

Institut für Architektur von Anwendungssystemen  
Universität Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Studienarbeit Nr. 2166

# **„Pipes and Filter“ - Architektur und Workflow-basierte Systeme: ein praktischer Vergleich**

Oliver Eckhardt

<b>Studiengang:</b>	Informatik
<b>Prüfer:</b>	Prof. Dr. Frank Leymann
<b>Betreuer:</b>	Dipl.-Inf. Thorsten Scheibler Dipl.-Inf. Tobias Unger
<b>begonnen am:</b>	01. Mai 2008
<b>beendet am:</b>	31. Oktober 2008
<b>CR-Klassifikation:</b>	H.4.1



# Inhaltsverzeichnis

---

<b>1. Einleitung</b>	<b>7</b>
<b>2. Architekturen</b>	<b>8</b>
2.1. Workflowbasierter Ansatz . . . . .	8
2.1.1. Workflows und Workflow-Management-Systeme . . . . .	8
2.1.2. Vor- und Nachteile des Workflow-Ansatzes . . . . .	11
2.2. Pipes and Filter . . . . .	12
2.2.1. Vor- und Nachteile des Pipes-and-Filter-Ansatzes . . . . .	13
2.3. Zentrale Unterschiede der beiden Ansätze . . . . .	14
2.4. Umsetzung mittels IBM Websphere 6.1 . . . . .	15
2.4.1. Service Component Architecture . . . . .	16
2.4.2. Websphere Process Server . . . . .	17
2.4.3. Websphere Enterprise Service Bus . . . . .	19
2.4.4. Eignung der Websphere Produkte als Implementierung der Architekturen	20
<b>3. Testszenarios</b>	<b>23</b>
3.1. Das Erweiterte Loan Broker Szenario . . . . .	23
3.2. Patternentwurf . . . . .	24
3.3. Pipes and Filter - Implementierung . . . . .	27
3.4. Workflow - Implementierung . . . . .	34
<b>4. Rahmenbedingungen der Tests</b>	<b>41</b>
4.1. Testumgebung . . . . .	41
4.2. Messwerkzeuge . . . . .	42
4.3. Testfälle und Testmethodik . . . . .	44
<b>5. Testergebnisse</b>	<b>46</b>
5.1. Minimumtestfall . . . . .	46
5.1.1. Überblick über die Ergebnisse . . . . .	46
5.2. Mediumtestfall . . . . .	51
5.2.1. Überblick über die Ergebnisse . . . . .	51
5.3. Maximumtestfall . . . . .	56
5.3.1. Überblick über die Ergebnisse . . . . .	56

5.4.	Threadtestfälle . . . . .	61
5.4.1.	Überblick über die Ergebnisse . . . . .	61
5.5.	Serviceaufruftestfälle . . . . .	69
5.5.1.	Überblick über die Ergebnisse . . . . .	69
5.6.	Payloadtestfälle . . . . .	77
5.6.1.	Überblick über die Ergebnisse . . . . .	77
<b>6.</b>	<b>Gesamtbewertung</b>	<b>85</b>
6.1.	Zusammenfassung der Testergebnisse . . . . .	85
6.2.	Schlußfolgerungen bezüglich der Architektureigenschaften . . . . .	87
<b>7.</b>	<b>Zusammenfassung und Fazit</b>	<b>89</b>
<b>A.</b>	<b>Anhang</b>	<b>91</b>
A.1.	Konfigurationsänderungen bei den Wepshere-Produkten . . . . .	91
	<b>Literaturverzeichnis</b>	<b>94</b>

# Abbildungsverzeichnis

---

2.1. Workflow Reference Architecture [Coa] . . . . .	9
2.2. Bestandteile der SCA entnommen aus [KMC <sup>+</sup> 06] . . . . .	17
2.3. Websphere Process Server Components entnommen aus [KBC <sup>+</sup> 07] . . . . .	18
3.1. Patternentwurf . . . . .	26
3.2. Mediationflow . . . . .	29
3.3. Beginn des MediationFlow . . . . .	30
3.4. MediationFlow - Rating Agent . . . . .	30
3.5. MediationFlow - Förderungen . . . . .	31
3.6. MediationFlow - Banken . . . . .	32
3.7. MediationFlow - Risikoanalyse . . . . .	32
3.8. PayloadScanner und Ende des Mediationflows . . . . .	33
3.9. MicroFlow . . . . .	35
3.10. MicroFlow - Rating Agent . . . . .	36
3.11. MicroFlow - Förderungen . . . . .	36
3.12. MicroFlow - Banken . . . . .	37
3.13. MicroFlow - Risikoanalyse . . . . .	39
3.14. PayloadScanner und Ende des Microflows . . . . .	40
5.1. Minimumtest - Antwortzeiten aller Prozesstypen . . . . .	48
5.2. Minimumtest - Antwortzeiten Micro- und Mediationflow . . . . .	48
5.3. Minimumtest - Abweichungen aller Prozesstypen . . . . .	49
5.4. Mediumtest - Antwortzeiten aller Prozesstypen . . . . .	52
5.5. Mediumtest - Antwortzeiten Micro- und Mediationflow . . . . .	53
5.6. Mediumtest - Antwortzeiten aller Prozesstypen . . . . .	54
5.7. Mediumtest - Abweichungen aller Prozesstypen . . . . .	54
5.8. Maximumtest - Antwortzeiten aller Prozesstypen . . . . .	57
5.9. Maximumtest - Antwortzeiten Micro- und Mediationflow . . . . .	58
5.10. Maximumtest - Antwortzeiten Microflow . . . . .	59
5.11. Maximumtest - Antwortzeiten Mediationflow . . . . .	59
5.12. Maximumtest - Abweichungen aller Prozesstypen . . . . .	60
5.13. Threadtest - Antwortzeiten Microflow . . . . .	63
5.14. Threadtest - Antwortzeiten Microflow . . . . .	64
5.15. Threadtest - Antwortzeiten Mediationflow . . . . .	64
5.16. Threadtest - Antwortzeiten Mediationflow . . . . .	65
5.17. Threadtest - Antwortzeiten Macroflow . . . . .	65
5.18. Threadtest - Antwortzeiten Macroflow . . . . .	66

5.19. Threadtest - Abweichungen Microflow . . . . .	67
5.20. Threadtest - Abweichungen Mediationflow . . . . .	67
5.21. Threadtest - Abweichungen Macroflow . . . . .	68
5.22. Serviceaufruftest - Antwortzeiten Microflow . . . . .	72
5.23. Serviceaufruftest - Antwortzeiten Microflow . . . . .	72
5.24. Serviceaufruftest - Antwortzeiten Mediationflow . . . . .	73
5.25. Serviceaufruftest - Antwortzeiten Mediationflow . . . . .	73
5.26. Serviceaufruftest - Antwortzeiten Macroflow . . . . .	74
5.27. Serviceaufruftest - Antwortzeiten Macroflow . . . . .	74
5.28. Serviceaufruftest - Abweichungen Microflow . . . . .	75
5.29. Serviceaufruftest - Abweichungen Mediationflow . . . . .	75
5.30. Serviceaufruftest - Abweichungen Macroflow . . . . .	76
5.31. Payloadtest - Antwortzeiten Microflow . . . . .	79
5.32. Payloadtest - Antwortzeiten Microflow . . . . .	80
5.33. Payloadtest - Antwortzeiten Mediationflow . . . . .	80
5.34. Payloadtest - Antwortzeiten Mediationflow . . . . .	81
5.35. Payloadtest - Antwortzeiten Macroflow . . . . .	82
5.36. Payloadtest - Antwortzeiten Macroflow . . . . .	82
5.37. Payloadtest - Abweichungen Microflow . . . . .	83
5.38. Payloadtest - Abweichungen Mediationflow . . . . .	83
5.39. Payloadtest - Abweichungen Macroflow . . . . .	84

# Einleitung

---

Strukturierte Geschäftsprozesse sind in den letzten Jahrzehnten für Unternehmen von solcher Bedeutung geworden, dass sie mittlerweile als "major asset" verstanden werden [LRoo]. Dies verstärkt sich durch die Globalisierung und die damit einhergehende, verstärkte Arbeitsteilung. Immer mehr Firmen lagern immer größere Teile ihrer Produktion aus und sehen ihre eigentliche Aufgabe nur noch darin, die Teile zu einem Gesamtprodukt zusammenzuführen. Solche Geschäftsprozesse in Unternehmen zu unterstützen war schon immer eine wichtige Aufgabe der IT. Die Anforderungen werden dabei immer größer, insbesondere Flexibilität wurde mit sich immer schneller ändernden Anforderungen immer wichtiger. Anwendungen, in die die Prozessabläufe fest integriert sind, können diesen Anforderungen nicht genügen. Daher bemüht man sich seit Mitte der 90er Jahre, die Logik der Geschäftsprozesse aus den Anwendungen herauszunehmen, wie das in den Jahren zuvor bei der Datenhaltung mittels der Datenbanken geschehen war [AH04]. Aus Geschäftsprozessen wurden Workflows, die wiederum in Workflow-Management-Systemen ausgeführt werden.

Der Workflow-basierte Ansatz ist jedoch nicht die einzige Möglichkeit Geschäftsprozesse zu unterstützen. Eine der größten Herausforderungen bei Geschäftsprozessen besteht darin, bereits vorhandene Anwendungen in den Prozessablauf einzubinden. Hierbei soll Software miteinander verbunden werden, die meist nicht dafür konzipiert wurde, und möglicherweise auf unterschiedlichen Plattformen betrieben wird. Dies wird auch als Enterprise Application Integration bezeichnet. Eines der wichtigsten Architekturmuster der EAI ist die Pipes-and-Filter-Architektur. Diese Architektur beruht auf sog. Filtern, welche die tatsächliche Datenverarbeitung durchführen, und auf Pipes, die die Verbindung zwischen diesen Filtern herstellen. Dies ist ein in vielfacher Hinsicht anderer Ansatz, als er beim klassischen Workflowsystem verfolgt wird.

Ziel dieser Arbeit ist es, einen Performancevergleich zweier konkreter Produkte durchzuführen, die jeweils eine Umsetzung einer der beiden Architekturen darstellen. Zu diesem Zweck werden im nächsten Kapitel zunächst einmal die Architekturen sowie die Produkte näher vorgestellt und miteinander verglichen. In Kapitel 3 wird ein Testszenario eingeführt, das für beide Produkte implementiert wurde und als Grundlage der Messungen dient, während Kapitel 4 die allgemeinen Messbedingungen behandelt. In Kapitel 5 werden die Messergebnisse präsentiert und erläutert. Die Zusammenfassung der Ergebnisse wird schließlich in Kapitel 6 vorgestellt.

# Architekturen

---

## 2.1. Workflowbasierter Ansatz

### 2.1.1. Workflows und Workflow-Management-Systeme

Die Workflow Management Coalition [WfMC], die sich mit dem Etablieren von Standards im Bereich der Workflow-Technologie beschäftigt, definiert einen Workflow folgendermaßen:

The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules. [Coa99]

Ein Workflow kann also als die Automatisierung eines Geschäftsprozesses gesehen werden, wobei zwischen einem Workflow und einem Geschäftsprozess Unterschiede zu beachten sind. Die Bestandteile eines Geschäftsprozesses können grundsätzlich in zwei Kategorien eingeteilt werden. Zum einen jene, die manuell, also von Menschen ausgeführt werden müssen, und zum anderen solche, die automatisiert werden können. Die Beschreibung eines Prozesses, ein Prozessmodell, umfasst "die Struktur eines Geschäftsprozesses in der realen Welt" [LRoo], beinhaltet also den Geschäftsprozess als Ganzes. Ein Workflowmodell, das die Vorlage für einen Workflow darstellt, umfasst hingegen nur den Teil des Prozessmodells, der von Computern unterstützt werden kann. Dadurch kann das Workflowmodell verschieden große Teile des Prozessmodells umfassen, bis hin zum kompletten Prozess. Aber auch die Bereiche, die menschliches Eingreifen erfordern, werden im Workflowmodell nicht außen vor gelassen, denn die manuelle Bearbeitung von Aktivitäten kann in vielen Fällen ebenfalls durch Computer unterstützt werden. Dazu gehört beispielsweise das Bereitstellen der benötigten Daten, das Aufrufen von Anwendungsprogrammen, die für die Bearbeitung notwendig sind, sowie das Überprüfen und Weiterleiten der erarbeiteten Ergebnisse. Damit ein Workflowmodell dies alles leisten kann, muss es so formal sein, dass eine automatische Ausführung des Modells auf dafür vorgesehenen Systemen, den Workflow-Management-Systemen (WfMS), möglich ist.

Ein solches System soll nicht nur den Ablauf der Prozesse ermöglichen, sondern auch alle "Phasen eines Prozess-Lifecycles, also das Workflow-Management, durch IT-Werkzeuge

unterstützen“ [Mülo5]. Dies umfasst “alle Aufgaben, die bei der Analyse, der Modellierung, der Simulation, der Reorganisation sowie bei der Ausführung und Steuerung von Workflows benötigt werden“ [Mülo5]. Dabei muss ein WfMS mit anderen Softwarekomponenten zusammenarbeiten, die prinzipiell von unterschiedlichen Herstellern kommen können. Dies setzt entsprechende Standards voraus. Die Workflow Management Coalition hat zu diesem Zwecke eine Referenzarchitektur herausgegeben, die die Struktur und die Schnittstellen eines WfMS beschreibt. (Abbildung 2.1) [Coa].

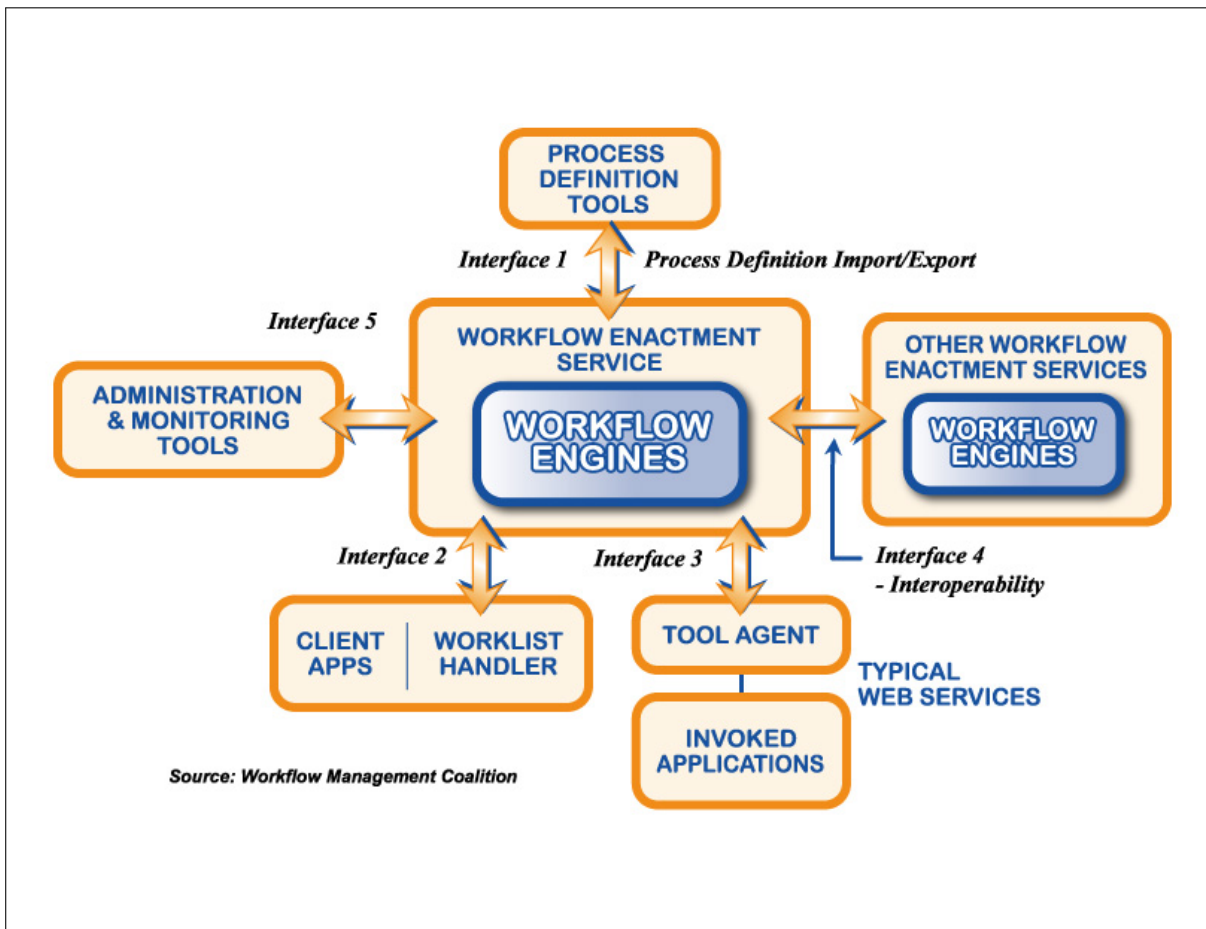


Abbildung 2.1.: Workflow Reference Architecture [Coa]

