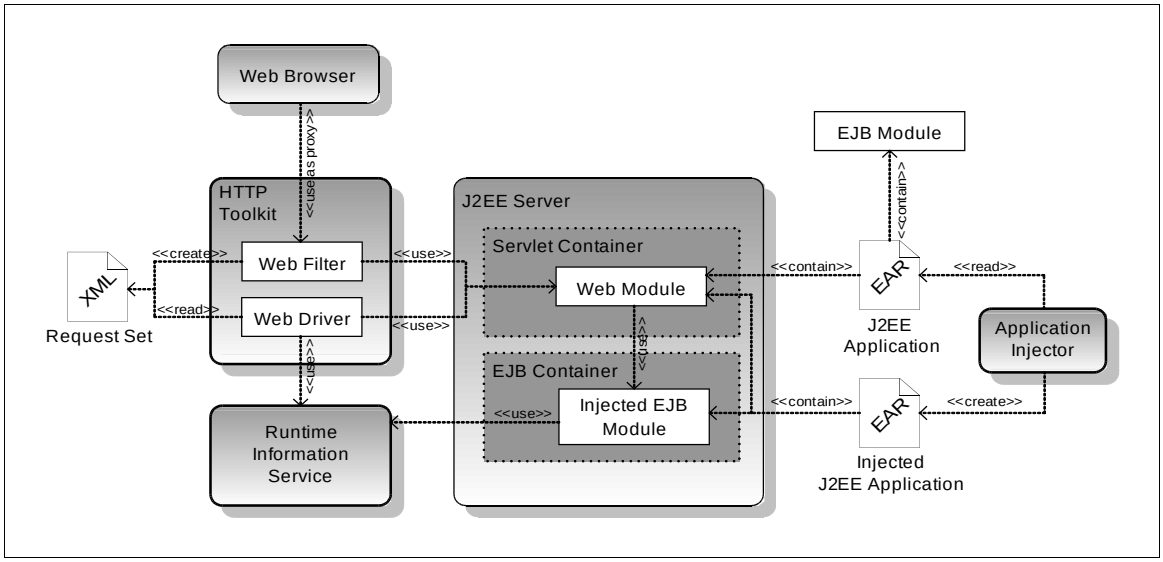

Abbildung 4.2: Das Zusammenspiel der Komponenten der Leistungsbewertungsplattform

Abbildung 4.2 stellt das Zusammenspiel der einzelnen Komponenten der Leistungsbewertungsplattform und den Komponenten der Laufzeitumgebung dar. Ein Webbrowser benutzt beim Zugriff auf die zu prüfende Anwendung die Web-Filter-Komponente des HTTP-Werkzeugkastens als HTTP-Proxy, um eine Anfragemenge aufzuzeichnen. Mit dem J2EE-Anwendungsinjektor wird das EJB-Modul der zu prüfenden Anwendung um Filter-Funktionalität erweitert. Während einer Prüfung ermöglicht diese Filter-Funktionalität das Sammeln von EJB-Laufzeitinformationen, sowie deren Übermittlung an den Laufzeitinformationsdienst. Mit der Web-Driver-Komponente des HTTP-Werkzeugkastens werden zuvor aufgezeichnete Abfragen an die injizierte, zu prüfende Anwendung gesendet und entsprechende Antwortzeiten ermittelt. Die Antwortzeiten des Web-Moduls in Kombination mit den Laufzeitinformationen der EJB-Komponenten lassen Rückschlüsse auf die Interaktion zwischen Web-Schnittstelle und EJB-Modul zu. Insbesondere können Laufzeit-intensive Leistungen in ihren Komponenten identifiziert werden.

4.3 Anwendungsfälle

Die prototypische Implementierung der vorgestellten Leistungsbewertungskonzepte ermöglicht die Untersuchung einer J2EE-Anwendung bezüglich des Verhaltens der Web-Schicht gegenüber darunter liegenden Anwendungsschichten. Dies geschieht durch die Ermittlung anteiliger Antwortzeiten der Web-Schicht und darunter liegender Schichten. Das Anwendungsfalldiagramm in Abbildung 4.3 zeigt, dass die Ermittlung anteiliger Antwortzeiten im Rahmen dieser Arbeit drei Anwendungsfälle beinhaltet. Diese sind das Aufzeichnen von Arbeitsabläufen, die Vorbereitung einer J2EE-Anwendung und die Durchführung einer Prüfung. Diese drei Vorgänge werden in den folgenden Abschnitten jeweils aus Benutzersicht erläutert. Dabei wird wieder der Zusammenhang zu der Methodik der Leistungsbewertung hergestellt, die in Kapitel 2 vorgestellt wurde. Der erste Schritt dieser Methodik umfasst die Spezifikation relevanter Leistungskennzahlen. Dies geschieht bei der Nutzung der prototypischen Implementierung implizit, da diese eigens für die Ermittlung anteiliger Antwortzeiten und der Ableitung der Aufrufpotenzierung umgesetzt wurde.

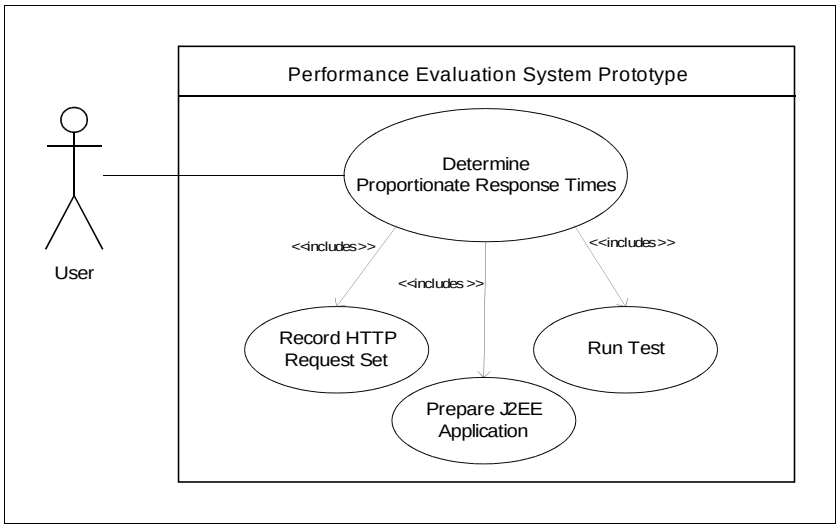

Abbildung 4.3: Hauptanwendungsfall des prototypischen Leistungsbewertungssystems

4.3.1 Aufzeichnen von Arbeitsabläufen

Die Aufzeichnung von Arbeitsabläufen entspricht dem zweiten Schritt der Leistungsbewertungsmethodik. Sie ermöglicht es, Testpläne zu erzeugen, die den tatsächlichen Anforderungen an eine Server-basierte Anwendung entsprechen. Das Interaktionsübersichtsdiagramm in Abbildung 4.4 stellt die hierfür notwendigen Schritte dar. Bevor eine Anwendung genutzt werden kann muss sie offensichtlich installiert werden. In diesem Fall handelt es sich um eine Server-basierte J2EE-Anwendung, die entsprechend in einem gegebenenfalls zu startenden J2EE-Server installiert werden muss. Nach der Vorbereitung der Anwendung, für die Arbeitsabläufe aufgezeichnet werden sollen, müssen der HTTP-Werkzeugkasten und ein Webbrowser gestartet werden. Beim Start befindet sich der HTTP-Werkzeugkasten im Aufnahmemodus, das heißt, seine Web-Filter-Komponente ist aktiv und zeichnet alle (auf TCP/IP Port 8484) an sie gestellten Anfragen auf und leitet sie an den eigentlichen Webserver weiter. Falls noch nicht geschehen, muss der Webbrowser so konfiguriert werden, dass er den HTTP-Werkzeugkasten als HTTP-Proxy verwendet. Nach dieser Vorbereitung wird der

Webbrowser zur Steuerung der Anwendung genutzt. Die HTTP-Kommunikation erscheint dabei in der Baumansicht des HTTP-Werkzeugkastens (zu sehen in Abbildung 4.7, Seite 58). Nach der Durchführung eines Arbeitsablaufes wird dieser mit dem HTTP-Werkzeugkasten in eine XML-Datei gespeichert. Dieser Vorgang wird für alle relevanten Arbeitsabläufe wiederholt. Die XML-Dateien dienen der Web-Driver-Komponente später als Testplan für die Durchführung einer Prüfung.