In deren Mittelpunkt steht der Workflow Enactment Service. Dieser Service besteht aus einer oder mehreren Workflow Engines und ermöglicht es, Instanzen der vorliegenden Prozessmodelle zu erzeugen und diese auszuführen. Das Interface 1 der Architektur soll die Schnittstelle zu Modellierungswerkzeugen (Process Definition Tools) ermöglichen, mit denen die Workflow-Modelle definiert werden. Über Interface 2 wird die Verbindung zu Worklist Handlern ermöglicht. Diese Worklist Handler sollen die Aufgaben entgegennehmen, die manuell, also von einem Benutzer ausgeführt werden müssen. Im einfachsten Fall zeigt

der Worklist Handler lediglich die Aufgaben an, die zu erledigen sind. Um den Benutzer zu unterstützen sind aber noch eine Vielzahl anderer Möglichkeiten denkbar, angefangen davon, die Aufgaben nach bestimmten Kriterien vorzusortieren bis hin zur Unterstützung bei benötigten Anwendungsprogrammen, wie oben bereits beschrieben. Interface 3 ist notwendig, damit die automatisch ablaufenden Aktivitäten des Workflow, die bestimmten Anwendungen zugeordnet sind, diese mittels standardisierter Schnittstellen aufrufen können. Das Interface 4 soll die Verbindung zu anderen Workflowsystemen schaffen, und so die Möglichkeit geben, dass dort vorliegende Prozessmodelle gestartet werden können oder mit dort laufenden Workflow-Instanzen kommuniziert werden kann. Interface 5 soll es Steuerungs- und Überwachungswerkzeugen, die nicht im System selbst integriert sind, ermöglichen, auf dieses zuzugreifen, es zu steuern und seine Daten auszuwerten [Hol95]. Ein Prozessmodell wird also zunächst über Interface 1 in das WfMS eingebracht. Wird der Prozess nun gestartet, wird in der Workflow Engine eine Instanz des Workflowmodells erzeugt. Entsprechend dem definierten Prozessmodells und der übergebenen Eingabedaten, werden nun die einzelnen Aktivitäten des Workflows durchlaufen. Je nach Art der Aktivität werden entweder interne Anweisungen durchgeführt, externe Anwendungen mittels Interface 3 aufgerufen oder Anweisungen an einen Worklist Handler durch Interface 2 gegeben. Ebenfalls kann über Interface 4 mit anderen Workflow-Management-Systemen kommuniziert werden. Gesteuert und überwacht kann der Ablauf einer solchen Instanz durch Anwendungen, die mit Hilfe von Interface 5 auf die WfMS zugreifen.

Prinzipiell ist es erwünscht, dass ein Geschäftsprozess als Transaktion abläuft. Ein Prozess soll entweder erfolgreich ausgeführt werden, oder es soll zu keinen Änderungen kommen. Transaktionen sind aber kurzlebig, während Workflows im Allgemeinen langlebig und auch unterbrechbar sein können. Außerdem ist es durch den Aufruf externer Anwendungen meist nicht möglich, Transaktionen im klassischen Sinne durchzuführen, da die Anwendungen dafür ebenfalls entsprechende Funktionen unterstützen müssten. Daher ist ein Workflow meist in mehrere Transaktionen unterteilt, und für den Fall eines Abbruchs des Prozesses muss auf Mittel wie Kompensation zurückgegriffen werden.

Ein Prozess kann also prinzipiell in zwei Kategorien eingeteilt werden, nämlich in lang-laufende Prozesse und in kurz-laufende Prozesse. Diese Einteilung ist nicht von der durchschnittlichen Laufzeit abhängig, sondern beschreibt eher das potentielle Verhalten. Die Einteilung wird in der Regel bei der Erstellung des Prozessmodells gemacht und legt fest, ob der Prozess als eine einzige Transaktion läuft (kurz-laufend) oder in mehrere Transaktionen unterteilt wird, deren einzelne Ergebnisse persistent gespeichert werden (lang-laufend). Ein solcher lang-laufender Prozess ist auch unterbrechbar, kann also an bestimmten Stellen angehalten und dann zu einem beliebigen Zeitpunkt später wieder aufgenommen werden. Dies ist besonders wichtig, wenn einzelne Aktivitäten des Prozesses von Menschen ausgeführt werden müssen, und sich dabei obere zeitliche Grenzen nur schwer angeben lassen.

### 2.1.2. Vor- und Nachteile des Workflow-Ansatzes

Die Vorteile eines Workflows liegen zu allererst einmal in der Trennung von Geschäftsfunktionen und Geschäftslogik. Das Workflowmodell stellt die Logik des Geschäftsprozesses dar, also dessen Ablauf. Innerhalb dieses Ablaufes werden dann die Geschäftsfunktionen, meist in Form von separaten Softwarekomponenten aufgerufen. Durch das Kapseln des Ablaufs in eigenständige Modelle wird die Prozesslogik aus der Anwendung herausgelöst. Dies schafft eine neue Form der Flexibilität. Die Erstellung von Geschäftsfunktionen und Workflowmodellen kann jeweils eigenen Spezialisten übertragen werden. Das Workflowmodell selbst kann unabhängig vom WfMS und den darauf laufenden Instanzen modelliert und getestet werden. Die erstellten Modelle können portiert und damit auf mehreren Systemen gleichzeitig eingesetzt werden und sind im Idealfall auch nicht an ein bestimmtes WfMS gebunden. Viele Änderungen im Geschäftsprozess betreffen nur den Ablauf, also die Geschäftslogik, aber nicht die verwendeten Funktionen. Genau solche Änderungen können mittels eines WfMS leicht umgesetzt und sogar so deployed werden, dass die Änderungen nur für Instanzen gelten, die ab einem bestimmten Zeitpunkt erstellt werden, während alle älteren Instanzen nach dem alten Modell ausgeführt werden.

Um die Komplexität von Geschäftsprozessen bewältigen zu können, bieten die zur Modellierung eingesetzten Workflowsprachen diverse Möglichkeiten den Kontrollfluss zu steuern. Angefangen von einfachen, bedingten Verzweigungen bis hin zur Angabe komplexer Bedingungen, um das Vereinigen paralleler Pfade zu beschreiben, stehen konsistente Metamodelle zur Verfügung um Geschäftsprozesse geeignet modellieren zu können. Abgesehen von der Parallelität, die im modellierten Prozess selbst enthalten ist, ermöglicht das WfMS den parallelen Ablauf mehrerer Instanzen desselben Workflowmodells und kümmert sich so um die Skalierbarkeit der Geschäftsprozesse.

Da das WfMS als zentrale Komponente die Workflows steuert, werden sämtliche Informationen über die Instanzen zentral im WfMS gespeichert und können dort abgefragt werden. Damit können zunächst einmal die einzelnen Workflows selbst überwacht, ihr Status abgefragt oder Funktionen, wie eine Deadline für laufende Instanzen, ermöglicht werden. Der genaue Ablauf jedes Workflows kann als Historie abgespeichert werden und zu Dokumentationszwecken dienen. Dies kann wegen gesetzlicher Anforderungen wichtig sein, oder die Analyse der durchgeführten Geschäftsprozesse ermöglichen. Eigenschaften, wie die Laufzeit eines Prozesses, eventuelle Engpässe oder häufige Problempunkte, können aus solchen Analysen ermittelt werden.

Die Nachteile von Workflows ergeben sich zumeist ebenfalls aus dem WfMS als zentraler Komponente des Gesamtsystems. Zunächst ist damit natürlich ein Single Point of Failure geschaffen, besonders wenn auf dem System verschiedene Workflowmodelle deployed sind, deren Prozesse ohne das WfMS alle ausfallen, obwohl sie sonst nichts miteinander zu tun haben. Da sämtliche Aufrufe der Workflows über das WfMS laufen müssen, kann es natürlich auch zu einem Engpass werden, der wiederum alle Workflows, die auf dem System laufen, beeinträchtigen kann.

Ein anderes Problem liegt in mangelnden Standards. Wie beschrieben muss ein WfMS um seine Vorteile nutzen zu können mit einer Vielzahl von Anwendungen zusammenarbeiten,

was nur über Standards möglich ist. Diese Standards liegen aber häufig nicht vor oder werden zu wenig unterstützt. Die Workflowmodelle beispielsweise lassen sich oft nicht auf den WfMS verschiedener Hersteller nutzen, da selbst bei der Verwendung standardisierter Sprachen noch häufig proprietäre Erweiterungen in den Modellen verwendet werden.

### 2.2. Pipes and Filter

Pipes and Filter (PaF) ist ein Architekturansatz, der in der Enterprise Application Integration (EAI) vielfach Verwendung findet. EAI wird hierbei als Funktion verstanden, "die in einem Geschäftsprozess eingebundenen Applikationen in einer heterogenen IT-Umgebung zu verwenden und somit einen Datenaustausch zwischen den Applikationen ohne einen Medienbruch zu erreichen"[Mül05]. Auch hier geht es also um Geschäftsprozesse, doch steht weniger der Prozess im Vordergrund als vielmehr die technische Möglichkeit, die beteiligten Komponenten miteinander kommunizieren zu lassen. Die Grundidee ist ein Gesamtsystem, das aus einzelnen Komponenten, sog. Filtern, besteht, die wiederum über Kommunikationskanäle miteinander verbunden sind, den Pipes.

Die Filter sind jeweils für eine bestimmte Teilaufgabe der Gesamtfunktion des Systems verantwortlich. Diese Aufgaben bestehen meist in einer Form von Datentransformation. Das bedeutet, ein Filter ergänzt, verfeinert oder transformiert seine Inputdaten und gibt die Outputdaten weiter. Der Begriff Filter hat hierbei historische Gründe und soll nicht bedeuten, dass lediglich Filteroperationen stattfinden. Untereinander sollen die Filter möglichst lose gekoppelt sein. Das bedeutet, dass keiner der Filter seinen Vorgänger oder seinen Nachfolger kennen soll, und die Filter auch keinen gemeinsamen Zustand teilen dürfen.

Die Pipes haben prinzipiell nur die Aufgabe die Outputdaten eines Filters als Inputdaten an den nachfolgenden Filter zu liefern. Im klassischen Fall besitzt jeder Filter ein Input- und ein Output-Interface, sog. Ports, über die er mit den Pipes verbunden ist, die die Verbindung zu seinem Vorgänger bzw. zu seinem Nachfolger herstellen. In der Praxis kann eine Komponente aber durchaus mehrere Input- und Output-Ports aufweisen. Im Idealfall sind die Ports aller Filter identisch, so dass sich prinzipiell jeder Filter mit jedem durch eine Pipe verbinden lässt, allerdings werden hier in der Praxis meist Abstriche gemacht. Die dadurch notwendige Formatkonvertierung muss dann entweder in einem der beteiligten Filter erfolgen, oder es wird zu diesem Zweck ein separater Filter dazwischengeschaltet. Unter Umständen können auch die Pipes die Aufgabe der Formatkonvertierung übernehmen [Cop95], womit sie sich dann aber nur noch schwer gegen Filter abgrenzen lassen. Die Pipes basieren häufig auf asynchronem Messaging, womit sie das Zwischenspeichern der Daten übernehmen. Es sind aber auch synchrone Implementierungen, beispielsweise auf Basis von Procedure-Calls, denkbar. Damit werden jedoch die Eigenschaften der Losen Kopplung abgeschwächt, womit wichtige Vorteile verloren gehen.

Die Anordnung der Filter erfolgt in der Regel sequentiell oder parallel und ändert sich nicht dynamisch [Cop95]. Gerichtete Graphen sind zwar möglich, können aber Probleme aufwerfen, beispielsweise bei Schleifen [Bus00]. Durch die Abgeschlossenheit der Filter und unterstützt durch möglichst identische Ports, ist es relativ einfach möglich in das Gesamtsys-

tem neue Filter zu integrieren, bestehende zu entfernen, oder die Abfolge der Filter beliebig zu verändern.

### 2.2.1. Vor- und Nachteile des Pipes-and-Filter-Ansatzes

Die Vorteile einer PaF-Architektur ergeben sich zunächst einmal aus ihrer losen Kopplung. Sie ist sehr flexibel, da die Filter quasi beliebig angeordnet, ergänzt oder entfernt werden können. Da die Filter eine in sich abgeschlossene Funktionalität bieten, können sie separat getestet werden und sind auch wiederverwendbar, können also ohne große Anpassungen in anderen Systemen eingesetzt werden. Außerdem sind die PaF-Systeme vergleichsweise robust. Ein Fehler in einem Filter beeinflusst zunächst einmal nicht die anderen Filter. Im Idealfall kann das System bis zu dem fehlerhaften Filter ordnungsgemäß funktionieren, und die Inputnachrichten, die auf Grund des Fehlers nicht verarbeitet werden, bleiben einfach in der Input-Pipe der fehlerhaften Komponente gespeichert. Funktioniert die Komponente wieder normal, können die aufgelaufenen Inputnachrichten abgearbeitet werden. Recoverymechanismen müssen nur innerhalb der jeweiligen Filter implementiert sein, um ein Forward-Recovery des gesamten Systems zu ermöglichen. Denn alle Daten, die sich nicht in einem Filter befinden sind persistent in den Pipes gespeichert und ein globaler Zustand, der ggf. wiederhergestellt werden müsste, existiert nicht.

Für die Verbesserung des Durchsatzes bietet die PaF-Architektur einige Möglichkeiten. Zunächst einmal kann jeder Filter mit seiner eigenen Geschwindigkeit arbeiten. Da er unabhängig von den anderen Filtern ist, muss er nicht auf seinen Nachfolger warten, sondern kann seine Outputdaten in die Output-Pipe stellen, sobald er fertig ist. Prinzipiell muss er sich aber natürlich nach der Geschwindigkeit seines Vorgängers richten, da er auf dessen Outputdaten angewiesen ist. Wenn einzelne Filter einen Engpass darstellen, kann dies jedoch innerhalb der PaF-Architektur effizient durch Parallelität ausgeglichen werden. In diesem Fall kann der jeweilige Filter einfach mehrfach in das System integriert werden, und die Daten dann auf diese Filter aufgeteilt werden. Durch die Eigenschaften der losen Kopplung ist ein Hinzufügen solcher Filter bei Bedarf relativ einfach möglich. Bei einem Engpass muss also nicht das gesamte System mehrfach zur Verfügung stehen, sondern lediglich die Filter, die den Engpass darstellen, müssen für eine parallele Verarbeitung mehrfach in das System eingefügt werden. Für den Durchsatz kann außerdem ein Pipeline-Effekt ausgenutzt werden. Dieser beruht darauf, dass jede Komponente mit neuen Daten weiterarbeiten kann, sobald sie ihre letzten Daten an die nächste Komponente weitergegeben hat. Dadurch ist das Gesamtsystem nicht nur mit einem Datenpaket beschäftigt und muss warten bis dieses das gesamte System durchlaufen hat. Stattdessen können prinzipiell alle Filter parallel an ihren eigenen Datenpaketen arbeiten, und damit als Optimum so viele Datenpakete vom Gesamtsystem gleichzeitig bearbeitet werden, wie Filter vorhanden sind.