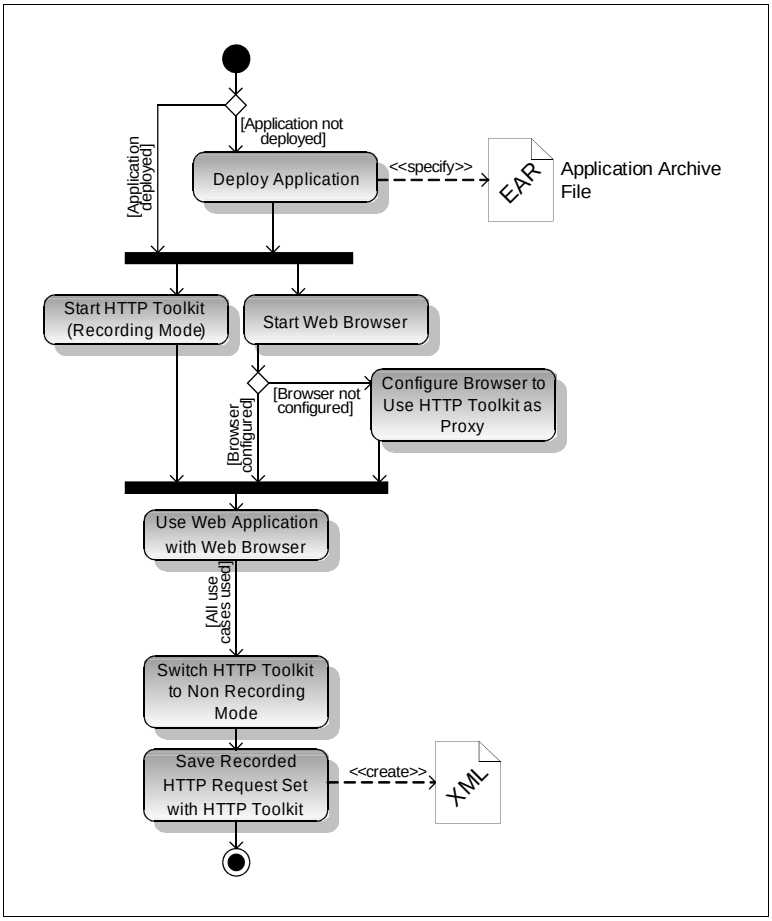

Abbildung 4.4: Aufzeichnen eines Arbeitsablaufs mittels des HTTP-Werkzeugkastens

4.3.2 Vorbereiten einer J2EE-Anwendung

Nachdem Arbeitsabläufe einer zu prüfenden Anwendung aufgezeichnet wurden ist es im Prinzip bereits möglich, diese Anwendung bezüglich einiger Leistungskennzahlen zu prüfen. Beispielsweise könnten Antwortzeiten und Durchsätze ermittelt werden. Das Ziel der prototypischen Implementierung der Leistungsbewertungsplattform ist jedoch die Ermittlung anteiliger Antwortzeiten der Web-Schicht und der darunter liegenden Anwendungsschichten. Hierfür ist es erforderlich, die zu prüfende Anwendung um zusätzliche Funktionalität anzureichern, die es ermöglicht, Laufzeitinformationen bezüglich der EJB-Schicht zu ermitteln. Im Interaktionsübersichtsdiagramm in Abbildung 4.5 werden die notwendigen Schritte der Injizierung von Zusatzfunktionalität in eine J2EE-Anwendung dargestellt. Zunächst muss der J2EE-Anwendungsinkjektor gestartet werden. Falls es die J2EE-Anwendung erfordert, müssen dabei auch Anwendungsbibliotheken gegenüber der Java-Laufzeitumgebung bekannt gegeben

werden. Der J2EE-Anwendungsinjektor fordert den Benutzer auf, ein Quell-J2EE-Anwendungsarchiv zu spezifizieren. Hierbei handelt es sich um die zu prüfende Anwendung. Als nächstes muss ein Ziel-J2EE-Anwendungsarchiv angegeben werden, das von dem Injektor erstellt wird. Während des Injizierens werden diverse Statusmeldungen ausgegeben, die unter Anderem Auskunft über den Quelltext der eingefügten Funktionalität geben. Diese Vorbereitung einer J2EE-Anwendung kann keinem der Schritte der Leistungsbewertungsmethodik direkt zugeordnet werden. Prinzipiell handelt es sich um eine Vorbereitung zum vierten Schritt, der Durchführung der Prüfung.

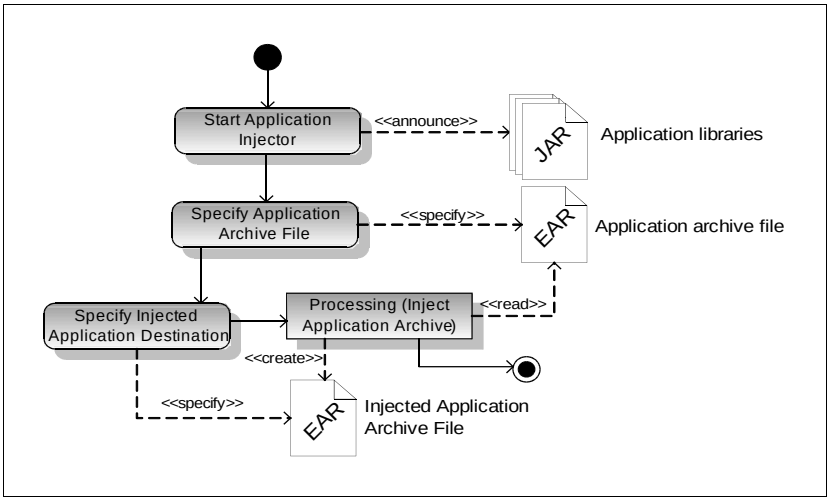

Abbildung 4.5: Injizieren von Zusatzfunktionalität in eine J2EE-Anwendung

4.3.3 Durchführung einer Prüfung

Nachdem mindestens ein Testplan erstellt und die zu prüfende J2EE-Anwendung vorbereitet wurde, kann die eigentliche Prüfung zur Ermittlung anteiliger Antwortzeiten durchgeführt werden. Bei der prototypischen Umsetzung umfasst dieser Vorgang auch die Auswahl von Testplänen und gegebenenfalls einer Untermenge von Anfragen. Bezüglich der Leistungsbewertungsmethodik werden also die beiden letzten Schritte behandelt. Das Interaktionsübersichtsdiagramm in Abbildung 4.6 stellt den Ablauf einer Prüfung dar. Falls noch nicht geschehen, muss die vorab vorbereitete J2EE-Anwendung zunächst im Anwendungsserver installiert werden. Außerdem muss der Laufzeitinformationsdienst gestartet werden, um die Kommunikation zwischen der Web-Driver-Komponente des HTTP-Werkzeugkastens und der in die J2EE-Anwendung injizierten EJB-Filter-Komponenten zu vermitteln. Im nächsten Schritt wird der HTTP-Werkzeugkasten gestartet und die gewünschte Anfragemenge (ein zuvor aufgezeichneter Arbeitsablauf) geladen, die als Testplan genutzt werden soll. Der HTTP-Werkzeugkasten schaltet in diesem Moment automatisch den Aufnahmemodus ab. Nun können gewünschte Anfragen ausgewählt werden, um diese auszuführen. Alternativ ist es aber auch möglich, alle Anfragen der geladenen Anfragemenge auszuführen. Während der Durchführung der Prüfung werden Statusmeldungen zu der Ausführung der einzelnen Anfragen ausgegeben, in denen die gesamte sowie die anteiligen Antwortzeiten angegeben sind. Nachdem alle gewünschten Anfragen ausgeführt wurden, wird aus den anteiligen Antwortzeiten ein Interaktionsdiagramm generiert und angezeigt. Dadurch wird die Aufrufstruktur und -potenzierung einzelner Leistungen dargestellt. Die Statusmeldungen sowie das generierte Interaktionsdiagramm der Prüfung einer Testanwendung sind in Abbildung 4.7 auf Seite 58 dargestellt.

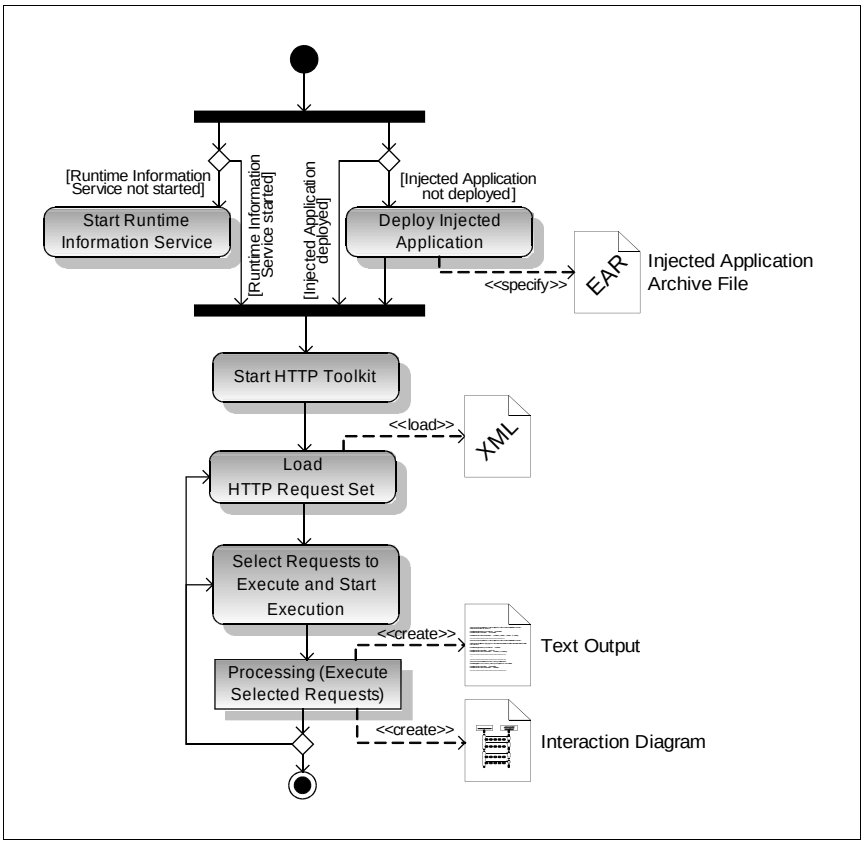

Abbildung 4.6: Durchführung einer Prüfung

4.4 Verwendete Entwurfsmuster

Für die prototypische Implementierung der Prüf- und Hilfskomponenten wurden verschiedene Entwurfsmuster der *Gang of Four* [GHJV01] verwendet. Diese Entwurfsmuster werden im Folgenden jeweils kurz erläutert und die Entscheidung für ihre Verwendung argumentiert.

4.4.1 Das Beobachtermuster

Das Beobachtermuster gehört zur Kategorie der Verhaltensmuster. Das Beobachtermuster ermöglicht einem Objekt (in diesem Zusammenhang Subjekt genannt) die Weitergabe von Ereignissen an abhängige Objekte, die Beobachter. Dafür müssen die Klassen der Beobachter eine Beobachterschnittstelle umsetzen, mit der sie sich an dem Subjekt registrieren können. Bei Eintritt eines Ereignisses werden von dem Subjekt die entsprechenden Methoden an der Beobachterschnittstelle aller registrierten Beobachter aufgerufen. Dieses Vorgehen birgt den Vorteil, dass Subjekte und Beobachter unabhängig von einander geändert werden können, da sie ausschließlich über eine definierte Schnittstelle interagieren.

4.4.2 Die Fabrikmethode

Die Fabrikmethode gehört zur Kategorie der Erzeugungsmuster. Die Fabrikmethode definiert eine Schnittstelle (die Erzeugerschnittstelle) zur Erzeugung von Objekten (den Produkten). Es fällt

dabei in den Verantwortungsbereich der konkreten Erzeuger, von welcher Klasse die konkreten Produkte sind. Der Einsatz der Fabrikmethode ist flexibler als die direkte Erzeugung von Objekten, da an den Stellen, an denen Produktobjekte genutzt werden sollen bei dem Bezug über eine Fabrik keinerlei Wissen über deren Erzeugung benötigt wird.

4.4.3 Das Stellvertretermuster

Das Stellvertretermuster gehört zur Kategorie der Strukturmuster. Das Stellvertretermuster sieht ein Stellvetreterobjekt vor, das sich nach Außen wie das vertretene Objekt (in diesem Zusammenhang Subjekt genannt) verhält, indem es Anfragen an das Subjekt weiterleitet. Stellvertreter können genutzt werden, um an den eigentlichen Zugriff auf ein Subjekt weitere Funktionalität zu binden, während das Subjekt von dieser Zusatzfunktionalität unabhängig bleibt. In diesem Sinne ist das Stellvertretermuster ein geeignetes Muster zur Umsetzung des in Kapitel 3 vorgestellten Filterkonzeptes.

4.5 Der HTTP-Werkzeugkasten

Der HTTP-Werkzeugkasten vereint zwei Komponenten in einer Benutzungsschnittstelle, eine Web-Filter-Komponente und eine Web-Driver-Komponente. Die Web-Filter-Komponente kann als HTTP-Proxy genutzt werden, der in der Lage ist, HTTP-Anfragen aufzuzeichnen. Aufgezeichnete Anfragen können mit dem HTTP-Werkzeugkasten benannt, gruppiert und in Form einer XML-Datei gespeichert werden. Mit der Web-Driver-Komponente des HTTP-Werkzeugkastens ist es möglich, zuvor aufgezeichnete und gespeicherte HTTP-Anfragemengen zu laden und als Testpläne für die Durchführung von Prüfungen zu verwenden. Während einer Prüfung gibt der HTTP-Werkzeugkasten Statusmeldungen mit den ermittelten (anteiligen) Antwortzeiten aus, nach erfolgter Prüfung wird ein Interaktionsdiagramm des Prüfungsverlaufs erstellt (Abbildung 4.7).

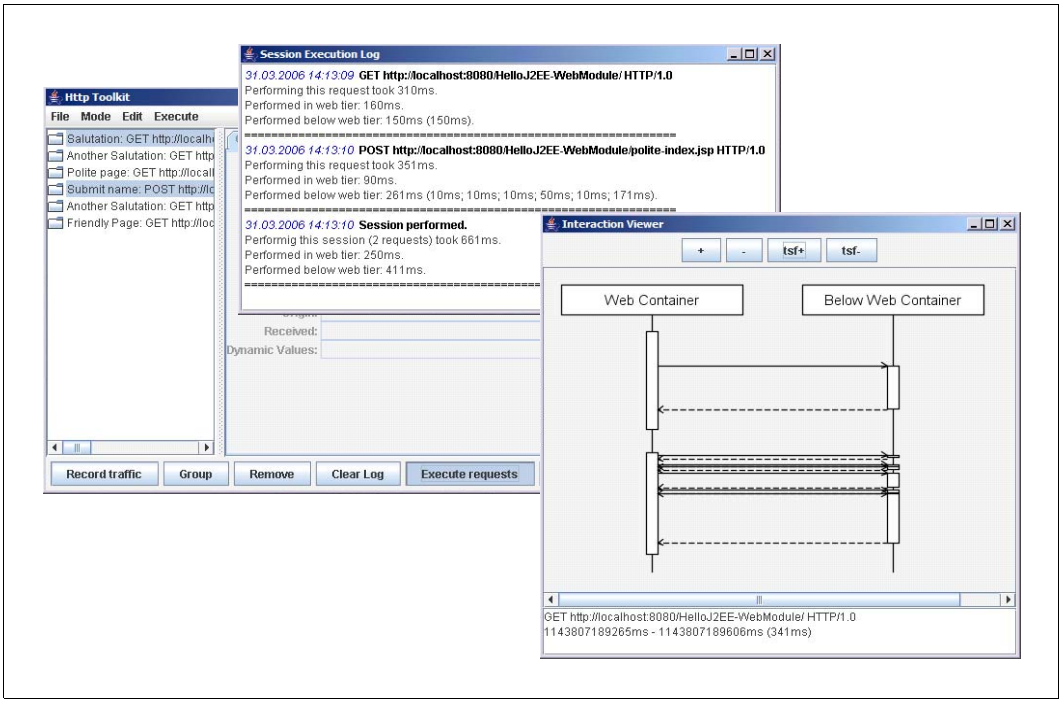

Abbildung 4.7: Der HTTP-Werkzeugkasten nach einer Prüfung

4.5.1 Die Implementierung einer Web-Filter-Komponente

Die Web-Filter-Komponente ist in der prototypischen Implementierung als HTTP-Proxy-Server umgesetzt worden. Da die Implementierung eines TCP/IP-Servers auch für den Laufzeitinformationsdienst notwendig war, ist ein kompaktes Server-Rahmenwerk entwickelt worden, das die Umsetzung von TCP/IP-Serverkomponenten vereinheitlicht und vereinfacht. Der HTTP-Proxy-Server basiert auf einer im Rahmen dieser Arbeit entwickelten Komponente, die als universeller HTTP-Proxy eingesetzt werden kann. Über die Registrierung verschiedener Filter kann die Funktionalität dieser Proxy-Komponente angepasst werden. Die Verarbeitungskomponenten des HTTP-Proxy-Servers realisieren Methoden zum Empfangen und Senden beliebiger HTTP-Nachrichten. Die Datenhaltungskomponenten ermöglichen durch die Nutzung des Java-Beans-Konzepts die Persistierung aufgezeichneter HTTP-Anfragefolgen. Das Server-Rahmenwerk sowie die Umsetzung der einzelnen Komponenten des HTTP-Proxy-Servers werden im Folgenden jeweils erläutert.