An Nachteilen muss zunächst einmal der Ressourcenbedarf genannt werden, da für jede Pipe entsprechende Ressourcen allokiert werden müssen, und die Anzahl der Pipes sehr groß werden kann. Hinzu kommt ein gewisser Aufwand für Formatkonvertierung. Wenn die Komponenten nicht sehr gut aufeinander abgestimmt sind, müssen die jeweiligen internen Formate stets am Ende konvertiert werden, damit der nächste Filter die Daten

weiterverarbeiten kann. Ja nachdem, wie weit man die lose Kopplung treiben möchte, werden die Daten dabei auf ein im Gesamtsystem einheitliches Format gebracht, das jeder Filter akzeptiert, oder jeweils nur auf das Format des aktuell nachfolgenden Filters konvertiert. Die erste Methode ermöglicht ein problemloses Neuordnen der Filter, jedoch lässt sich solch ein globales Format oft nur schwer definieren. Vermeidet man diese Schwierigkeit, erhält man ein enger gekoppeltes System und muss bei einer Abänderung der Filterfolge jedes Mal auch die Formatkonvertierung anpassen.

Ein Nachteil, der sich direkt aus der losen Kopplung ergibt, ist das Fehlen von globalen Informationen. Das Nutzen gemeinsamer Daten über einzelne Filter hinweg ist bisweilen sinnvoll, aber in einer PaF-Architektur nur schwer umzusetzen. Auch gibt es keine zentrale Stelle, an der Informationen über das Gesamtsystem verfügbar sind. Sollte so etwas erforderlich sein, muss es erst separat implementiert werden, was durch die lose Kopplung entsprechend erschwert wird.

### 2.3. Zentrale Unterschiede der beiden Ansätze

WfMS und die PaF-Architektur ähneln sich in vielen Dingen. Mit beiden können Geschäftsprozesse technisch unterstützt und den Anforderungen von EAI begegnet werden. Beide sind auf Komponenten angewiesen, die in sich abgeschlossen sind und unabhängig von anderen Komponenten ihre Funktionalität anbieten. WfMS sind dabei deutlich stärker auf die Umsetzung von Geschäftsprozessen ausgelegt, während bei PaF-Systemen die Integration heterogener Systeme, egal zu welchem Zweck, im Vordergrund steht.

Die ersten Unterschiede ergeben sich bei der Erstellung lauffähiger Prozesse. Beim WfMS findet dies unabhängig vom eigentlichen System mittels spezieller Modellierungswerkzeuge statt, und hat ein Workflowmodell als Ergebnis. Bei einem PaF-System werden im klassischen Fall die Pipes entsprechend angelegt bzw. der Zugriff der Filter auf die Pipes konfiguriert, was nicht in einem Modell sondern in einem fertigen System resultiert. Hier wird also auf unterschiedlichen Abstraktionsebenen gearbeitet.

Beim Start eines Prozesses unterscheiden sich die beiden Ansätze ebenfalls. Das WfMS erstellt in diesem Fall eine Instanz des entsprechenden Workflow-Modells, das dann in einer separaten Workflow-Engine innerhalb der WfMS abläuft. In einem System, das auf der PaF-Architektur beruht, wird im klassischen Fall die entsprechende Aufrufnachricht einfach an die erste Komponente übergeben, die diese Nachricht dann verarbeitet und weiterleitet. Im Gesamtsystem befinden sich dann mehrere Datenobjekte einzelner Prozesse, die nur soweit von einander abgetrennt sind, als dass sie auf unterschiedlichen Komponenten verarbeitet werden. Das kann zu Problemen führen, wenn sich bspw. der Datenfluss an einer Stelle teilt und in einem späteren Filter wieder zusammengeführt wird. Dabei können Datenobjekte unterschiedlicher Prozesse bei dem Sammelfilter eintreffen und es muss gewährleistet sein, dass diese dann richtig einander zugeordnet werden können. Bei einem WfMS kann dies nicht passieren, da die Prozesse separat in ihrer eigenen Workflowinstanz ablaufen.

Die Strukturen, die beide Ansätze abbilden können, sind ebenfalls sehr ähnlich. Workflowsprachen bieten auf den ersten Blick mehr Möglichkeiten komplexe Abläufe zu beschreiben, sind aber auch auf einer höheren Abstraktionsebene als ein PaF-System. Die einzelnen

Sprachelemente werden ja erst vom WfMS ausgeführt. In einer PaF-Architektur können die jeweiligen Funktionen stattdessen in entsprechende Filter implementiert werden, die auch die Freiheit bieten, ganz neue Funktionen für spezielle Anforderungen zu erstellen.

Ein wichtiger Unterschied ist, dass bei einer PaF-Architektur die Struktur im Regelfall vom System selbst vorgegeben ist, nämlich wie die einzelnen Filter durch die Pipes miteinander verbunden sind. Daher kann ein System, das auf solch einer PaF-Architektur beruht auch immer nur jeweils ein Prozessmodell unterstützen. Beim WfMS hingegen wird die Struktur durch das Workflowmodell vorgegeben, und ein WfMS ist in der Lage jedes gültige Workflowmodell auszuführen, auch parallel. Damit kann sich prinzipiell auch der Ablauf eines Workflows dynamisch, also zur Laufzeit, ändern, was in einer PaF-Architektur grundsätzlich so nicht möglich ist.

Ein zentraler Unterschied zwischen beiden Ansätzen stellt die zentrale Komponente des WfMS im Gegensatz zu den vielen, prinzipiell gleichberechtigten Komponenten einer PaF-Architektur dar. Während das WfMS den kompletten Prozess steuert und kontrolliert, gibt es in einem PaF-System keine zentrale Steuerung, da der Ablauf von den Pipes vorgegeben wird. Hier gibt es natürlich gewisse Einschränkungen, abhängig von der Implementierung der PaF-Architektur. Wird bspw. eine Message-orientierte Middleware für die Pipes verwendet, kann diese auch als zentrale Komponente betrachtet werden, die für die Steuerung verantwortlich ist. Aber rein von der Architektur her ist so etwas nicht vorgesehen. Auf die Probleme einer solchen zentralen Komponente wurde bei den Nachteilen des WfMS bereits hingewiesen.

Aus dem Unterschied der zentralen Komponente im Gegensatz zu verteilten Komponenten folgt einer der wesentlichsten Unterschiede der beiden Ansätze: die verschiedene Verfügbarkeit von Informationen im System. In einem WfMS ist der globale Zustand eines jeden Workflows jederzeit verfügbar, bei einem PaF-System ist dieser Zustand prinzipiell nicht zugänglich. Bei letzterem ist das auch schwieriger, da die einzelnen Prozesse nicht in unterschiedlichen Instanzen getrennt ablaufen, sondern auf die einzelnen Komponenten des Gesamtsystems verteilt sein können. Das alles erschwert in einem PaF-System auch das Überwachen der ablaufenden Prozesse sowie das Erfassen der Ablaufdaten für Historien. Entsprechend sind darauf basierende Funktionen und Analysen, wie sie für das WfMS oben beschrieben wurden, schwierig. In einem PaF-System ist auch, anders als in einem WfMS, der Datenfluss nicht unabhängig vom Kontrollfluss. In einem Workflow können die Daten, die von einer Applikation erzeugt werden, direkt in den Input-Container der Anwendung kopiert werden, die diese Daten als nächste benötigt, auch wenn sie erst einige Aktivitäten später im Prozess aufgerufen wird. Da bei einem PaF-System die Filter keine Zustände teilen, müssen solche Daten stets von Filter zu Filter übergeben werden, bis der Filter erreicht ist, der diese Daten benötigt.

## 2.4. Umsetzung mittels IBM Websphere 6.1

Zur Implementierung der beiden Ansätze wurden der IBM Websphere Process Server 6.1.2 sowie der darin enthaltene IBM Websphere Enterprise Service Bus verwendet, die in diesem Kapitel näher vorgestellt werden sollen. Beide Produkte sind Bestandteile der Referenzarchi-

tektur der IBM SOA-Foundation. Diese Architektur besteht aus einer integrierten Menge von Software-Produkten und soll ein Unternehmen zur Umsetzung einer Service-Orientierten-Architektur befähigen [KBC<sup>+</sup>07].

Der Websphere Process Server gehört dabei zu den Kern-Komponenten der Architektur. Er liefert die Voraussetzungen um den Ablauf und die Interaktion zwischen verschiedenen Services zu ermöglichen, die einen Geschäftsprozess implementieren. Der Websphere Enterprise Service Bus wird den unterstützenden Komponenten der Architektur zugeordnet. Er stellt die IBM-Version einer (Teil-)Umsetzung eines Enterprise Service Bus dar, und ist damit eine Middleware-Komponente. Beide Produkte werden in den Kapiteln 2.4.2 auf der nächsten Seite bzw. 2.4.3 auf Seite 19 näher vorgestellt. Weitere Details finden sich in den Kapiteln zur Implementierung des Testszenarios (Kapitel 3.3 auf Seite 27 und 3.4 auf Seite 34)

### 2.4.1. Service Component Architecture

Die Bestandteile der Referenz-Architektur der SOA-Foundation sollen ein integriertes Gesamtpaket darstellen. Grundlage dafür ist, dass Teile dieser Architektur wiederum auf einer gemeinsamen Architektur basieren, der Service Component Architecture (SCA). Während der Begriff SOA ein abstraktes Konzept ist, um Services und ihre Beziehungen untereinander zu beschreiben, stellt die SCA eine Implementierung einer SOA dar. Sie bietet ein universelles Modell, um Business Services, die Geschäftsdaten veröffentlichen oder verarbeiten, zu implementieren. Zu diesem Zweck werden Geschäftslogik und Implementierungsdetails voneinander getrennt [KMC<sup>+</sup>06]. Die SCA definiert hierzu Module und Komponenten, die über standardisierte Schnittstellen miteinander kommunizieren und Daten in Form von Business Objects austauschen (s. Abbildung 2.2 auf der nächsten Seite) [KGG<sup>+</sup>07].

Kernelemente der SCA sind die sog. **Komponenten**, in sich abgeschlossene, wiederverwendbare Einheiten. Für die Implementierung dieser Komponenten gibt es diverse Möglichkeiten, die unter anderem Java-Klassen, EJBs oder auch BPEL-Prozesse (Business Process Execution Language [BPE]) umfassen können. Diese Komponenten bieten ihre Funktionalität über Schnittstellen, sog. Interfaces, an und können selbst auch andere Komponenten über sog. Referenzen aufrufen, die dann wiederum die Interfaces der aufzurufenden Komponenten verwenden. Das Interface einer Komponente umfasst hierbei die Operationen, die aufgerufen werden können, und die verwendeten Daten, also Parametertypen, Rückgabetypen und Ausnahmen. Zur Beschreibung der Interfaces wird im wesentlichen WSDL [WSD01] verwendet [KGG<sup>+</sup>07].

Komponenten können in einem **Modul** kombiniert werden, wodurch das Modul selbst eine Geschäftsfunktion ausführt bzw. unterstützt. Ein Modul besteht also aus einer oder mehrerer Komponenten und kann direkt deployed werden. Mittels sog. Imports kann ein Modul die Funktionalität anderer Module bzw. deren Komponenten nutzen. Ebenso können damit auch Nicht-SCA-Komponenten genutzt werden, sofern sie geeignete Schnittstellen zur Verfügung stellen. Seine eigenen Funktionen stellt eine SCA-Komponente über sog. Exports zur Verfügung.

Grundsätzlich gibt es zwei Arten von Modulen, die Business Service Module und die Mediation Module. Ein Business Service Modul implementiert die Logik eines Geschäftsprozesses und enthält normalerweise eine BusinessFlow-Komponente, die einen BPEL-Prozess darstellt. Ein Mediation-Modul hingegen umfasst typischerweise eine MediationFlow-Komponente, deren Aufgabe es ist, die Kommunikation zwischen Anwendungen durch Routing und Transformation der übermittelten Daten zu ermöglichen.

Um die Integration innerhalb und zwischen den verschiedenen Produkten zu ermöglichen wird als gemeinsames Modell für Geschäftsdaten innerhalb der Architektur der offene Standard der Service Data Objects (SDO) verwendet. Diese bieten eine Datenstruktur, die aus einfachen Daten, anderen SDOs oder beidem aufgebaut sein kann. Ebenfalls enthält ein SDO Referenzen zu Metadaten, die über die jeweils gespeicherten Daten Auskunft geben. Diese SDOs sind wiederum die Grundlage für die **Business Objects**, die das eigentliche Datenmodell der SCA darstellen [KMC<sup>+</sup>06].

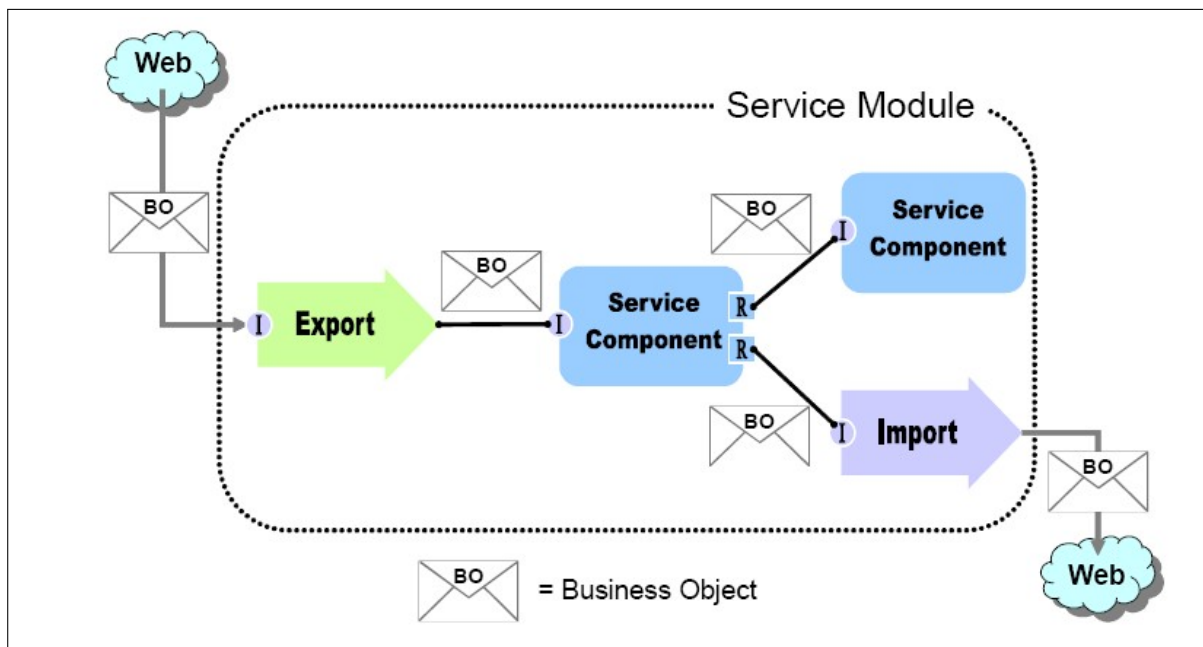


Abbildung 2.2.: Bestandteile der SCA entnommen aus [KMC<sup>+</sup>06]

### 2.4.2. Websphere Process Server

Der Websphere Process Server (WPS) stellt nun ein Workflow-Management-System dar, das auf der Service-Component-Architektur basiert. Seine Architektur besteht aus drei Ebenen, dem SOA-Core, den Supporting Services und den Service Components [KMC<sup>+</sup>06] (s. Abbildung 2.3 auf der nächsten Seite)

Der SOA-Core besteht aus der beschriebenen Service Component Architecture sowie den

## 2. Architekturen

Business Objects als Grundlage des SOA-Gesamtkonzeptes von IBM. Die Common Event Infrastructure bietet ein grundlegendes Ereignis-Management für die Überwachung von Programmabläufen. Sie wird im Kapitel 4.2 auf Seite 42 näher behandelt. Durch die Verwendung der SCA können die Vorteile eines einheitlichen Aufrufmodells und eines einheitlichen Datenmodells genutzt werden. Die Supporting Services, zu denen in Form von Mediation Flows auch der Websphere Enterprise Service Bus beiträgt (s. 2.4.3 auf der nächsten Seite), stellen die notwendige Funktionalität zur Verfügung, um Prozesse unterschiedlicher Anbieter miteinander verbinden zu können [KBC<sup>+</sup>07]. Die Service Components stellen nun die eigentlichen Kernelemente des Process Servers dar. Sie bieten die Funktion Workflowmodelle (Business Processes) zu erstellen und menschliche Eingriffe in den Geschäftsprozess zu ermöglichen (Human Tasks). Die Business State Machines und Business Rules sind weitere Hilfsmittel um Geschäftsprozesse zu automatisieren, jedoch in dieser Studienarbeit nicht relevant. All diese Komponenten sind Komponenten im Sinne der SCA. Sie werden also jeweils durch ein Interface beschrieben und können innerhalb eines Moduls miteinander verbunden werden oder über Imports des Moduls auf andere Module zugreifen. Da die Interfaces unabhängig von der konkreten Implementierung sind, also ob eine Komponente beispielsweise ein Business Process oder eine State Machine ist, kann die Implementierung einer Komponente jederzeit ausgetauscht werden, ohne das die Gesamtstruktur innerhalb des Moduls geändert werden muss. Die wichtigste Komponente zur Automatisierung von Geschäftsprozessen stellt die Business Process - Komponente dar. Sie bietet eine WS-BPEL-konforme Laufzeitumgebung für Workflows, und unterstützt dabei lang- und kurzlaufende Prozesse.