Das Server-Rahmenwerk

Das Server-Rahmenwerk ist eine Sammlung von Java-Klassen, die helfen kann, die Umsetzung von TCP/IP-Servern einheitlich zu gestalten und zu vereinfachen. Die Kernkomponente des Server-Rahmenwerks ist die Klasse `ServiceThread`. Ihre Aufgabe besteht darin, in einem selbständigen Thread auf eingehende Verbindungen an einem im Konstruktor übergebenen Server-Socket-Objekt zu warten. Sobald eine eingehende Verbindung hergestellt wurde, wird diese an ein Verarbeitungsobjekt übergeben, das dafür verantwortlich ist, eingehende Anfragen Protokoll-gemäß zu verarbeiten. Die Klassen dieser Verarbeitungsobjekte beerben und implementieren die abstrakte Klasse `ConnectionHandler`. Sie werden von einem Fabrik-Objekt, einer `ConnectionHandlerFactory`, bezogen. Die Nutzung des Server-Rahmenwerks hat den Vorteil, dass immer wiederkehrende Aufgaben der TCP/IP-Server-Konstruktion bereits erfüllt sind und so im Wesentlichen nur das gewünschte Protokoll umgesetzt werden muss. Tatsächlich beschränkt sich die Entwicklung eines TCP/IP-Servers mit dem Server-Rahmenwerk auf die Implementierung der beiden abstrakten Klassen `ConnectionHandler` und `ConnectionHandlerFactory`.

```
1:  public interface HttpFilter {
2:     public static enum Action {
3:           SEND, IGNORE
4:     }
5:
6:     public Action filterRequest(RequestFilterEvent evt);
7:
8:     public Action filterResponse(ResponseFilterEvent evt);
9:  }
```

Quelltext 4.1: Die HTTP-Filter-Schnittstelle

Der HTTP-Proxy-Server

Der HTTP-Proxy-Server des HTTP-Werkzeugkastens stellt die eigentliche Web-Filter-Komponente dar. Er basiert auf der Klasse `FilteringHttpProxy`, die es mittels des Beobachtermusters ermöglicht, HTTP-Anfragen und -Antworten während der Kommunikation zwischen einem Webbrowser und der Web-Schnittstelle einer Server-basieren Anwendung aufzuzeichnen. Zu diesem Zweck können an einem Filtering-HTTP-Proxy-Objekt Filterobjekte registriert werden, deren Klassen die Schnittstelle `HttpFilter` implementieren. Deren Methoden `filterRequest` beziehungsweise `filterResponse` (Quelltext 4.1, Zeilen 6 und 8) werden dann bei den entsprechenden Ereignissen vom Filtering-HTTP-Proxy aufgerufen.

Die Verarbeitungskomponenten des HTTP-Proxys

Die wesentliche Leistung des HTTP-Proxys besteht in der Fähigkeit, HTTP-Anfragen eines Browsers zu empfangen, diese an den richtigen Webserver zu leiten und die entsprechenden HTTP-Antworten zu empfangen, um diese wiederum an den Browser zu übergeben. Prinzipiell bietet die Java-Laufzeitumgebung zwar bereits einige Mechanismen, um HTTP-Anfragen an einen Webserver zu senden und entsprechende HTTP-Antworten zu empfangen, allerdings beruhen diese Mechanismen auf Klassen (neben anderen `java.net.HttpURLConnection`), die für das Weiterleiten von Anfragen relativ ungeeignet sind, da sie im Wesentlichen für den Empfang von HTML-Dokumenten entworfen wurden. Darüber hinaus fehlt bislang ein Mechanismus zum Empfangen von HTTP-Anfragen.

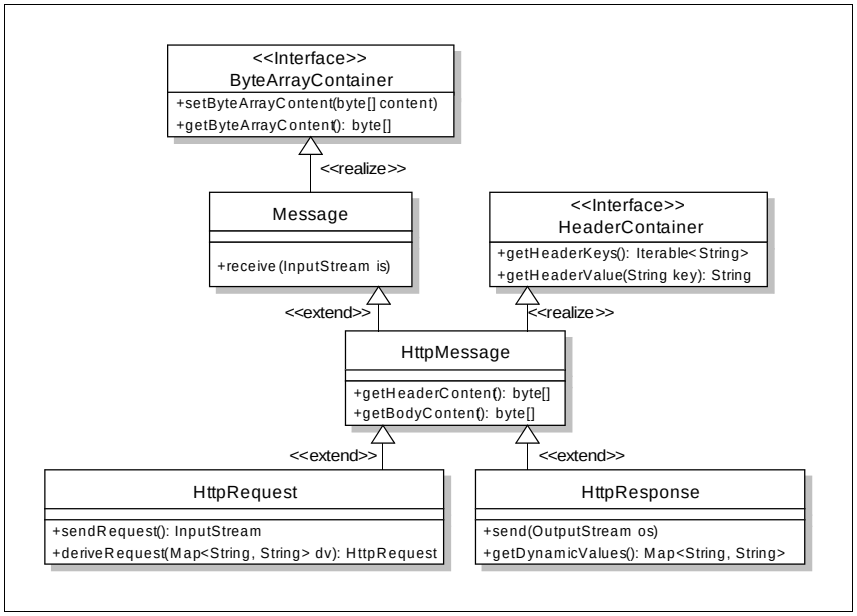

Abbildung 4.8: Klassenstruktur der HTTP-Verarbeitungskomponenten

Aus diesen Gründen wurden die notwendigen Verarbeitungskomponenten zum Senden und Empfangen von HTTP-Anfragen und -Antworten im Rahmen der prototypischen Implementierung geschaffen. Dies bringt den Vorteil mit sich, dass die Funktionalität sehr genau auf die Anforderungen zugeschnitten werden konnten. Abbildung 4.8 stellt die Klassenstruktur der HTTP-Verarbeitungskomponenten dar. Die große Tiefe der Vererbungshierarchie dient der Partitionierung der Funktionalität. Die wesentlichen Komponenten `HttpRequest` und `HttpResponse` sind in der Lage, HTTP-Anfragen und Antworten von beliebigen Eingabeströmen

(`java.io.InputStream`) zu empfangen und an beliebige Ausgabeströme (`java.io.OutputStream`) zu senden.

Die Datenhaltungskomponenten des HTTP-Proxys

Die Verarbeitungskomponenten des HTTP-Proxys stellen die Funktionalität zur Verfügung, die benötigt wird, um HTTP-Anfragen und -Antworten mit Webbrowsern und Webservern auszutauschen. Objekte der Klassen `HttpRequest` und `HttpResponse` enthalten nach dem Empfang die vollständigen HTTP-Nachrichten. Diese Inhalte könnten im Prinzip mit beliebigen Serialisierungsmechanismen persistiert werden. Dies entspräche allerdings nicht dem Anspruch einer sauberen Trennung zwischen Datenhaltung und Verarbeitung. Die Anforderungen an die prototypische Implementierung sehen außerdem die Möglichkeit vor, HTTP-Nachrichten zu benennen und in Gruppen zusammenzufassen. Aus diesen Gründen wurden Datenhaltungskomponenten entwickelt, die ausschließlich als strukturierte Datencontainer dienen.

```
1:  <java version="1.5.0_06" class="java.beans.XMLDecoder">
2:   <object class="de...io.HttpRequestBean">
3:    <void property="name">
4:     <string>GET http://localhost:8080/ HTTP/1.0</string>
5:    </void>
6:    ...
7:   </object>
8:  </java>
```

Quelltext 4.2: Auszug einer XML-serialisierten HTTP-Anfrage

Abbildung 4.9 zeigt die Klassenstruktur dieser Datenhaltungskomponenten. Bei der Entwicklung der Datenhaltungskomponenten wurden die Vorgaben der *JavaBeans* eingehalten. Dies ermöglicht eine einfache Serialisierung und Deserialisierung mittels der Klassen `XMLEncoder` und `XMLDecoder`. Diese Klassen werden von der Java-Laufzeitumgebung zur Verfügung gestellt.

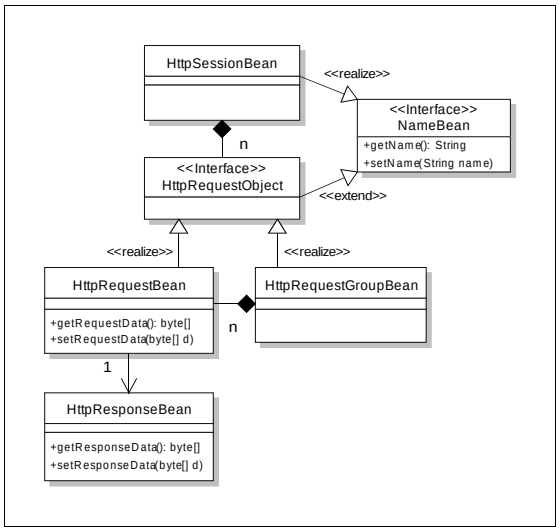

Abbildung 4.9: Klassenstruktur der Datenobjekte einer HTTP-Sitzung

Der Serialisierungsmechanismus erzeugt XML-Text aus den öffentlichen Eigenschaften von JavaBeans. Quelltextauszug 4.2 zeigt exemplarisch eine serialisierte HTTP-Anfrage. Die erzeugten XML-Dokumente hängen nicht so stark von der Implementierung ihrer entsprechenden Klassen ab, wie binäre Repräsentationen, die mittels des Java-Serialiserungsmechanismus erzeugt wurden. Dadurch ist es möglich, einmal erzeugte XML-Repräsentationen auch nach kleineren Änderungen an den Klassen, die sie repräsentieren, zu verwenden. Selbst eine Änderung der Java-Paketstruktur kann in den XML-Dokumenten durch ein einfaches Suchen und Ersetzen nachvollzogen werden.