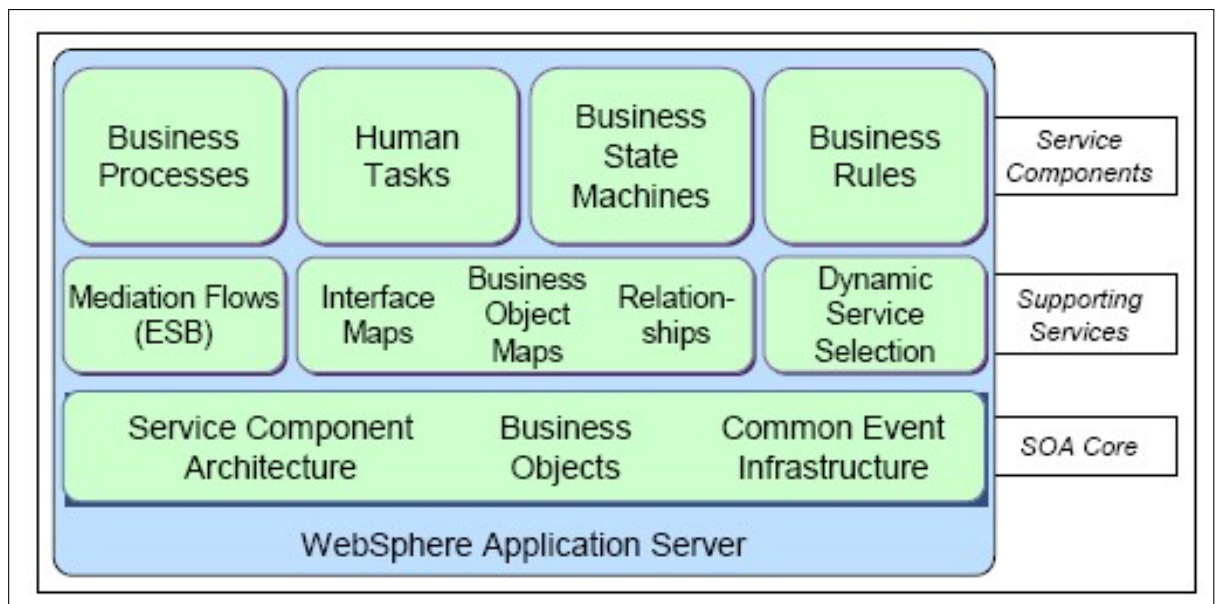


Abbildung 2.3.: Websphere Process Server Components entnommen aus [KBC<sup>+</sup>07]

### Business Processes

Workflows werden im WebSphere Process Server als Business Processes bezeichnet. Grundlage dieser Business Processes ist die Business Process Execution Language BPEL als verwendete Workflowsprache. Ein Business Process kann entweder lang- oder kurzlaufend sein. Kurzlaufende Prozesse sind nur für automatische Aktivitäten gedacht, die innerhalb kurzer Zeit ablaufen. Die einzelnen Zwischenschritte des Workflows werden nicht persistent gespeichert, können im Fehlerfall also nicht wiederhergestellt werden. Der komplette Prozess läuft als eine Transaktion ab, und beansprucht entsprechend weniger Ressourcen im Vergleich zu einem langlaufenden Prozess. Verzweigungen innerhalb des Workflowmodells werden nicht parallel ausgeführt, da der gesamte Workflow in einem einzelnen Thread abläuft.

Ein langlaufender Prozess kann sich über mehrere Stunden bis hin zu Tagen oder Monaten erstrecken. Er besteht im Regelfall aus mehreren, einzelnen Transaktionen und das transaktionale Verhalten der beteiligten Aktivitäten kann entsprechend konfiguriert werden. Die Ergebnisse der einzelnen Zwischenschritte werden gespeichert und sind im Fehlerfall wiederherstellbar, außerdem bietet ein langlaufender Prozess die Möglichkeit, Verzweigungen innerhalb des Workflows parallel auszuführen.

### 2.4.3. Websphere Enterprise Service Bus

Ein Enterprise Service Bus [ESB] ist zunächst einmal ein Architekturmuster, und kein konkretes Software Produkt. Der Bus soll ein Bestandteil der Middleware-Infrastruktur darstellen, die die Implementierung einer SOA unterstützt. In einer SOA ist es wichtig, dass die Kommunikation zwischen zwei Service-Partnern nicht direkt, sondern über eine Zwischenschicht erfolgt [Faso8]. Die Aufgabe dieser Zwischenschicht ist nicht einfach nur der Transport der Nachrichten, sondern bietet sog. Vermittlungsunterstützung (Mediation Support). Dazu gehört zunächst einmal das Weiterleiten von Nachrichten an den jeweiligen Empfänger, der entweder festgelegt ist, oder dynamisch aus dem Inhalt der Nachrichten ermittelt wird. Hierbei können zusätzliche Funktionen ausgeführt werden, wie die Transformation der Nachricht von einem Format in ein anderes oder das Verwenden eines anderen Protokolls als dem, mit dem die Nachricht ursprünglich versendet wurde [Faso8], [KBC<sup>+</sup>07]. Ein ESB stellt also ein zentralisierte, logische Komponente im Sinne einer Hub-and-Spoke-Architektur dar, mit deren Hilfe die Kommunikation der Services untereinander unterstützt wird.

Der Websphere Enterprise Service Bus (WESB) stellt nun eine Umsetzung dieses Architekturmusters dar. Seine Aufgabe besteht darin, die Anfragen an Services abzufangen und zusätzliche Vermittlungsschritte, wie oben beschrieben, auszuführen, um die in einer SOA gewünschte lose Kopplung der Services zu ermöglichen [KMC<sup>+</sup>06].

Diese Funktionalität wird auch vom Websphere Process Server genutzt, weshalb der WESB auch zu den Supporting Services des WPS gehört. (s. a. Abbildung 2.2 auf Seite 17). Entsprechend basiert der WESB auch auf den selben Grundlagen wie der WPS, wie zum Beispiel der Service Component Architecture.

Für die ESB-Funktionalität existiert daher eine spezielle SCA-Komponente, die Mediation-Flow-Komponente. Aus SCA-Sicht unterscheidet sie sich nicht von jeder anderen SCA-Komponente, da heißt sie besitzt ein WSDL-Interface, mit dem sie ihre Funktionalität

zur Verfügung stellt, und kann mittels Referenzen andere SCA-Komponenten nutzen. Die einzelnen Verarbeitungsschritte innerhalb des Mediation-Flows werden von sog. Mediation Primitives ausgeführt. Sie sind die kleinsten Bestandteile des WESB und haben je nach Typ spezielle Aufgaben, wie zum Beispiel das Ändern des Nachrichtenformats, des Nachrichteninhalts, das Loggen von Messages, usw. Sie werden miteinander zu einer Verarbeitungsabfolge verbunden, die die Vermittlungsaufgabe des ESB ausführt [KBC<sup>+</sup>07].

Für die Mediation-Flow-Komponente steht ein spezielles SCA-Modul zur Verfügung, das Mediation-Modul. Wie jedes andere SCA-Modul wird dessen Funktionalität durch Exports verfügbar gemacht und es kann durch entsprechende Imports andere Module verwenden. Als einheitliche Datenstruktur werden auch hier Business Objects verwendet. Für die Mediation-Flow-Komponente wird eine spezielle Form davon verwendet, das Service Message Object (SMO). In der Mediation-Flow-Komponente geht es weniger um das Ausführen von Geschäftsprozessen als vielmehr um das Verarbeiten von Nachrichten. Daher ist das Datenformat der SMOs am typischen, technischen Aufbau von Nachrichten angelehnt. Die SMOs besitzen einen Body, der die Anwendungsdaten beinhaltet, sowie einen Header-Abschnitt, der protokollspezifische Informationen enthält. Darüber hinaus besitzt jedes SMO auch einen Kontext-Abschnitt für verarbeitungsspezifische Informationen (s. a. 3.3 auf Seite 27).

### 2.4.4. Eignung der Websphere Produkte als Implementierung der Architekturen

Der Websphere Process Server ist ein typisches Workflow-Management-System. Er stellt eine Laufzeitumgebung für Workflowinstanzen dar, und bietet die Möglichkeit Workflowmodelle zu importieren, andere Anwendungen im Rahmen eines Workflows aufzurufen, menschliche Tätigkeiten zu koordinieren, andere Workflows aufzurufen, sowie Administrations- und Überwachungsfunktionen. Die einzelnen Interfaces entsprechend der Referenzarchitektur der WfMC sind also implementiert. Mit BPEL wird eine standardisierte und verbreitete Workflowsprache unterstützt, die zusätzliche mit IBM-spezifischen Erweiterungen genutzt werden kann. Die Möglichkeit der zentralen Administration zeigt sich unter anderem in der Admin-Console, mit der sämtliche relevanten Einstellungen für das System vorgenommen werden können, wie zum Beispiel das Deployen von Prozessen. Mit dem Business Process Explorer wird eine Anwendung mitgeliefert, die einen Überblick über sämtliche laufenden Prozesse bietet, bis hin welche Aktivität gerade mit welchen Daten ausgeführt wird. Der Process Server erfüllt also alle Anforderungen, die oben an ein Workflow-Management-System gestellt werden.

Beim WESB und der PaF-Architektur sind diese Übereinstimmungen weniger offensichtlich. Die Probleme beginnen bereits damit, dass die PaF-Architektur weniger scharf definiert ist, als das beim WfMS der Fall ist. Wie beschrieben, gibt es unterschiedliche Auffassungen darüber, ob sich eine Pipe asynchron verhalten muss, oder ob die Ports sämtlicher Filter identisch sein müssen. Der WESB erfüllt viele der oben beschriebenen Anforderungen nicht. Ein Mediation-Flow besteht zwar aus einzelnen Komponenten, den Mediation Primitives. Diese sind aber keine voneinander unabhängigen Komponenten, sondern Bestandteile eines Mediation Moduls. Dieses Modul wird als Ganzes instantiiert, wenn eine Anfrage für den Mediation Flow eintrifft. Auch kann jede einzelne Instanz eines

Mediation Flows stets nur eine Anfrage gleichzeitig verarbeiten. Die einzelnen Primitives einer Instanz können nicht mit unterschiedlichen Anfragen beschäftigt sein. Damit kann der oben beschriebene Pipeline-Effekt zur Durchsatzsteigerung nicht genutzt werden. Des Weiteren stellen die Verbindungen der Primitives keine asynchrone Kommunikation dar. Tatsächlich läuft ein Mediatonflow stets in einer Transaktion ab. Die Recoveryvorteile, die man bei einer PaF-Architektur erwarten könnte, finden sich hier also auch nicht, während man eher Ähnlichkeit mit einem Workflowsystem sieht.

Zu einem WfMS gibt es jedoch entscheidende Unterschiede. Zunächst einmal ist die Zielsetzung beim WESB auf einer niedrigeren Abstraktionsebene als bspw. beim WPS. Beim WPS steht der Ablauf von Geschäftsprozessen im Mittelpunkt und die damit einhergehenden Anforderungen, wie beispielsweise unterbrechbare Prozesse und Unterstützung für menschliche Interaktion. Beim WESB geht es um die Transformierung und Vermittlung von Nachrichten, unabhängig davon, welche Zwecke die Nachrichten erfüllen. Es geht nicht darum Geschäftsfunktionen miteinander zu verbinden, sondern beliebige Endpunkte. Genauso, wie eine PaF-Architektur deutlich vielseitiger einsetzbar ist, da sie die Abstraktionsebene der Workflowmodelle nicht berücksichtigt, ist auch ein Mediation Flow prinzipiell nicht nur auf die Unterstützung von Geschäftsprozessen begrenzt, sondern von diesen unabhängig. Während man bei einem Workflow, den man auf dem WPS instantiiieren möchte, sich möglichst weit von technischen Details lösen will, spielen beim Mediationflow Nachrichtenformate und -protokolle eine wichtige Rolle. Die niedrigere Abstraktionsebene einer PaF-Architektur, die stärker die tatsächliche technische Umsetzung berücksichtigen muss, findet sich also beim WESB im Vergleich zum WPS bzw. einem WfMS wieder.

Dass die Verarbeitung und Weitergabe von Nachrichten im Mittelpunkt des Mediation Flows stehen, führt zu weiteren Merkmalen einer PaF-Architektur. Das Problem von Komponenten, deren Ports nicht identisch sind, findet sich hier wieder. Die jeweilige Nachricht muss stets so angepasst werden, dass die nachfolgenden Komponenten sie verarbeiten können. Dies zeigt sich beispielsweise in der Mediation Flow - Implementierung des Testzenarios (s. Kapitel 3.3 auf Seite 27). Dort finden sich eine große Anzahl von Business-Objekt-Mappern, die für diese Transformation der Nachrichten notwendig sind.

Des Weiteren kann jeglicher Austausch von Information zwischen den einzelnen Primitives nur über die Weitergabe von Nachrichten erfolgen. Dadurch gibt es, wie in einer PaF-Architektur, keine Trennung von Daten- und Kontrollfluss. Werden Daten, die in einer Primitive erzeugt werden, erst später im Mediationflow wiederbenötigt, müssen sie von Primitive zu Primitive über den Austausch von Nachrichten weitergereicht und dabei jeweils entsprechend transformiert werden. Durch diese Form des Informationsaustausches folgt auch, dass keine globale Information existiert. Eine Information, die an einer Stelle erzeugt wird, kann erst dann an einer anderen Stelle verfügbar sein, wenn diese die entsprechende Nachricht erhält. Dieser gravierende Nachteil einer PaF-Architektur findet sich also auch beim WESB.