4.5.2 Die Implementierung einer Web-Driver-Komponente

Die Web-Driver-Komponente des HTTP-Werkzeugkastens dient dem Senden vorab aufgezeichneter HTTP-Anfragemengen an die Web-Schnittstelle einer zu prüfenden Server-basierten Anwendung. Diese Anforderung erfüllt die im Rahmen der Web-Filter-Komponente entwickelte Klasse `HttpResponse` bereits. Allerdings war es erforderlich, dynamische Aspekte von HTTP-Anfragen zu behandeln. Bei diesen dynamischen Aspekten handelt es sich insbesondere um die bereits angesprochenen Sitzungsidentifikationsnummern. Die Werte von dynamischen Aspekten werden als Cookies oder URL-codierter Parameter übermittelt. Die Werte dynamischer Aspekte sind allerdings einerseits in HTTP-Anfragemengen aufgezeichnet, werden andererseits aber normalerweise bei jeder Sitzung vom Server neu bestimmt. Entsprechend führt die Web-Driver-Komponente Buch über vom Server empfangene Werte dynamischer Aspekte und pflegt diese an den entsprechenden Stellen in die HTTP-Anfragen ein, bevor die Anfragen an den Web-Server gesendet werden. Über die Kommunikation mit der Web-Schnittstelle der zu prüfenden Anwendung hinaus besteht die Aufgabe der Web-Driver-Komponente darin, Antwortzeiten zu messen und wenn möglich anteilige Antwortzeiten zu errechnen. Die Ermittlung anteiliger Antwortzeiten erfordert das Vorhandensein eines Laufzeitinformationsdienstes sowie eine zu prüfende Anwendung, die um Funktionalität angereichert wurde, die Laufzeitinformationen ermittelt. Die Generierung und Injizierung dieser Funktionalität ist die Aufgabe des J2EE-Injektors, der im nächsten Abschnitt vorgestellt wird.

4.6 Der J2EE-Injektor

Bei der prototypischen Implementierung dieser Arbeit besteht die Aufgabe des J2EE-Injektors darin, eine zu prüfende Anwendung um zusätzliche Funktionalität anzureichern, die es ermöglicht, Laufzeitinformationen bezüglich der EJB-Schicht dieser Anwendung zu ermitteln. Dieser Vorgang umfasst vier wesentliche Schritte. Zunächst werden die Komponenten einer zu prüfenden J2EE-Anwendung aus dem Anwendungsarchiv extrahiert und die EJB-Modularchive identifiziert. Darauf hin werden die relevanten Java-Klassen der EJB-Module ermittelt und extrahiert um aus den Informationen, die mit Hilfe der Java Reflection API ermittelt werden können, Filterkomponenten zu generieren. Nachdem die generierten Filterkomponenten kompiliert wurden, werden diese abschließend durch Manipulation des EJB-Deployment-Deskriptors in das EJB-Modul injiziert.

4.6.1 Extrahieren der Komponenten einer J2EE-Anwendung

Um die EJB-Module einer J2EE-Anwendung um zusätzliche Funktionalität anzureichern, ist es erforderlich, die entsprechenden Komponenten des Anwendungsarchivs zu extrahieren. Da J2EE-Anwendungsarchive auf dem JAR-Archivformat basieren, welches wiederum auf dem ZIP-

Dateiformat aufbaut, können die Einträge eines J2EE-Anwendungsarchivs mit den entsprechenden Mitteln extrahiert werden. Die Java-Laufzeitumgebung bietet Klassen an, die Funktionalität zur Behandlung von ZIP-Archiven (`java.util.zip.ZipFile`) und JAR-Archiven (`java.util.jar.JarFile`) zur Verfügung stellen. Allerdings ist es mit diesen Klassen in der zur Fertigstellung dieser Arbeit aktuellen Version von Java (1.5.0_06) nicht möglich, Archive aus Archiven zu extrahieren. Dies ist aber notwendig, da es sich bei den Anwendungsmodulen innerhalb eines J2EE-Anwendungsarchivs wiederum um spezialisierte JAR-Archive handelt. Aus diesem Grund wird die etwas umständlichere Methode über die Klasse `java.util.zip.ZipInputStream` gewählt, bei der die einzelnen ZIP-Einträge sequenziell ausgelesen werden müssen. Die extrahierten Komponenten einer J2EE-Anwendung werden bei der prototypischen Implementierung in ein temporäres Verzeichnis abgelegt, wo sie weiterverarbeitet werden können. Zur Lokalisierung der EJB-Module wird der Anwendungs-Deployment-Deskriptor der J2EE-Anwendung mit den Mitteln des JDOM-Rahmenwerks [JDO2006] eingelesen und bezüglich EJB-Modul-Einträgen untersucht (siehe Quelltextauszug 3.1 in Abschnitt 3.2.4 auf Seite 32).

4.6.2 Extrahieren der Komponenten eines EJB-Moduls

Nachdem die EJB-Module einer J2EE-Anwendung identifiziert und lokalisiert wurden, ist es erforderlich, wenigstens den EJB-Deployment-Deskriptor sowie die Klassen der Enterprise Java Beans aus den EJB-Modularchiven zu extrahieren. Da auch EJB-Module potenziell wieder Archive enthalten können, wurde auch hier der Mechanismus des sequentiellen Extrahierens aller ZIP-Einträge in ein temporäres Verzeichnis gewählt. Um die einzelnen Klassen der Enterprise Java Beans zu identifizieren, wird wiederum mittels des JDOM-Rahmenwerks der EJB-Deployment-Deskriptor eingelesen und bezüglich Angaben zu den Home- und Component-Schnittstellen sowie den implementierenden Klassen untersucht (siehe Quelltextauszug 3.2 in Abschnitt 3.2.4 auf Seite 33).

4.6.3 Automatische Generierung von EJB-Filter-Komponenten

Nachdem die Komponenten eines EJB-Modularchivs extrahiert wurden und die einzelnen Enterprise Java Beans identifiziert sowie ihre zugehörigen Component-Schnittstellen und implementierenden Klassen lokalisiert wurden, erfolgt für jede Enterprise-Java-Beans-Komponente die Generierung einer entsprechenden Filterkomponente in drei Schritten. Diese Filterkomponenten folgen dem Stellvertretermuster und geben alle an sie gerichteten Anfragen an die eigentliche Enterprise Java Beans weiter. Damit die Schnittstellen sowie die implementierende Klasse eines Enterprise Java Beans in der Java-Laufzeitumgebung genutzt werden kann, müssen im ersten Schritt die entsprechenden Java-Klassen bekannt gemacht werden. In der prototypischen Implementierung wird hierfür die Klasse `java.net.URLClassLoader` verwendet, die in der Lage ist, Java-Klassendateien zur Laufzeit zu laden. Der nächste Schritt besteht im Generieren einer Stellvertreter-Klasse, die alle Methoden des ursprünglichen Enterprise Java Beans zur Verfügung stellt. Quelltextauszug 4.3 zeigt exemplarisch einen Ausschnitt aus einer generierten Klasse `ComplexHelloBean_Proxy`, die Stellvertreter der Klasse `ComplexHelloBean` ist. Für die Generierung solcher Stellvertreter-Klassen wird die implementierende Klasse mit der Java Reflection API bezüglich ihrer Struktur untersucht, das heißt die beerbten Klassen und implementierten Schnittstellen sowie die öffentlichen Methoden werden ermittelt. Diese Struktur wird auf die entsprechende Stellvertreter-Klasse übertragen. Dabei wird jede Methode in der

Stellvertreter-Klasse derart implementiert, dass ein Methodenaufruf zunächst im Laufzeitinformationsdienst registriert wird (Quelltextauszug 4.3, Zeilen 16 und 17). Als nächstes wird die entsprechende Methode an einem Objekt der ursprünglichen Implementierung des Enterprise Java Beans aufgerufen (Quelltextauszug 4.3, Zeile 19). Nach Abarbeitung der Methode wird das Ende des Methodenaufrufs wiederum im Laufzeitinformationsdienst vermerkt (Quelltextauszug 4.3, Zeilen 22 bis 24). Eine besondere Herausforderung war in diesem Zusammenhang die Vorbereitung der Parameter und die Nachbereitung eventueller Rückgabewerte bei der Nutzung primitiver Datentypen. Da dieser Arbeit der J2EE-Standard in der Version 1.4 zugrunde liegt, darf bei der Generierung von Stellvertreter-Klassen keine Funktionalität von Java 5 verwendet werden, weshalb das Boxing und Unboxing von primitiven Datentypen entsprechend behandelt werden musste. Nachdem der Quelltext einer Stellvertreter-Klasse generiert ist, folgt im letzten Schritt die Kompilierung zu einer Java-Klasse. Im Rahmen dieser Arbeit wurden zu diesem Zweck verschiedene Java-Übersetzer untersucht. Der anfangs favorisierte *javac* von Sun stellte sich als unzweckmäßig heraus, da von der Nutzung aus der Java-Laufzeitumgebung heraus abgeraten wird. Letztlich fiel die Wahl auf den Java-Übersetzer der Eclipse Foundation der im Rahmen der Open-Source-Entwicklungsumgebung Eclipse entwickelt wird. Der Eclipse Compiler lässt sich problemlos aus einem Java-Programm heraus aufrufen und ist in der Lage, Java-Klassen für bestimmte Java-Versionen zu erzeugen.

```
1:  public class ComplexHelloBean_Proxy implements SessionBean,
2:                                      ComplexHelloRemoteBusiness,
3:                                      ComplexHelloLocalBusiness {
4:
5:    private ComplexHelloBean proxiedBean;
6:
7:    public ComplexHelloBean_Proxy() {
8:      proxiedBean = new session.ComplexHelloBean();
9:      SessionBeanEventManager.broadcastSessionBeanCreatedEvent
10:         (proxiedBean, new Object[] {});
11:   }
12:
13:   public void setName(java.lang.String arg0) {
14:     String iid = SessionBeanEventManager.generateInvocationID();
15:     Object[] parameters = new Object[] {arg0};
16:     SessionBeanEventManager.broadcastMethodInvocationStartingEvent
17:          (proxiedBean, "setName", parameters, iid);
18:     long startTime = System.currentTimeMillis();
19:     proxiedBean.setName(arg0);
20:     long finishTime = System.currentTimeMillis();
21:     Object returnValue = null;
22:     SessionBeanEventManager.broadcastMethodInvocationFinishedEvent
23:          (proxiedBean, "setName", parameters, iid, returnValue,
24:           startTime, finishTime);
25:   }
26:   ...
27: }
```