Zur Eignung des WESB als Implementierung einer PaF-Architektur bleibt zu sagen, dass es sich sicher nicht um eine typische Umsetzung handelt. Die Primitives als Komponenten existieren nicht einzeln, sondern werden innerhalb des Moduls als ganzes erzeugt. Innerhalb der Instanz es solchen Modules befinden sich auch stets nur die Daten einer Anfrage. Es findet innerhalb eines Mediation Flows keine asynchrone Kommunikation statt und der komplette Prozess läuft in einer einzigen Transaktion ab. Damit ist die Robustheit des

## 2. Architekturen

---

Systems nicht besser als bei einem kurzlaufenden Workflow. Dem gegenüber steht die größere Nähe zur technischen Umsetzung, wie man sie bei einer PaF-Architektur erwartet und insbesondere der charakteristische Austausch von Informationen ausschließlich mittels Nachrichten. Die daraus resultierenden Konsequenzen, wie die Notwendigkeit häufiger Transformationen und insbesondere das Fehlen von global verfügbaren Informationen, stellen kennzeichnende Eigenschaften einer PaF-Architektur da. Hinzu kommt ein bedeutender Vorteil, der für den Einsatz des WESB im Rahmen dieser Arbeit spricht. Da es hier um den Vergleich zweier Architekturen anhand von konkreten Implementierungen geht, ist es notwendig, dass die verwendete Software sich technisch möglichst ebenbürtig sind. Ein kommerzielles Produkt, das seit Jahren produktiv im Einsatz ist und kontinuierlich weiterentwickelt und optimiert wird, mit einer anderen Software, beispielsweise aus dem Open Source Bereich, zu vergleichen, würde nur wenig Rückschlüsse auf die Architektur zulassen. Auch der Vergleich zweier kommerzieller Produkte bringt diese Probleme mit sich, da Unterschiede bei der Performance stets aus Mängeln bei der jeweiligen Implementierung oder auch der Konfiguration der Produkte herrühren könnten. Diese Probleme lassen sich natürlich nie ganz beseitigen, und resultieren einfach aus dem Versuch, Architekturen an Hand konkreter Implementierungen zu vergleichen. Durch die Verwendung zweier Produkte, die auf vielen gemeinsamen Grundlagen aufbauen und zusammen entwickelt und optimiert wurden, wird hier versucht, den Einfluss solcher Probleme möglichst gering gehalten.

# Testszenarios

---

Als Grundlage für die Testläufe wurde ein Szenario entworfen, das dann mittels der unterschiedlichen Prozesstypen implementiert wurde. Basis für dieses Szenario war das bei [HW03] beschriebene Loan Broker Szenario. Das hier verwendete Erweiterte Loan Broker Szenario ergänzt das einfache Szenario um zusätzliche Dienste und Abläufe. Im Folgenden wird zunächst der Prozess näher beschrieben, der durch das Szenario dargestellt wird. Hier soll nicht versucht werden, einen existierenden Wirtschaftsprozess realistisch abzubilden, sondern einen größeren Ablauf darzustellen, der ausreichend komplex genug ist, um anspruchsvolle Testläufe durchführen zu können. Die folgende Beschreibung des Szenarios dient lediglich dazu, eine plastischere Darstellung des Prozesses zu geben, als wenn kein Bezug zur Realität aufgezeigt würde.

Im Anschluss daran folgt eine Beschreibung, wie der Prozess unabhängig von den später verwendeten Prozesstypen in sog. Patterns dargestellt werden kann (s. Abschnitt 3.2 auf der nächsten Seite). Dadurch soll die technische Umsetzung des Szenarios zunächst einmal unabhängig von den später verwendeten Prozesstypen aufgezeigt werden. Die verwendeten Patterns beruhen ebenfalls auf [HW03].

### 3.1. Das Erweiterte Loan Broker Szenario

Das erweiterte Szenario stellt prinzipiell dieselbe Situation da, wie das einfache. Ein Kreditantrag wird eingereicht, es werden verschiedene Banken dazu befragt, und am Ende das beste Angebot an den Kunden zurückgegeben. Beim Erweiterten Loan Broker Szenario kommt nun hinzu, dass auch staatliche Förderprogramme mitgeprüft werden und darüber hinaus eine interne Überprüfung der Tilgungsfähigkeit stattfindet, um den Kunden auf eventuelle Risiken hinzuweisen.

Zu Beginn werden, wie beim einfachen Szenario, Informationen über den Kunden bei einem Rating-Agent abgefragt. Hierbei wird anhand der Sozialversicherungsnummer der Credit-Score und die Credit-History abgefragt.

Im Anschluss daran kann auf Wunsch des Kunden geprüft werden, ob der Kredit teilweise durch staatliche Subventionen ersetzt bzw. eine staatliche Bürgschaft gegeben werden

kann. Beide Formen staatlicher Unterstützung können entweder bei einem nationalen oder bei einem EU-Förderprogramm beantragt werden. Es kann auch durchaus beides, also eine Subvention und eine Bürgschaft, beantragt werden, wobei nicht beide aus dem selben Förderprogramm kommen müssen. Für beide Förderprogramme sind unterschiedliche Voraussetzungen zu erfüllen. Ein jeweils eigener Dienst ermittelt für die Subvention bzw für die Bürgschaft, ob bessere Aussichten bei dem nationalen Programm oder bei dem EU-Programm bestehen. Entsprechend werden danach die Anträge gestellt, und die Ergebnisse in den Kreditantrag eingearbeitet.

Im nächsten Schritt wird der Kreditantrag an mehrere Banken weitergeleitet. Hierzu wird der Eingangsnachricht des Prozesses eine Liste hinzugefügt, die angibt, welche Banken aufgerufen werden sollen. Wenn alle Banken geantwortet haben, wird das beste Angebot weitergegeben.

Im nächsten Abschnitt prüft der Loan-Broker die Tilgungsfähigkeit des Kunden, um diesen auf mögliche Risiken durch die Aufnahme des Kredits hinzuweisen. Er greift hierfür auf verschiedene Informationsdienste zurück. Hierbei werden Informationen über den Gesundheitszustand von der Krankenkasse des Kunden, über dessen Altersvorsorge aus den Daten der Rentenversicherung, sowie allgemeine Informationen über dessen Arbeitgeber abgefragt, um die wirtschaftliche Lage des Unternehmens und damit die Arbeitsplatzsicherheit zu bestimmen. Der Loan-Broker versucht möglichst wenige Dienste zu nutzen, da hierbei hohe Gebühren anfallen. Daher werden die Dienste in der Reihenfolge der Höhe ihrer Gebühren abgefragt. Bei der ersten negativen Antwort wird kein weiterer Dienst mehr aufgerufen, sondern nur die bisher gefundenen Risiken zurückgegeben. Da sich die Höhe der Gebühren stets ändern kann, ist die Reihenfolge der Dienste nicht fest im Prozessablauf implementiert, sondern kann an einer zentralen Stelle im Prozess angepasst werden.

Abschließend werden alle Informationen bestehend aus Kreditzusagen, Förderungen sowie Risiken bezüglich der Tilgungsfähigkeit gesammelt in der Antwortnachricht zurückgegeben.

## 3.2. Patternentwurf

Eine Darstellung des Szenarios mit Hilfe von Pattern zeigt die Abbildung 3.1 auf Seite 26. Die nachfolgenden Beschreibung bezieht sich auf dieses Diagramm. Zu den Funktionen der einzelnen Pattern sei noch einmal auf [HW03] verwiesen. Sie sind aber auch im Internet abrufbar [EAI], dort sind sie jedoch jeweils nur kurz beschrieben.

Zu Beginn des Prozesses wird der Aufruf des Rating-Agents dargestellt. Dies geschieht mittels eines Content Enrichers, der einen Endpoint aufruft. Prinzipiell findet solch ein Content Enricher quasi bei jedem Endpoint im Prozess Anwendung, da niemals exakt die Antwortnachricht des Endpoints weitergeleitet wird, sondern stets nur Informationen daraus der vorliegenden Nachricht hinzugefügt werden. Um das Modell übersichtlich zu halten, wurden bei den weiteren Endpoints die Darstellung eines Content Enrichers weggelassen. Aus dem selben Grund findet sich auch das Message Translator - Pattern nirgends, obwohl es quasi ständig verwendet wird.

Der folgende Content-Based Router ermittelt, ob zunächst die Möglichkeit staatlicher Förde-

ung geprüft werden soll. Ist dies nicht der Fall, wird die Nachricht direkt an den Bankbereich weitergeleitet. Es handelt sich hier also um eine Art Bypass, ähnlich dem Dertour-Pattern. Der Förderungenbereich zeigt den Verarbeitungsablauf für die staatlichen Förderprogramme. Der gesamte Bereich ist ein Composed Message Processor - Pattern mit einem Splitter und Router am Anfang und einer Aggregator-Komponente am Schluß. Der Splitter erzeugt aus der Ursprungsnachricht zunächst zwei unterschiedliche Anfrage-Nachrichten, mit denen die Förderungsfähigkeit abgefragt werden kann. Die Anfragenachrichten werden vom nachfolgenden Router an die passenden Dienste weitergeleitet. Die Antworten der beiden Dienste bestimmen, welche Förderungen bei welchem Förderprogramm beantragt werden. Die Logik dafür steckt im zweiten Content-Based Router, der die Nachrichten dann jeweils an die Endpoints der Förderprogramme weiterleitet. Deren Antworten werden am Schluß vom Aggregator zu einer Nachricht zusammengefügt.

Der Bankenbereich stellt im wesentlichen das klassische Loan Broker Szenario dar. Hier liegt ein Scatter-Gather - Pattern mit einer Recipient List vor. Das bedeutet, dass in der Nachricht festgelegt ist, welcher der Bankdienste aufgerufen werden soll. Zum Zwecke der Performance-Messung ist es möglich, hier eine beliebig lange Liste anzugeben, so dass ein Dienst auch mehrfach aufgerufen werden kann. Das macht für das Szenario zwar keinen Sinn, ermöglicht aber die Simulation beliebig vieler Dienstaufrufe an einer Stelle im Verarbeitungsablauf. Das Aggregator-Element übernimmt schließlich aus sämtlichen Antworten das günstigste Angebot.

Der Risikoanalysebereich stellt die interne Überprüfung der Tilgungsfähigkeit dar. Die verschiedenen Dienste sollen, wie oben beschrieben, in einer möglichst optimalen Reihenfolge aufgerufen werden, was mittels des Routing Slip - Patterns erfolgt. Zu diesem Zweck ist in der Komponente zu Beginn des Bereichs die optimale Abfolge der Dienste gespeichert, die dann als Routing Slip der Nachricht hinzugefügt wird. Von dort wird die Nachricht an die erste Komponente des Routing Slips weitergeleitet. Sofern dort keine negativen Informationen zurückgegeben werden, leitet der nachfolgende Router die Nachricht an die zweite Komponente entsprechend des Routing Slips weiter, und so fort. Jede Komponente ist also mit jeder Komponente mittels nachfolgendem Router verbunden, und die Abfolge wird über den Routing Slip bestimmt, der am Anfang des Bereichs der Nachricht hinzugefügt wird. Gibt eine der Komponenten eine negative Antwort zurück, wird die vorgegebene Route abgebrochen und die Nachricht direkt zum Ende der Prozesses weitergeleitet, wo sie dann schließlich zurückgegeben wird.

### 3. Testszenarios

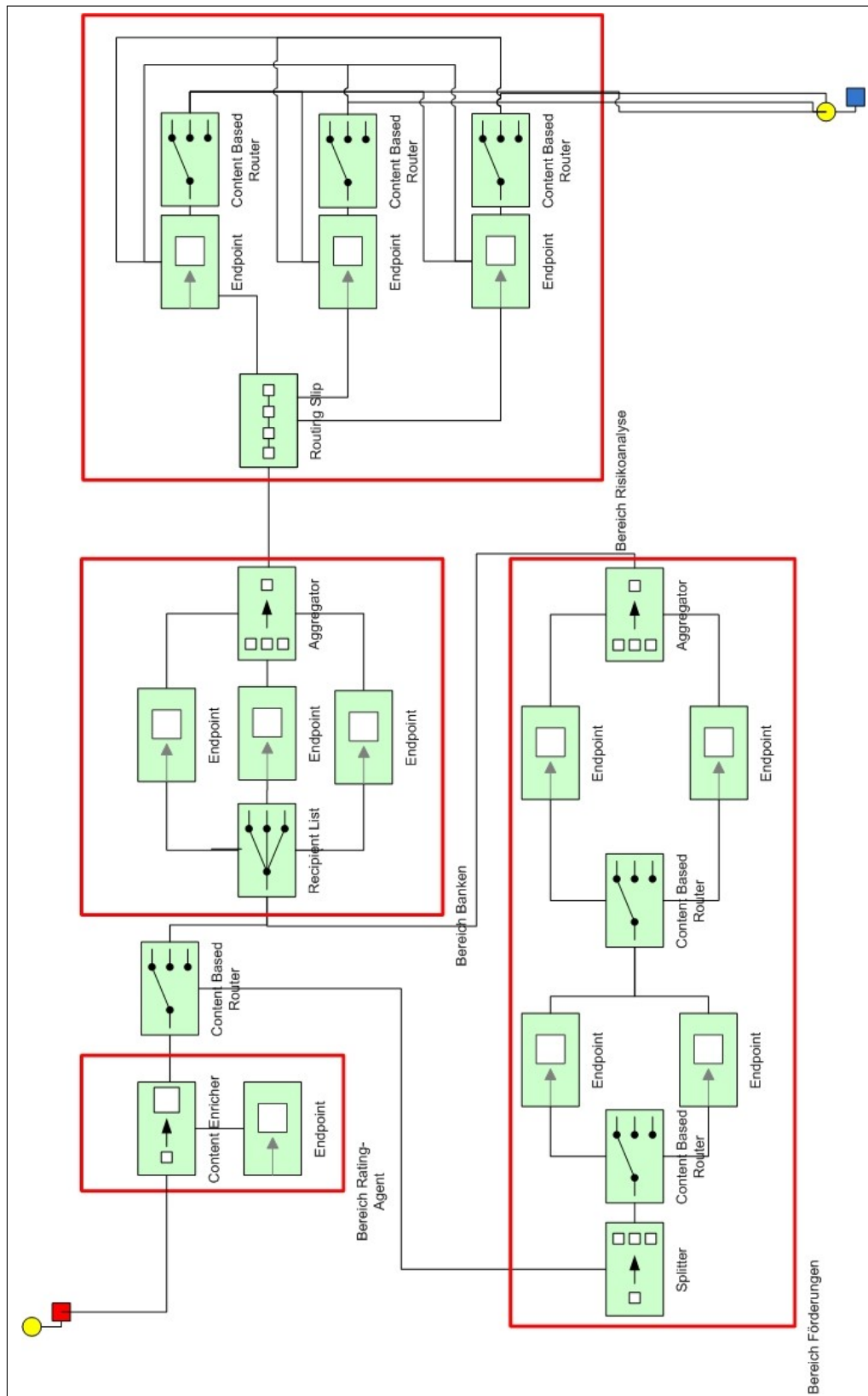


Abbildung 3.1.: Patternentwurf

### 3.3. Pipes and Filter - Implementierung

Der folgende Abschnitt beschreibt die Umsetzung des Loan Broker Szenarios mittels eines Mediation Flows. Die Abbildung 3.2 auf Seite 29 zeigt einen Gesamtüberblick über das Szenario, wobei die zusammengehörenden Elemente entsprechend gekennzeichnet sind. Die Bereiche Rating Agent, Förderungen, Banken und Risikoanalyse stellen die Umsetzung der gleichbezeichneten Bereiche aus dem Patterndiagramm dar. Auf den folgenden Seiten wird dann genauer auf die Implementierung der einzelnen Abschnitte eingegangen und, wo es nicht offensichtlich ist, erläutert, warum bestimmte Patterns auf die vorliegende Weise umgesetzt wurden.

#### Die Kontexte

Beim Mediation Flow spielen die sog. Kontexte einer Nachricht eine wichtige Rolle. Da sie bei den folgenden Beschreibungen immer wieder erwähnt werden, soll hier kurz etwas zur Funktionsweise gesagt werden. Für das Szenario sind zwei Kontexte von Bedeutung: der Transient Context und der Shared Context. Der MediationFlow zeichnet sich dadurch aus, dass eine Nachricht von Komponente zu Komponente weitergereicht wird, also von einem Mediation Primitive zum nächsten. Diese Nachricht bleibt dabei nicht immer dieselbe, sondern wird fast immer transformiert. Besonders wenn ein externer Service aufgerufen wird, gibt das entsprechende ServiceCall-Primitive dessen Antwortnachricht weiter. In der Aufrufnachricht der Primitive sind aber im allgemeinen auch Daten enthalten, die nicht für den externen Service gedacht sind, sondern erst später im Mediation Flow benötigt werden. Damit diese auch in der Ausgangsnachricht wieder enthalten sind, müssen sie im sog. transienten Kontext (Transient Context) der Nachricht enthalten sein. Der Transient Context wird also für Daten benötigt, die über mehrere Mediation Primitives hinweg in der jeweils aktuellen Nachricht bleiben soll. Der Inhalt des Transient Contexts der Eingangsnachricht einer Mediation Primitive wird also stets der jeweiligen Ausgangsnachricht hinzugefügt, und bleibt so erhalten.

Der Shared Context hingegen wird für Aggregationen benötigt, also für Prozessabschnitte, in den das FanOut- und das FanIn-Primitive verwendet werden (im folgenden Text sind das FanOut-Primitive als Splitter und das FanIn-Primitive als Aggregator bezeichnet, da das ihre Funktion besser bezeichnet). Diese Splitter- und Aggregator-Primitives werden immer dann verwendet, wenn dieselbe Nachricht an mehrere weitere Primitives geschickt werden soll. Dies ist beispielsweise beim Aufruf der Banken notwendig. Dieser Aufruf erfolgt aber nicht parallel, sondern sequentiell. Die in den Splitter eingehende Nachricht wird zunächst auf einem seiner ausgehenden Pfade weitergeleitet bis sie schließlich beim zugehörigen Aggregator ankommt. Dort wird geprüft, ob die eingestellte Abschlussbedingung erfüllt ist (bspw. ob bereits eine festgelegte Anzahl an Nachrichten eingetroffen ist). Ist diese Bedingung noch nicht erfüllt, wird die Kontrolle wieder zurück an den Splitter übergeben, der die ursprüngliche Nachricht erneut auf einem anderen Pfad abschickt. Die Informationen der ersten Nachricht werden aber nicht im Aggregator gespeichert. Tatsächlich geht der Inhalt des Body und des Transient Contexts der Nachricht verloren. Deren Werte entsprechen in

### 3. Testszenarios

---

jeder Nachricht, die der Splitter losschickt, der ursprünglichen Eingangsnachricht. Lediglich Änderungen, die im sog. Shared Context der Nachricht vorgenommen werden, bleiben jeweils nach Rückgabe der Kontrolle an den Splitter in dessen neuer Nachricht vorhanden. Sollen also Ergebnisse, die auf verschiedenen Pfaden des Splitters entstanden sind, erhalten bleiben, müssen sie jeweils im Shared Context gespeichert werden, da sie sonst nicht in die jeweils aktuelle Nachricht hineinkopiert werden. Wenn schließlich die Abschlussbedingung des Aggregators erfüllt ist, wird die letzte empfangene Nachricht weitergeleitet. Die Ergebnisse, die auf dem Pfad entstanden sind, den diese Nachricht durchlaufen hat, können in deren Body bzw. in ihrem Transienten Context enthalten sein. Die Ergebnisse, die auf den anderen Pfaden entstanden sind, können aber nur im Shared Context der aktuellen Nachricht enthalten sein, weil dessen Inhalt stets von Nachricht zu Nachricht weitergereicht wurde. Beispiele hierfür finden sich bei der Umsetzung des Förderbereichs auf Seite 30 und des Bankenbereichs auf Seite 32.

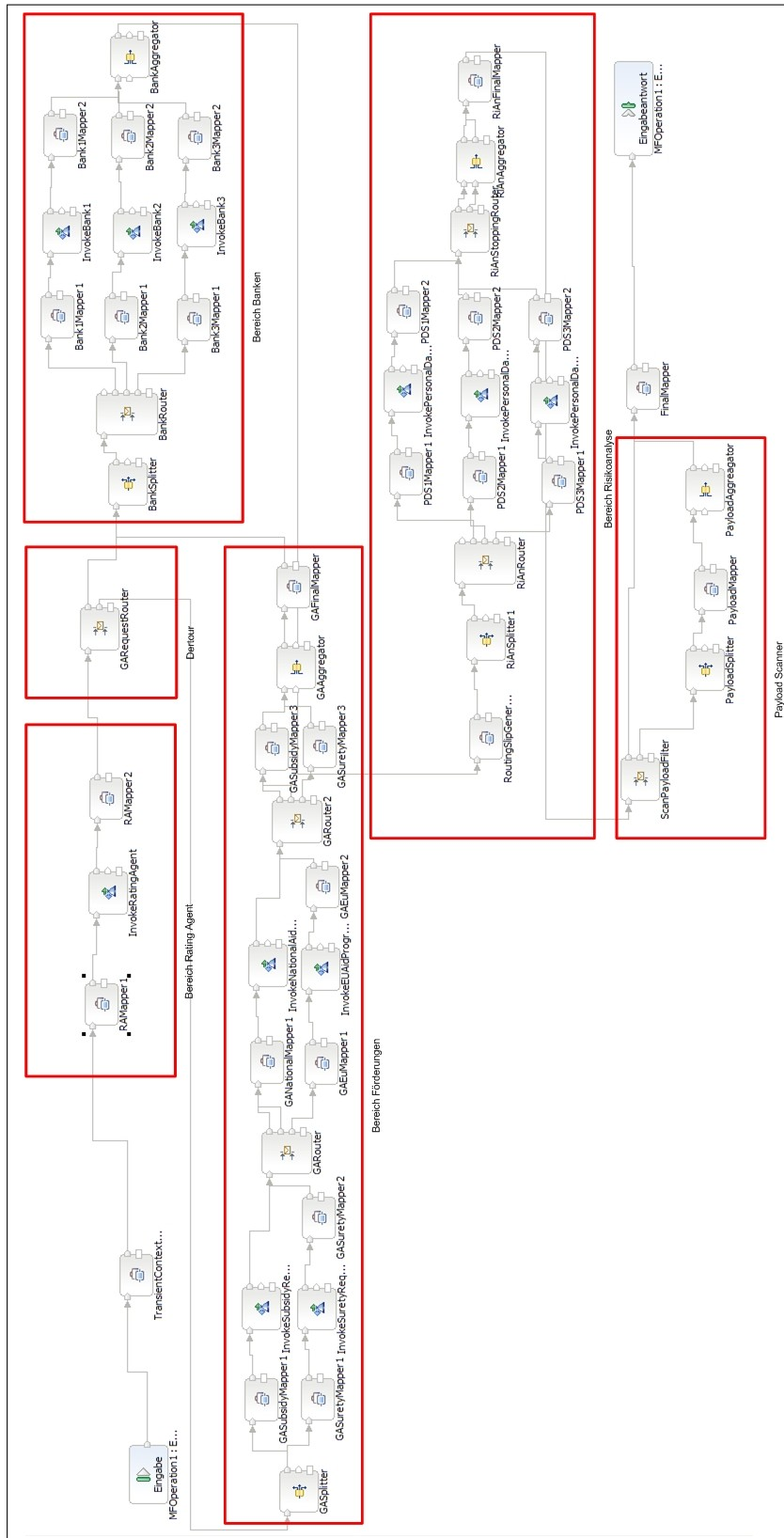


Abbildung 3.2.: Mediationflow

#### Beginn des Prozesses

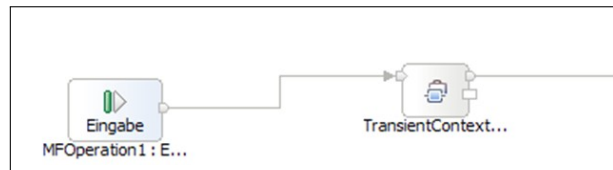


Abbildung 3.3.: Beginn des MediationFlow

Nach Eingang der Nachricht in den MediationFlow werden im TransientContextMapper zunächst alle Werte der Eingangsmessage in den Transient Context kopiert. Damit stehen diese Werte für spätere Aufrufe zur Verfügung.

#### Bereich Rating Agent - Patterns: Dertour

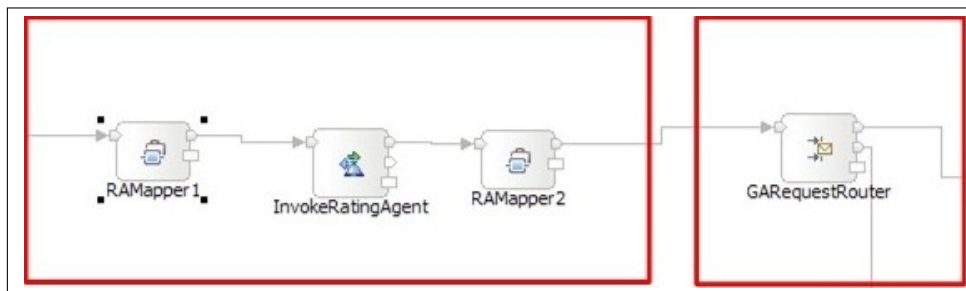


Abbildung 3.4.: MediationFlow - Rating Agent

Die Nachricht wird weiter an den Business Object - Mapper (BO-Mapper), hier der RAMapper1, des Rating Agent geleitet. Dieser Mapper transformiert die eingehende Nachricht in das passende Format für den Rating Agent, der anschließend damit aufgerufen wird (InvokeRatingAgent). Der RAMapper2 schreibt schließlich die Ergebnisse des Aufrufes aus dem Body der Ausgangsnachricht in deren Transient Context, damit sie später weiterhin zur Verfügung stehen.

Abhängig davon, ob in der Aufrufnachricht des Prozesses vermerkt ist, ob öffentliche Förderungen erwünscht sind, leitet der GARquestRouter anschließend die Nachricht entweder direkt an den Bankabschnitt oder erst an der Förderungsbereich weiter.

#### Bereich Förderungen - Patterns: Context Based Routers

Im Förderungsabschnitt (GovernmentalAid) wird zunächst mittels eines Splitters (GASplitter) die Nachricht in eine Anfrage an den Dienst für Subventionsanforderungen (InvokeSubsidyRequirementService) sowie in eine Anfrage an den Dienst für Bürgschaftsanforderungen

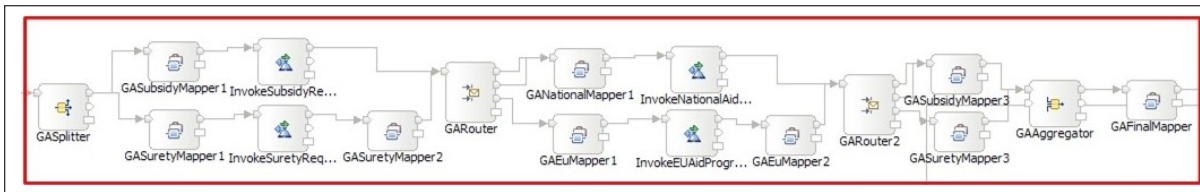


Abbildung 3.5.: MediationFlow - Förderungen

(InvokeSuretyRequirementService) aufgeteilt. Hier sei noch einmal erwähnt, dass diese Pfade nicht parallel ausgeführt werden. Es wird also zunächst eine Nachricht auf einen der ausgehenden Pfade des Splitters geschickt, und diese wird bis zum zugehörigen Aggregator (GAAggregator) weiterverarbeitet. Erst dann wird der Splitter die zweite Nachricht auf dem anderen Pfad losschicken.

Die Business Object - Mapper (GASubsidyMapper<sub>1</sub> bzw. GASuretyMapper<sub>1</sub>) transformieren die jeweilige Nachricht in das passende Format für den jeweiligen Dienst. Zurückgegeben wird von jedem Dienst eine Antwort, die enthält, welche Subvention bzw. Bürgschaft bei der nationalen Behörde und bei der EU verfügbar sind, zusammen mit einer jeweiligen Erfolgseinschätzung für die Beantragung. Der BO-Mapper GASuretyRequest<sub>2</sub> wird benötigt, da die jeweiligen Antwortnachrichten der Dienste unterschiedliche Formate besitzen, aber der anschließende Router (GARouter) nur ein eingehendes Format akzeptiert. Daher wird die Antwort des Bürgschaftsanforderungsdienstes (InvokeSuretyRequest) in dasselbe Antwortformat wie das des Subventionsanforderungsdienstes (InvokeSubsidyRequest) transformiert, das dann als einheitliches Format weiterverwendet wird.

Abhängig von der Erfolgseinschätzung leitet der GARouter die Nachricht entweder an die nationale Behörde (InvokeNationalAidProgram) oder an die EU-Behörde (InvokeEuAidProgram) weiter. Die jeweiligen Mapper (GANationalMapper<sub>1</sub> und GAEUMapper<sub>1</sub>) erzeugen zuvor den jeweiligen Antrag, entweder für Subvention oder für eine Bürgschaft (abhängig davon, ob die Nachricht zuvor vom SubsidyRequirementService oder vom SuretyRequirementService kam). Der GAEUMapper<sub>2</sub> sorgt wieder für ein einheitliches Format am Eingang des GARouters<sub>2</sub>, wie das der GASuretyMapper für den GARouter tut. Der GARouter<sub>2</sub> sorgt anschließend dafür, dass die Antworten von EU- oder nationaler Behörde entsprechend ihrem Inhalt (also entweder Subvention oder Bürgschaft) an den passenden Mapper weitergeleitet werden (GASubsidyMapper<sub>3</sub> oder GASuretyMapper<sub>3</sub>), die dann die Antworten in die passende Stelle des Shared Contextes einfügen. Die GAAggregator-Komponente leitet weiter, sobald die zweite Nachricht eingeht. Der GAFinalMapper transformiert die Nachricht schließlich so, dass sie vom Format her wieder in den normalen Prozessablauf eingefügt werden kann (also vor dem BankSplitter s. u.).

### Bereich Banken - Patterns: Recipient List, Scatter Gather

Im Bankabschnitt wird der Kreditantrag an eine Anzahl an Banken weitergeleitet. Die Banken werden in einer Bankenliste in der Aufrufnachricht des Prozesses angegeben. Möglich sind

### 3. Testszenarios

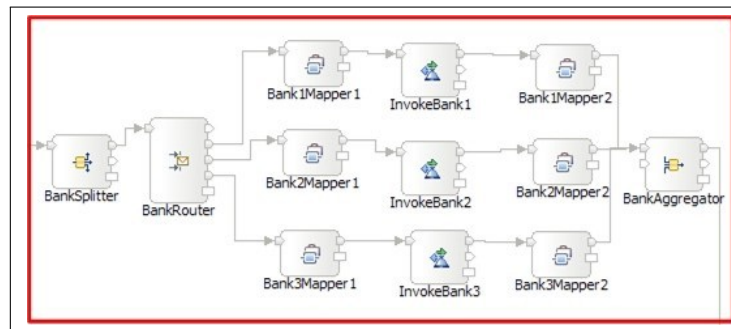


Abbildung 3.6.: MediationFlow - Banken

Bank1 bis Bank3, wobei jede Bank mehrfach aufgerufen werden kann. Zu diesem Zweck erzeugt der **BankSplitter** für jeden Eintrag der Bankenliste eine Nachricht. Der **Bankrouter** leitet diese entsprechend dem jeweiligen Eintrag an Bank1, Bank2 oder Bank3 weiter. Der erste Mapper einer Bank erzeugt die jeweilige Anfrage, der zweite Mapper einer Bank fügt das Kreditangebot in den **SharedContext** ein, sofern dort nicht bereits ein besseres Angebot eingetragen ist. Der **BankAggregator** leitet weiter, sobald alle Einträge der Bankenliste abgearbeitet sind.