Quelltext 4.3: Ausschnitt einer generierten Stellvertreter-Klasse

4.6.4 Injizieren von EJB-Filter-Komponenten in ein EJB-Modul

Sind alle Stellvertreter-Klassen generiert und kompiliert, müssen diese wieder in einem EJB-Modularchiv abgelegt werden. Darüber hinaus ist es erforderlich, den EJB-Deployment-Deskriptor so umzuschreiben, dass die Einträge, die auf die implementierenden Klassen der Enterprise Java Beans verweisen, die Stellvertreter-Klassen adressieren. Nachdem alle Komponenten wieder zu Modularchiven und die Modularchive wieder zu einem J2EE-Anwendungsarchiv zusammengefasst wurden, werden abschließend die erzeugten temporären Verzeichnisse entfernt. Das generierte J2EE-Anwendungsarchiv kann jetzt in einem J2EE-Anwendungsserver installiert werden.

4.7 Der Laufzeitinformationsdienst

Die Aufgabe des Laufzeitinformationsdienstes besteht darin, den Austausch von Laufzeitinformationen zwischen Prüfkomponenten zu ermöglichen. Der Laufzeitinformationsdienst ist hierfür notwendig, da eine direkte Kommunikation zwischen Prüfkomponenten im Allgemeinen nicht möglich ist. Dies liegt daran, dass Komponenten, die in J2EE-Containern ausgeführt werden bestimmten Sicherheitseinschränkungen unterliegen. Unter anderem ist es nicht möglich, aus einem J2EE-Container heraus einen TPC/IP-Serverdienst zu starten. Gegebenenfalls könnte man zwar die Sicherheitseinstellungen des J2EE-Containers anpassen, um diese Einschränkung aufzuheben, hierfür ist aber durch den J2EE-Standard kein Vorgehen festgelegt und solch eine Anpassung kann abhängig vom verwendetem J2EE-Server einen tiefen Eingriff in dessen Konfiguration erfordern. Darüber hinaus besteht prinzipiell die Möglichkeit, den Laufzeitinformationsdienst in eine Prüfkomponente zu integrieren, die außerhalb eines J2EE-Containers ausgeführt wird. In dem im Rahmen dieser Arbeit entwickelten Prototyp ist die Web-Driver-Komponente des HTTP-Werkzeugkastens solch eine J2EE-Container-unabhängige Komponente. Da aber grundsätzlich auch Prüfkomponenten, die jeweils in J2EE-Containern ausgeführt werden, in der Lage sein sollen, untereinander Laufzeitinformationen auszutauschen, wurde hier der allgemeinere Ansatz eines Laufzeitinformationsdienstes als eigenständige Komponente verfolgt.

4.7.1 Arbeitsweise des Laufzeitinformationsdienstes

Der Laufzeitinformationsdienst wurde im Rahmen des für diese Arbeit entwickelten Prototyps als einfacher TCP/IP-Dienst umgesetzt. Er verfolgt eine einfache ASCII-Zeilen-basierte Anfrage-Antwort-Strategie. Das bedeutet, dass der Laufzeitinformationsdienst nach dem Aufbau einer Verbindung durch eine Clientanwendung zunächst genau eine Zeile ASCII-Text erwartet, die eine gültige Anfrage darstellt. Je nach der Art der Anfrage reagiert der Laufzeitinformationsdienst entweder mit einer Antwort, die aus einer oder mehreren Zeilen ASCII-Text bestehen kann, oder er beendet die Verbindung, falls keine Antwort erforderlich ist. Sieht das Protokoll das Senden einer Antwort auf eine Anfrage vor, so erwartet der Laufzeitinformationsdienst, dass die Clientanwendung nach vollständigem Erhalt der Antwortnachricht die Verbindung beendet. In jedem Fall wird also pro Verbindung nur eine Anfrage verarbeitet.

4.7.2 Protokoll des Laufzeitinformationsdienstes

Das Protokoll des Laufzeitinformationsdienstes ist erweiterbar entwickelt worden. Für die prototypische Implementierung im Rahmen dieser Arbeit ist nur die Behandlung der Laufzeitinformationen von EJB-Komponenten sowie eine rudimentären Sitzungsverwaltung notwendig. Insgesamt sind dafür sechs Protokollbefehle erforderlich, die im Folgenden jeweils kurz erläutert werden. In den einzelnen Quelltextauszügen werden die ausgetauschten Nachrichten dargestellt. Jede Zeile beginnt hierbei mit einem Pfeil. Eine Nachricht, die von einer Clientanwendung an den Laufzeitinformationsdienst gesendet wird (eine Anfrage), beginnt mit einem „=>“, eine Nachricht, die vom Laufzeitinformationsdienst an eine Clientanwendung gesendet wird (eine Antwort), beginnt mit einem „<=“. Außerdem werden Platzhalter verwendet, die die Form *„«name»“* haben. Darüber hinaus stellen *„*EMPTY*“* eine leere Zeile und *„*CLOSE*“* das Beenden einer Verbindung dar. Falls eine Nachricht nicht genau einmal erwartet wird, wird die mögliche Anzahl durch *„[0..n]“* angegeben. Alternative Nachrichten werden durch ein „|“ getrennt dargestellt.

```
1:  => ejb-put-period: «start»; «finish»; «action»
2:  <= *CLOSE*
```

Quelltext 4.4: Ablegen von EJB-Laufzeitinformationen

Laufzeitinformationen von EJB-Komponenten bestehen aus der Anfangszeit, der Fertigstellungszeit und einer textuellen Darstellung einer erbrachten Leistung. Die Behandlung der Laufzeitinformationen von EJB-Komponenten erfordert die Möglichkeit einerseits Informationen zu Laufzeit von EJB-Leistungen abzulegen und andererseits abzufragen, welche abgelegten Laufzeitinformationen innerhalb eines bestimmten Zeitintervalls liegen. Das Protokoll des Laufzeitinformationsdienstes sieht vor, dass Anfragen, die die Behandlung von EJB-Komponenten betreffen, mit dem Präfix „`ejb-`“ beginnen. Wie in Quelltextauszug 4.4 definiert, dient die Anfrage „`ejb-put-period`“ mit der Angabe der Anfangszeit, der Fertigstellungszeit und einer textuellen Darstellung einer Leistung zum Ablegen der entsprechenden Laufzeitinformation. Mit der in Quelltextauszug 4.5 angegebenen Anfrage „`ejb-request-period`“ mit der nachfolgenden Angabe eines Zeitintervalls werden alle dem Laufzeitinformationsdienst bekannten Laufzeitinformationen abgerufen, die in dem entsprechenden Zeitintervall liegen.

```
1:  => ejb-request-period: «beginafter»; «finishbefore»
2:  <= start; finish; action
3:  <= «start»; «finish»; «action» [0..n]
4:  <= *EMPTY*
5:  => *CLOSE*
```

Quelltext 4.5: Abfragen von EJB-Laufzeitinformationen

Neben der Behandlung von Laufzeitinformationen von EJB-Komponenten erfordert die im Rahmen dieser Arbeit entstandene prototypische Implementierung eine rudimentäre Sitzungsverwaltung. Der Einsatz von Sitzungen dient im Rahmen dieser Arbeit der zeitlichen Synchronisation verteilter Prüfkomponenten. Der hier vorgestellte Prototyp ermöglicht den Einsatz des HTTP-Werkzeugkastens als Web-Driver-Komponente und einer injizierten J2EE-Anwendung auf unterschiedlichen Rechnern. Da die Uhren auf unterschiedlichen Rechnern im Normalfall nicht

exakt auf dieselbe Uhrzeit eingestellt sind, ist es ohne weitere Mechanismen nicht möglich, einer HTTP-Anfrage EJB-Laufzeitinformationen zuzuordnen.

```
1:  => session-start
2:  <= «session-id»
3:  => *CLOSE*
```