#### Bereich Risikoanalyse - Patterns: Routing Slip

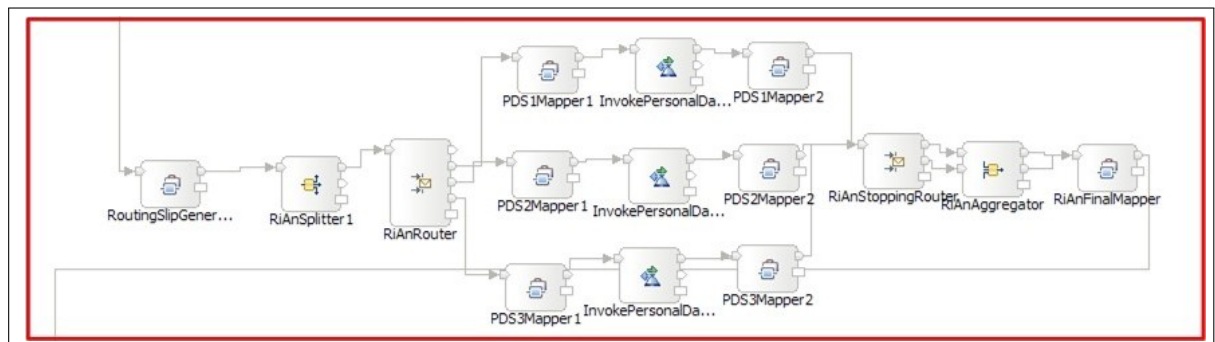


Abbildung 3.7.: MediationFlow - Risikoanalyse

Im Bereich Risikoanalyse sollen drei verschiedene Dienste aufgerufen werden, die jeweils eine Risikoeinschätzung des Kreditnehmers zurückgeben. Die Reihenfolge der Aufrufe wird zu Beginn im **RoutingSlipGenerator** festgelegt, der eine entsprechende Liste (den **RoutingSlip**) der Nachricht hinzufügt. Die naheliegenste Umsetzung des **Routing Slip** - Patterns wäre ein Router direkt nach dem **Routing Slip** sowie nach jedem Aufruf der Dienste, wobei diese Router jeweils untereinander verbunden sind. Durch die sequentielle Funktionsweise des **Splitter-Aggregator-Paares** bietet sich hier aber eine einfachere Umsetzung an, die den Vorteil

hat, dass keine extra Logik für die Reihenfolge implementiert werden muss. Normalerweise müsste nämlich nach jedem Schritt markiert werden, welche Einträge des Routing Slips bereits abgearbeitet sind, und der nachfolgende Router müsste prüfen, welches der aktuelle Eintrag ist. Der RiAnSplitter<sub>1</sub> durchläuft nun aber einfach die Einträge des Routing Slips und der RiAnRouter ruft entsprechend des aktuellen Eintrags den passenden Dienst auf (ähnlich wie es beim Bereich Banken geschieht). Da bei der Rückgabe eines hohen Risikos kein weiterer Dienst aufgerufen werden soll, prüft der RiAnStoppingRouter bei jeder Antwort das Risikolevel. Falls dieses hoch ist, wird die Nachricht an das Input-Stop-Terminal des RiAnAggregators weitergeleitet. Dieser leitet dann sofort die Nachricht weiter, ansonsten wartet er, bis alle Einträge des Routing Slips abgearbeitet sind. Der RiAnFinalMapper transformiert schließlich die Nachricht in das Nachrichtenformat der Ausgangsnachricht des Prozesses.

#### Payload Scanner und Ende des Prozesses

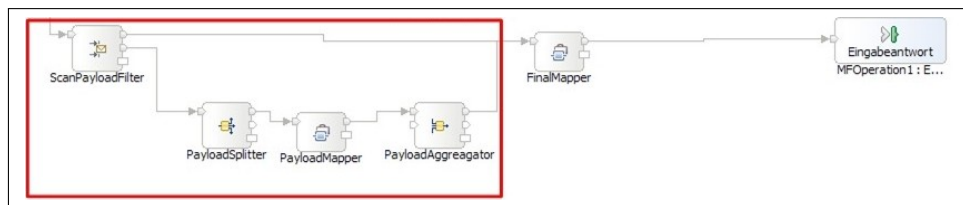


Abbildung 3.8.: PayloadScanner und Ende des Mediationflows

Der letzte Bereich, der sich nicht im Patternentwurf des Szenarios wiederfindet, ist der Bereich des Payload Scanners. Dieser ergibt sich nicht aus den Anforderungen des Szenarios, sondern wurde eingefügt, um die Auswirkungen unterschiedlich großer Nachrichtengrößen besser testen zu können. Zu diesem Zweck ist der Eingangsnachricht des Prozesses das Payload-Element hinzugefügt. Es besteht aus bis zu 50 Unterelementen von je 10 Kb-Größe und soll es ermöglichen, auch große Nachrichten zu simulieren. Da die Payload nirgends sonst im Prozess Verwendung findet, kann in diesem Bereich des Prozesses jedes Teilelement der Payload einmal innerhalb der Nachricht kopiert werden. Zu diesem Zweck existiert in der Eingangsnachricht des Prozesses das Feld ScanPayload, indem ein boolescher Wert übergeben wird. Abhängig von diesem Wert leitet der ScanPayloadFilter seine Nachricht entweder direkt an den FinalMapper weiter, oder übergibt sie an den PayloadSplitter. Dieser Splitter erzeugt sequentiell für jedes Element der Payload eine Nachricht. In jeder dieser Nachrichten wird das jeweils aktuelle Payload-Element vom PayloadMapper in ein anderes Feld kopiert. Der PayloadAggregator leitet dann die letzte Nachricht weiter, sobald alle Elemente der Payload durchgearbeitet wurden. Der FinalMapper kopiert anschließend alle benötigten Elemente aus dem Transient und Shared Context in den Body der Nachricht, der dann schließlich als Ausgangsnachricht des Prozesses zurückgegeben wird.

## 3.4. Workflow - Implementierung

Im folgenden wird die Umsetzung des Loan Broker Szenarios mittels eines MicroFlows beschrieben. Abbildung 3.9 auf der nächsten Seite zeigt einen Gesamtüberblick über das Szenario. Die farbig hervorgehobenen Bereiche entsprechen den gleichbezeichneten Abschnitten aus dem Patterndiagramm (Abbildung 3.1 auf Seite 26). Anschließend wird auf die einzelnen Abschnitte noch einmal näher eingegangen.

Der hier vorgestellte Workflow beschreibt gleichzeitig den MacroFlow, der nahezu identisch aufgebaut ist. Der einzige Unterschied ist, dass zwischen den hier rot eingezeichneten Bereichen jeweils noch ein Snippet-Element eingefügt wurde. Dieses dient im Normalfall dazu, Javacode in den Prozess einzufügen, was in diesem Fall aber nicht getan wird. Stattdessen wird die Möglichkeit des Elements genutzt, Transaktionsgrenzen festzulegen. Man kann nämlich angeben, ob das Snippet-Element an der bisherigen Transaktion teilnehmen, oder ob eine neue Transaktion gestartet werden soll. Hierdurch wird gewährleistet, dass jeder Bereich in einer separaten Transaktion ausgeführt wird.

### Architektur- und implementierungsbedingte Besonderheiten

Da der Patternentwurf näher an einem MediationFlow als an einem BPEL-Prozess liegt, konnten nicht alle Patterns direkt übernommen werden. Das Dertour-Pattern des Bereichs Förderungen beschränkt sich hier darauf, dass der komplette Abschnitt in einem Choice-Element liegt. Außerdem wurde der Bereich so implementiert, dass der Subventionsabschnitt sowie der Bürgschaftsabschnitt parallel zueinander ausgeführt werden können, was jedoch nur für den MacroFlow relevant ist (s. S. 37).

Der Routing Slip wurde anders als beim Mediation Flow mit anderen Mitteln als der Bankenbereich umgesetzt. Prinzipiell hätte sich auch beim Routing Slip ein ForEach-Element angeboten, jedoch ließ sich die EarlyExit-Bedingung für einen eventuellen vorzeitigen Abbruch nicht geeignet nutzen. Daher ist der Routing Slip mittels einer While-Schleife mit Index-Variable realisiert (s. S. 38).

### 3.4. Workflow - Implementierung

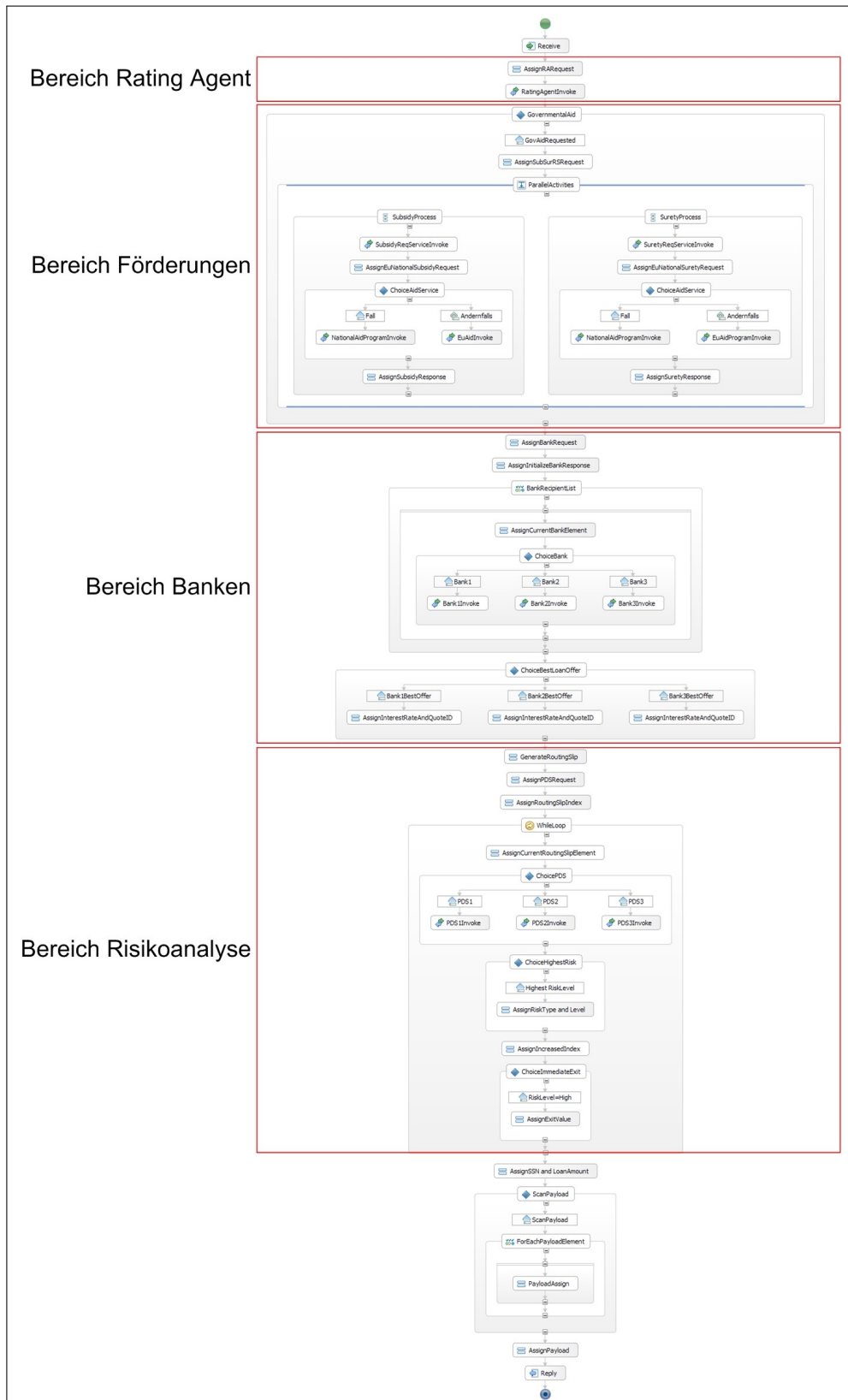


Abbildung 3.9.: MicroFlow

### 3. Testszenarios

#### Bereich Rating Agent

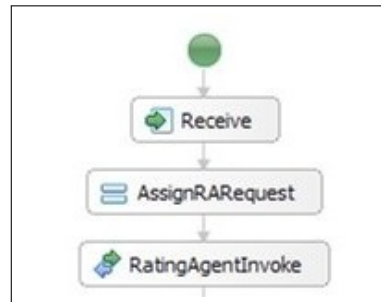


Abbildung 3.10.: MicroFlow - Rating Agent

Nach dem Aufruf sind zunächst alle Informationen der Eingangsnachricht in der Input-Variable des Prozesses vorhanden. Die notwendigen Informationen zum Aufruf werden durch AssignRARRequest in die Input-Variable des Rating Agent (RatingAgentInvoke) kopiert.

#### Bereich Förderungen - Patterns: Context Based Routers



Abbildung 3.11.: MicroFlow - Förderungen

Der komplette Förderungsabschnitt befindet sich in einem Choice-Element, da Förderungen nur beantragt werden, wenn dies in der Eingangsnachricht entsprechend vermerkt ist. Die Prozessabschnitte für Subventionen und Bürgschaften sind, anders als beim Mediation Flow, voneinander getrennt, so dass sie parallel ablaufen können. Beide werden mit derselben Input-Variable gestartet, in die die notwendigen Daten im AssignSubSurRSRequest-Element kopiert werden.

Der Ablauf in beiden Prozessabschnitten ist der gleiche: Zunächst wird der Anforderungsdienst für die jeweilige öffentliche Hilfe aufgerufen (SubsidyReqServiceInvoke bzw. SuretyReqServiceInvoke), der zurückgibt, in welcher Höhe und mit welcher Erfolgswahrscheinlichkeit mit Unterstützung zu rechnen ist. Anschließend werden den Inputvariablen der Förderdienste die notwendigen Werte zugewiesen. Abhängig von der Erfolgswahrscheinlichkeit werden anschließend im jeweiligen ChoiceAidService-Element entweder der Förderdienst der nationalen Behörde (NationalAidProgramInvoke) oder der EU-Behörde (EUAidProgramInvoke) aufgerufen. Deren Rückgaben werden anschließend in die Variablen der Ausgangsnachricht des Prozesses kopiert (AssignSubsidyResponse bzw. AssignSuretyResponse). Die beiden Prozessabschnitte werden im MicroFlow sequentiell durchgeführt, da dort keine parallelen Pfade unterstützt werden. Im MacroFlow hingegen werden die Prozessabschnitte parallel ausgeführt.

### Bereich Banken - Patterns: Recipient List, Scatter Gather

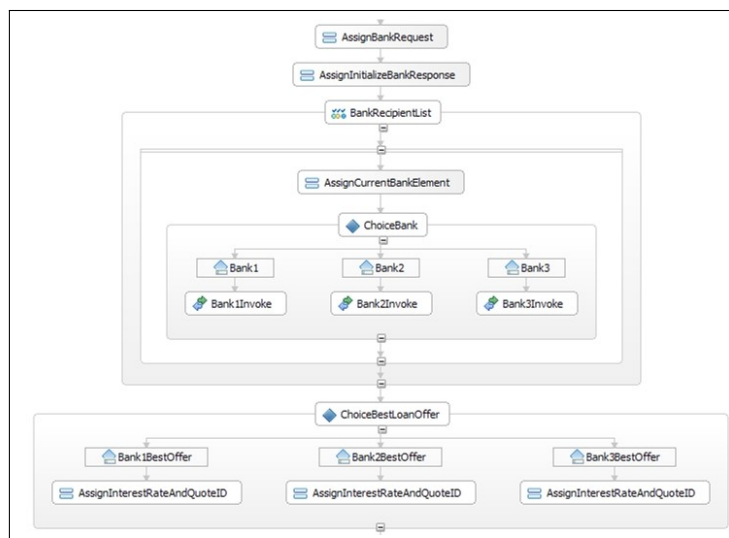


Abbildung 3.12.: MicroFlow - Banken

Für den Bankbereich werden zunächst der Input-Variable für die Bankdienste im AssignBankRequest-Element Werte zugewiesen. Außerdem wird die Output-Variable mit geeigneten Werten initialisiert, damit es beim späteren Vergleich für die Auswahl des besten Angebotes nicht zu Fehlern kommt. Der folgende, erste Teil des Bankabschnittes befindet sich

### 3. Testszenarios

---

in einem ForEach-Element. Dieses durchläuft alle Elemente der Bankliste der Input-Variable des Prozesses. Im AssignCurrentBankElement-Element wird der aktuelle Eintrag der Liste in eine Variable kopiert. Im ChoiceBank-Element wird dann auf Grundlage dieses Variablenwertes bestimmt, welcher Bankdienst aufgerufen werden soll (Bank1Invoke, Bank2Invoke bzw. Bank3 Invoke).