Quelltext 4.6: Starten einer Sitzung

Die Sitzungsverwaltung des Laufzeitinformationsdienstes ermöglicht das Starten und Beenden einer Sitzung, sowie die Ermittlung des Zeitintervalls, in dem die Sitzung aktiv war. Vor dem Beginn einer HTTP-Anfrage startet die Web-Driver-Komponente eine Sitzung, nach dem vollständigen Empfang der zugehörigen HTTP-Antwort beendet die Web-Driver-Komponente diese Sitzung und ruft deren Laufzeitinformation ab. Anhand dieses Zeitintervalls können anschließend die im zeitlichen Rahmen der HTTP-Anfrage liegenden EJB-Laufzeitinformationen angefordert werden.

```
1:  => session-finish: «session-id»
2:  <= *CLOSE*
```

Quelltext 4.7: Beenden einer Sitzung

Das Protokoll des Laufzeitinformationsdienstes sieht vor, dass Anfragen, die die Sitzungsverwaltung betreffen, mit dem Präfix „`session-`“ beginnen. Eine Sitzung wird mittels der Anfrage „`session-start`“ gestartet. Auf diese Anfrage erhält die Client-Anwendung eine Sitzungsidentifikationsnummer, die für die weitere Verwendung der Sitzung genutzt wird. Dieser Ablauf ist in Quelltextauszug 4.6 spezifiziert.

```
1:  => session-request-period: «session-id»
2:  <= «start»; «finish»; «session-id» | *EMPTY*
3:  => *CLOSE*
```

Quelltext 4.8: Laufzeitinformationen einer Sitzung abrufen

Die Anfrage „`session-finish`“ mit der Angabe einer Sitzungsidentifikationsnummer beendet wie in Quelltextauszug 4.7 angegeben die entsprechende Sitzung im Laufzeitinformationsdienst. Ausschließlich zu beendeten Sitzungen können Laufzeitinformationen abgerufen werden. Wie in Quelltextauszug 4.8 dargestellt werden mit der Anfrage „`session-request-period`“ und der Angabe einer Sitzungsidentifikationsnummer Anfangs- und Endzeitpunkt der entsprechenden Sitzung angefordert. Existiert die Sitzung nicht oder wurde sie noch nicht beendet, sendet der Laufzeitinformationsdienst eine leere Zeile. Wird eine Sitzung nicht länger benötigt, sollte sie freigegeben werden, um auch den entsprechenden Speicher im Laufzeitinformationsdienst freizugeben. Dies geschieht wie in Quelltextauszug 4.9 angegeben durch die Anfrage „`session-destroy`“ gemeinsam mit der entsprechenden Sitzungsidentifikationsnummer.

```
1:  => session-destroy: «session-id»
2:  <= *CLOSE*
```

Quelltext 4.9: Freigeben einer Sitzung

4.7.3 Implementierung des Laufzeitinformationsdienstes

Der Laufzeitinformationsdienst wurde mit Hilfe des vorgestellten Server-Rahmenwerks entwickelt. Daher beschränkte sich die Umsetzung im Wesentlichen auf die Implementierung der abstrakten Klassen `ConnectionHandler` und `ConnectionHandlerFactory`. Beide wurden als anonyme Klassen innerhalb der Klasse `RuntimeInformationService` umgesetzt. Der Laufzeitinformationsdienst arbeitet in der prototypischen Implementierung auf dem TCP/IP-Port 18080. Zur einfacheren Client-seitigen Handhabbarkeit wurde zu dem Laufzeitinformationsdienst die Werkzeugklasse `RISClientTools` entwickelt, deren statische Methoden die verschiedenen Leistungen des Laufzeitinformationsdienstes aufrufen (siehe Quelltext 4.10).

```
1:  startRISSession(String host): String
2:  finishRISSession(String host, String sessionId)
3:  destroyRISSession(String host, String sessionId)
4:  receiveRISSessionPeriod(String host, String sId): Period
5:  receiveEJBRuntimeInformation(String host, Period p): Period[]
```

Quelltext 4.10: Die statischen Methoden der Klasse `RISClientTools`

4.8 Fazit

Das vorangegangene Kapitel hat die Besonderheiten der prototypischen Implementierung einiger wichtiger, in dieser Arbeit entwickelter Konzepte vorgestellt. Die prototypische Implementierung erlaubt die isolierte Betrachtung der Web-Schicht einer J2EE-Anwendung durch die Nutzung entsprechender Driver- und Filter-Komponenten, die es ermöglichen, anteilige Antwortzeiten zu ermitteln. Damit wurde die Umsetzbarkeit und praktische Nutzbarkeit der Konzepte des Web-Filters, des Web-Drivers und des EJB-Filters verifiziert. Darüber hinaus wurde für eventuell nachfolgende Arbeiten ein Zugang zu der prototypischen Implementierung geschaffen. Der Quellcode der hier vorgestellten Implementierung kann über den Arbeitsbereich Verteilte Systeme und Informationssysteme der Fakultät für Mathematik, Informatik und Naturwissenschaften an der Universität Hamburg bezogen werden.

5 Zusammenfassung und Ausblick

Dieses Kapitel schließt die vorliegende Diplomarbeit ab, indem die gewonnenen Erkenntnisse zusammengefasst und bewertet werden. Darüber hinaus werden Schlüsse gezogen, im Rahmen der Arbeit aufgeworfene oder offen gebliebene Fragestellungen genannt und in einem Ausblick kurz angerissen, welche weiterführenden Aufgabenstellungen sich aus dieser Arbeit ergeben könnten.

5.1 Zusammenfassung und Bewertung der Ergebnisse

Ausgangspunkt dieser Arbeit war die Fragestellung, ob es möglich ist, Server-basierte Anwendungen mit Mehrschichtarchitektur bezüglich ihrer Gesamtleistung und der Leistungsfähigkeit ihrer einzelnen Anwendungsschichten zu bewerten, ohne Anwendungsquelltexte zu verwenden. Hierfür wurden zunächst allgemeine Konzepte der Leistungsbewertung formal gefasst und die beiden wesentlichen Plattformen für die Entwicklung mehrschichtiger Server-basierter Anwendungen, Microsofts .NET-Rahmenwerk und die Java 2 Enterprise Edition von Sun, vorgestellt. Auf den formalen Grundlagen aufbauend wurden Konzepte für typische Server-basierte J2EE-Anwendungen entwickelt. In diesem Zusammenhang entstand der Begriff der Prüfkomponenten, der im Kontext dieser Arbeit Komponenten zum Kapseln von Anwendungsschichten meint. Das Konzept der Prüfkomponenten umfasst anfragende, filternde und ersetzende Komponenten. Es wurden jeweils Konzepte für Prüfkomponenten zu den typischen Anwendungsschichten von J2EE-Anwendungen entwickelt und bezüglich ihrer Relevanz im Rahmen dieser Arbeit bewertet. Abschließend wurde eine Auswahl dieser Prüfkomponenten prototypisch implementiert, um die isolierte Betrachtung der Web-Schicht einer J2EE-Anwendung zu ermöglichen. Mit dieser prototypischen Implementierung wurde gezeigt, dass es für J2EE-basierte Anwendungen möglich ist, einzelne Anwendungsschichten bezüglich ihrer Leistungsfähigkeit zu untersuchen, ohne auf Anwendungsquelltexte zurückzugreifen.

5.2 Schlussfolgerungen, Fragen und Ausblicke

In dieser Arbeit galt der Fokus insbesondere der Leistungsbewertung Server-basierter J2EE-Anwendungen. Insgesamt wurde ein in sich schlüssiger Satz von Konzepten entwickelt, der sich für die Umsetzung im J2EE-Umfeld eignet.

Die prototypische Implementierung dieser Konzepte setzt eine strikte Mehrschichtarchitektur der zu prüfenden Anwendungen voraus. Die Untersuchung einzelner Schichten einer Anwendung mit Mehrschichtarchitektur, die die Striktheit nicht einhält, erfordert die vollständige Kapselung jeder Anwendungsschicht. Das heißt, dass für jede Anwendungsschicht entsprechende Prüfkomponenten implementiert werden müssen. Die Möglichkeiten, die sich hieraus ergeben, könnten in einer nachfolgenden Arbeit untersucht werden.

Die Konzepte der verschiedenen Prüfkomponenten wurden für die J2EE-Version 1.4 entwickelt. Zwar werden aufgrund der großen Verbreitung auch in den kommenden Jahren sicherlich noch einige Anwendungssysteme für diese Version der J2EE-Plattform entwickelt und gepflegt, allerdings sind für die Folgeversion Java EE 5 viele Verbesserungen angekündigt, die insbesondere eine einfachere Entwicklung ermöglichen und Vorteile bezüglich der Performanz bringen sollen. Aus diesem Grund ist zu erwarten, dass dieser Nachfolger sich relativ schnell durchsetzen wird. Es wäre daher interessant zu untersuchen, wie die konzeptionellen Neuerungen in Java EE 5 sich auf

die hier entwickelten Konzepte der Leistungsbewertung auswirken. Insbesondere wird Java EE 5 einige Elemente der Aspekt-orientierten Programmierung unterstützen, was die Entwicklung von Filterkomponenten deutlich vereinfachen kann.

Obwohl die Konzepte dieser Arbeit mit besonderem Fokus auf die J2EE-Plattform entworfen wurden, scheint es aufgrund der konzeptionellen Ähnlichkeiten zwischen dem J2EE-Ansatz und der .NET-Plattform möglich zu sein, die Konzepte für die Umsetzung im .NET-Umfeld anzupassen. Eine genaue Untersuchung dieser Möglichkeit hätte den Rahmen dieser Diplomarbeit verlassen, wäre aber aufgrund der zunehmenden Beliebtheit von .NET sicherlich ein interessantes Folgeprojekt.

Im Rahmen dieser Arbeit wurde das Open-Source-Projekt JMeter vorgestellt, das ebenfalls der Leistungsbewertung von J2EE-Anwendungen dient. Allerdings bietet JMeter keine Möglichkeit, die innere Struktur eines Anwendungssystems zu untersuchen. Unter Umständen lassen sich die in dieser Arbeit entwickelten Konzepte der Leistungsbewertung sowie die prototypische Implementierung in das Rahmenwerk von JMeter integrieren. Auf diese Weise könnten die Ergebnisse dieser Arbeit einer breiteren Zielgruppe zugänglich gemacht werden.

Abschließend hat sich im Laufe dieser Diplomarbeit die Fragestellung ergeben, ob sich die Konzepte zur Leistungsbewertung auch für Funktionalitätsprüfungen von einzelnen Anwendungsschichten und Gesamtsystemen anpassen lassen. Solche Anpassungen könnten in Anlehnung an das Konzept der Unit-Tests umgesetzt werden. Auch hier würden gemäß der entwickelten Konzepte Anwendungsvorgänge durch Filterkomponenten aufgezeichnet werden. Eine Prüfung würde dann aber keine Leistungskennzahlen ermitteln, sondern prüfen, ob die Ergebnisse von angeforderten Leistungen den Erwartungen entsprechen. Der Vorteil dieses Ansatzes gegenüber klassischen Unit-Tests würde insbesondere darin liegen, dass Testanforderungen nicht mehr auf Entwicklerebene gestellt werden müssten, sondern auf Anwenderebene formuliert werden könnten.

Literaturverzeichnis

[And05] Andersen, L.: *JDBC™ 4.0 Specification, Public Draft v1.0.* Sun Microsystems, 2005

[Apa03] Apache Software Foundation: *BCEL Website.* Apache Software Foundation, 2003. http://jakarta.apache.org/bcel

[Apa05] Apache Software Foundation: *JMeter Website.* Apache Software Foundation, 2005. http://jakarta.apache.org/jmeter

[Ash04] Ashmore, Derek C.: *The J2EE Architect's Handbook.* DVT Press, 2004. ISBN 0-972-9548-99.

[BEA06] Bea Systems: *Bea WebLogic Server Website.* Bea Systems, 2006. http://www.bea.com

[CoYo03] Coward, D.; Yoshida, Y.: *Java™ Servlet Specification, Version 2.4*. Sun Microsystems, 2003

[DeGa96] Deutsch, P.; Gailly, J.-L.: *ZLIB Compressed Data Format Specification version 3.3*. Network Working Group, 1996. http://www.ietf.org/rfc/rfc1950.txt

[DeMi03] DeMichiel, L. G.: *Enterprise JavaBeans™ Specification, Version 2.1*. Sun Microsystems, 2003.

[Dor04] Van Dorst, V.: *BogoMips mini-Howto.* Wim van Dorst, 2004. http://www.clifton.nl/bogomips.html

[Dun94] Dunlaver, M. R.: *Building Better Applications – A Theory of Efficient Software Development.* International Thomson Publishing, 1994. ISBN 0-442-01740-50.

[EJT06] EJ-Technologies: *JProfiler Website.* EJ-Technologies, 2006. http://www.ej-technologies.com/products/jprofiler/overview.html

[EKT02] Engel, A.; Koschel, A.; Tritsch, R.: *J2EE kompakt – Enterprise Java: Konzepte und Umfeld.* Spektrum Akademischer Verlag, 2002. ISBN 3-8274-1381-8

[GHJV01] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2001. ISBN 3-8273-1862-9.

[Gre02] Gregory, K.: *Visual C++ .NET – Win32- und .NET-Anwendungen programmieren.* Martk+Technik, 2002. ISBN 3-8272-6459-6

[Ham97] Hamilton, G.: *JavaBeans™ Specification, Version 1.01-A*. Sun Microsystems, 1997.

[JDO06] The JDOM Project (Hrsg.): *JDOM Website*. The JDOM Project, 2006. http://www.jdom.org

[KBL05] Kifer, M.; Bernstein, A.; Lewis, P.: *Database Systems – An Application-Oriented Approach 2nd Edition.* Addison Wesley, 2005. ISBN 0-321-31256-2

[Lan04] Langner, T.: *Web-basierte Anwendungsentwicklung – Die wichtigsten Technologien für Webapplikationen im Überblick.* Spektrum Akademischer Verlag, 2004. ISBN 3-8274-1501-2

[Mic06] Microsoft Corporation (Hrsg.): *Microsoft .NET Pet Shop.* Microsoft Corporation (.NET framework community website), 2006. http://www.gotdotnet.com/team/compare/petshop.aspx

[PKW06] PKWARE Inc. (Hrsg.): *.ZIP File Format Specification*. PKWARE Inc., 2006. http://www.pkware.com/business_and_developers/developer/popups/appnote.txt

[SDN05] Sun Developer Network: *Java 2 Platform Standard Edition 5.0, API Documentation.* Sun Microsystems, 2005. http://java.sun.com/j2se/1.5.0/docs/api

[SDN06] Sun Developer Network: *Java Naming and Directory Interface (JNDI)*. Sun Microsystems, 2006. http://java.sun.com/products/jndi

[Sea02] Searls, R.: *JavaTM 2 Enterprise Edition Deployment API Specification, Version 1.1*. Sun Microsystems, 2002. http://java.sun.com/j2ee/tools/deployment/reference/docs/index.html

[Sha03] Shannon, B.: *Java 2 Platform Enterprise Edition Specification, v1.4*. Sun Microsystems, 2003. http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf

[Sha03a] Shannon, B.: *Java 2 Platfom, Enterprise Edition (J2EE): XML Schemas for J2EE Deployment Descriptors*. Sun Microsystems, 2003. http://java.sun.com/xml/ns/j2ee/index.html

[SJPC02] Sun J2EE Patterns Catalog: *Model-View-Controller.* Sun Microsystems, 2002. http://java.sun.com/blueprints/patterns/MVC.html

[Smi01] Smith, W.: *TCP Benchmarking An Ecommerce Solution, Revision 2.* Intel Corporation, 2001. http://tpc.org/tpcw/TPC-W_Wh.pdf

[Som98] Sommerer, A.: *The Java Archive (JAR) File Format*. Sun Microsystems, 1998. http://java.sun.com/developer/Books/javaprogramming/JAR

[SPEC06] SPEC (Hrsg.): *SPEC's Benchmarks and Published Results.* Standard Performance Corporation, 2006. http://www.spec.org/benchmarks.html

[Sun03] Sun Microsystems (Hrsg.): *J2EE™ Connector Architecture Specification*. Sun Microsystems,

2003. http://java.sun.com/j2ee/connector/download.html

[Sun06] Sun Microsystems (Hrsg.): *JDBC Technology.* Sun Microsystems, 2006. http://java.sun.com/products/jdbc

[Sun06a] Sun Microsystems (Hrsg.): *Enterprise Java Beans v2.1 API Documentation*. Sun Microsystems, 2006. http://java.sun.com/products/ejb/javadoc-2_1-fr/

[TaSt03] Tanenbaum, A.; van Steen, M.: *Verteilte Systeme – Grundlagen und Paradigmen.* Pearson Studium, 2003. ISBN 3-8273-7057-4

[ThLa03] Thai, T.; Lam, H.: *.NET Framework Essentials, 3rd Edition.* O'Reilly, 2003. ISBN 0-596-00505-9

[ZAO02] Zadrozny, P.; Aston, P.; Osborne, T.: *J2EE Performance Testing with BEA WebLogic Server.* Expert Press, 2002. ISBN 1-904-28400-00

[Zus99] Zuse, H.: *Der Rechner Z1.* Technische Universität Berlin, 1999. http://irb.cs.tu-berlin.de/~zuse/Konrad_Zuse/de/Rechner_Z1.html

Abbildungsverzeichnis

Quelltextverzeichnis

Tabellenverzeichnis

Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Außerdem erkläre ich, dass ich mit der Einstellung dieser Diplomarbeit in den Bestand der Bibliotheken der Universität Hamburg einverstanden bin.

Hamburg, den 20. April 2006

Alexander Kune