Da die Bankliste der Eingangsnachricht beliebig viele Einträge besitzen kann, ist es möglich, einen einzelnen Bankservice mehrfach aufzurufen. Da die Werte der Inputvariable sich jedoch nicht ändern, sind die Rückgabewerte stets die selben, so dass bei der anschließenden Aggregation effektiv nur drei Werte betrachtet werden müssen, die sich in der Output-Variable des jeweiligen Bankdienstes befinden. Im zweiten Teil des Bankbereiches wird im ChoiceBestLoanOffer-Element verglichen, welche der drei Outputvariablen das beste Angebot enthält, und die entsprechenden Daten in die Output-Variable des Prozesses kopiert (AssignInterestRateAndQuoteID).

#### **Bereich Risikoanalyse - Patterns: Routing Slip**

Im Bereich Risikoanalyse sollen drei Dienste in einer bestimmten Reihenfolge aufgerufen und abschließend das Risiko mit der höchsten Gefährdung zurückgegeben werden. Falls einer der Dienste dabei ein hohes Risiko zurückgibt, sollen die nachfolgenden Dienste nicht mehr aufgerufen werden. Zunächst wird ein Routing Slip generiert, indem eine Variable mit einer bestimmten Abfolge der Namen der drei Dienste versehen wird (GenerateRoutingSlip). Anschließend wird die Input-Variable der Dienste mit Werten versehen (AssignPDSRequest) und eine Index-Variable mit 1 initialisiert (AssignRoutingSlipIndex). Der eigentliche Ablauf findet in einem WhileLoop-Element statt. Die Schleife bricht ab, wenn die Index-Variable einen Wert von 4 aufweist. Zu Beginn jedes Durchlaufs wird das aktuelle Element des Routing Slips (ermittelt durch den aktuellen Wert des Index) in eine Variable kopiert (AssignCurrentRoutingSlipElement). Abhängig vom Wert dieser Variable wird im anschließenden Choice-Element (ChoicePDS) der jeweilige Dienst aufgerufen. Im nachfolgenden Choice-Element (ChoiceHighestRisk) wird der aktuelle Rückgabewert mit dem Wert verglichen, der bereits in der Output-Variable des Prozess enthalten ist. Falls die aktuelle Rückgabe das höhere Risiko aufweist, wird der Wert in die Output-Variable des Prozesses kopiert. Im nächsten Schritt wird die Index-Variable um eins erhöht (AssignIncreasedIndex). Abschließend wird geprüft, ob das aktuelle Risikolevel hoch ist (ChoiceImmediateExit). Ist dies der Fall wird der Wert der Index-Variable auf 4 erhöht (AssignExitValue) und damit der Abruch der While-Schleife erreicht. Die Schleife endet spätestens nach drei Durchläufen, wenn also jeder Dienst einmal aufgerufen wurde.

In den letzten beiden Assign-Elementen werden schließlich noch die unveränderten Werte der Input-Variablen des Prozesses, also die SSN, die LoanAmount sowie die Payload in die Output-Variable des Prozesses kopiert (AssignSSNandLoanAmount bzw. AssignPayload).

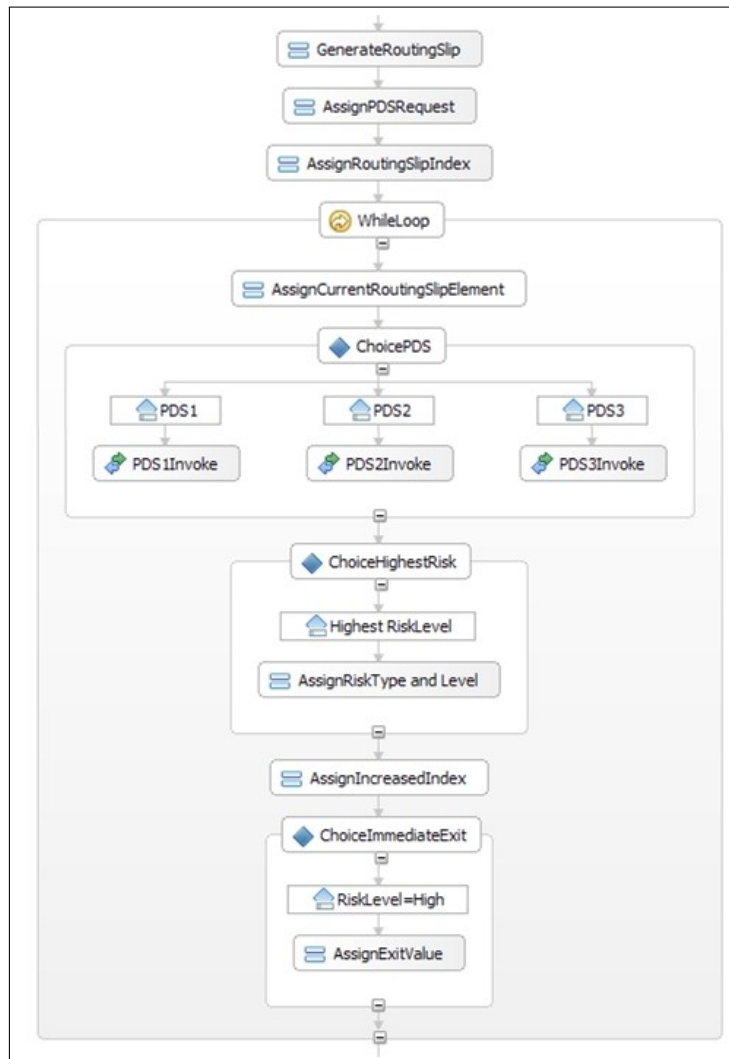


Abbildung 3.13.: MicroFlow - Risikoanalyse

### Payload Scanner und Ende des Prozesses

Der letzte Bereich entspricht dem Payload Scanner - Abschnitt, der bei der MediationFlow-Umsetzung bereits beschrieben wurde (s. den entsprechenden Abschnitt auf Seite 33). Im ersten Assign-Element (AssignSSN and Loan Amount) werden die Werte der SSN und der LoanAmount in die Variable der Ausgangsnachricht geschrieben. Anschließend beginnt der eigentliche ScanPayload-Abschnitt. Er befindet sich in einem Choice-Element und wird nur betreten, wenn die entsprechende Variable der Eingangsnachricht des Prozesses den passenden Wert enthält. Als nächster Schritt wird mit dem ForEach-Element für jedes Unter-element der Payload eine Assign-Operation ausgeführt (PayloadAssign), bei der das jeweilige Element einmal kopiert wird. Nach dem ScanPayload-Abschnitt wird die gesamte Payload

### 3. Testscenarios

---

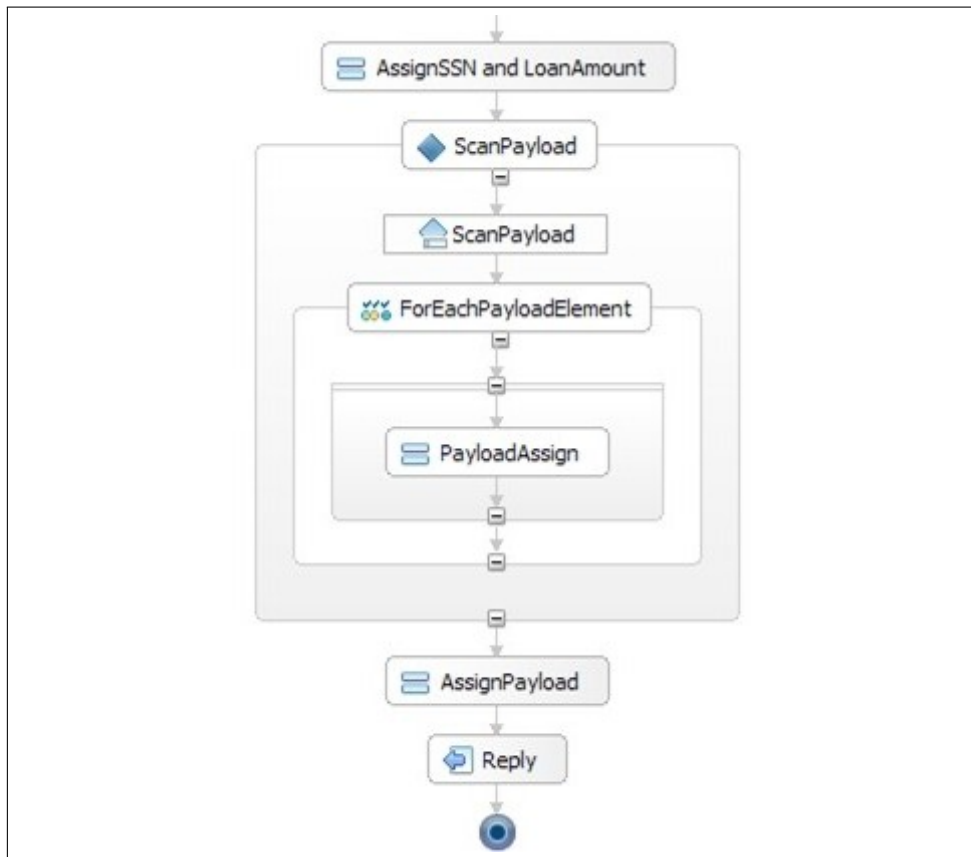


Abbildung 3.14.: PayloadScanner und Ende des Microflows

in die Ausgangsvariable des Prozesses kopiert und der Prozess gibt die Ausgangsnachricht zurück.

# Rahmenbedingungen der Tests

---

## 4.1. Testumgebung

Die einzelnen Testläufe wurden auf einem Intel Core 2 Quad CPU mit 2,4 GHz und 4 GB RAM durchgeführt. Als Betriebssystem wurde Windows XP Professional mit Service Pack 2 verwendet. Der verwendete Server für die Workflows (Micro- und Macro-Flow) war der IBM Websphere Process Server 6.1.2.0. Zum Zeitpunkt des Testes war dies die aktuellste Version. Für die Ausführung der Mediation Flows ist der enthaltene Websphere Enterprise Service Bus verantwortlich.

Beide Servertypen benötigen eine leistungsfähige Datenbank im Hintergrund, die auch für die Erhebung der Messdaten wichtig ist (s. a. Kapitel 4.2 auf der nächsten Seite). Hierfür wurde eine IBM DB2 v9.5.0.808 verwendet. Der Vorteil einer DB2 liegt vor allem darin, dass die Websphere Produkte vielfach auf diese Datenbank hin optimiert sind, und die meisten Beschreibungen von Tuningmaßnahmen die Verwendung einer DB2 voraussetzen. Die Datenbank war, wie auch die Server, auf der selben Festplatte wie das Betriebssystem installiert.

Tatsächlich sind die Standardeinstellungen der Produkte nicht geeignet, um eine hohe parallele Last zu verarbeiten. Daher war es notwendig, viele dieser Einstellungen zu verändern. Grundlage hierfür waren im wesentlichen [PT008] und [MPGMo8]. Eine Übersicht über die relevanten Parameter der Server und der Datenbank sowie die verwendeten Werte befinden sich im Anhang. Dort sind auch die jeweiligen Standardwerte beschrieben, und wo sie sich ändern lassen.

Die Webservices, die von den einzelnen Flows aufgerufen werden, liefen auf einem separaten System. Die Anforderungen an dieses System waren moderat. Daher bestand es aus einem Intel Pentium M Processor mit 1,6 GHz und 512 MB RAM. Der Server für die Webservices war ein Apache Tomcat 6.0.14. Die Verbindung zwischen den Systemen wurde durch ein 100 MB-Ethernet-Netzwerk hergestellt. Als Betriebssystem beider Systeme wurde Windows XP Professional mit Service Pack 2 verwendet.

Zur Durchführung der Tests wurde SoapUI 2.0.2 [SOU] verwendet. Mit Hilfe dieses Programms können Testnachrichten für Webservices erstellt werden, also auch für die Business- und Mediation-Flows, die jeweils über ein Webservice-Interface verfügen. Diese Nachrichten

können von einer einstellbaren Anzahl von Threads parallel übermittelt werden, bis eine festgelegte Anzahl von Testaufrufen durchgeführt wurde. SoapUI ermittelt hierbei unter anderem die durchschnittliche Antwortzeit der Anfragen sowie den durchschnittlichen Durchsatz an Transaktionen pro Sekunde. Das Programm lief bei den Tests auf dem selben System wie die Websphere-Server, da SoapUI nur wenig Ressourcen beansprucht, und Verzögerungen durch die Übermittlung von Nachrichten dadurch minimiert wurden.

### 4.2. Messwerkzeuge

Bei einem Performancevergleich können prinzipiell viele Größen gemessen werden, wie zum Beispiel die Antwortzeiten einzelner Anfragen, der Gesamtdurchsatz, der Speicherverbrauch, oder die Prozessorauslastung. Im Rahmen dieser Arbeit wurden lediglich die zeitlichen Anforderungen betrachtet, also die Antwortzeit sowie der Durchsatz. Da es hierbei um einen Vergleich zweier Implementierungen geht, waren die qualitativen Resultate von größerer Bedeutung als die quantitativen. Wichtiger als die eigentlichen Werte, war also der jeweilige Vergleich der verschiedenen Werte aus den unterschiedlichen Prozessstypen.

Um diese Werte zu ermitteln kann entweder mit den internen Mitteln der Server gemessen werden, oder auf externe Werkzeuge zurückgegriffen werden. Wie bereits angesprochen, ermöglicht SoapUI beispielsweise die Messung der Antwortzeiten seiner Anfragen, und stellt die daraus ableitbaren Daten, wie minimale, maximale und durchschnittliche Antwortzeit, oder die Anzahl der Transaktionen pro Sekunde zur Verfügung. Prinzipiell könnten diese Zahlen verwendet werden, jedoch kann der gemessene Prozess damit nur als ganzes betrachtet werden. Wie lange bestimmte Abschnitte innerhalb des Prozesses benötigen, wie lange der Prozess auf die Antwort eines aufgerufenen Services warten muss und dergleichen, lässt sich mit externen Werkzeugen nur schwer oder gar nicht messen.

Der Websphere Process Server (und damit einhergehend der WESB) bietet verschiedene interne Möglichkeiten, die Performance seiner Abläufe zu überprüfen. Prinzipiell stehen drei Ansätze zur Verfügung um solche Performancemessungen intern vorzunehmen. Die Performance Monitoring Infrastructure, die Request Metrics sowie die Erzeugung von Events innerhalb der Prozesse und deren Verarbeitung mittels Audit Log oder Common Event Infrastruktur. (s. hierzu auch [TM003])

Die Performance Monitoring Infrastructure liefert statistische Daten über die verwendeten System-Ressourcen. Im Hinblick auf Prozesse lassen sich diese nur als Ganzes betrachten, bzw. wenn vom Prozess Aufrufe an externe Dienste gehen. Hierfür lassen sich dann Daten wie Anzahl der Aufrufe, Anzahl erfolgreicher Requests, durchschnittl. Responsezeiten usw. auslesen. Spezifische Informationen zu den einzelnen Aufrufen wie Start- und Endzeitpunkt, aus denen sich wiederum andere Informationen ableiten ließen, fehlen aber. Im wesentlichen liegen damit dieselben Nachteile wie bei externen Werkzeugen vor.

Bei den Request Metrics werden Daten von einzelnen Aufrufen gesammelt. Sie liefern für einen bestimmten Requesttyp, bsw. WebServices, welche Prozesse jeweils von welchem Elternprozess aufgerufen werden, grobe Informationen über die Parameter, Endzeitpunkt des Prozesses, sowie die verstrichene Zeit. Aus diesen Daten können schon genauere Information abgeleitet werden, wie lange zum Beispiel ein Prozess braucht, bis er einen bestimmten

Aufruf macht. Einzelheiten zum inneren Ablauf des Prozesses können damit aber auch nicht erfasst werden, da Daten immer nur dann vorliegen, wenn ein Aufruf stattfindet. Die Ergebnisse können entweder an ein Auswertungstool übergeben werden, das die Application Response Measurement - API implementiert, oder sie kann in ein Logfile ausgegeben werden, aus dem man dann die Daten auslesen kann.

Um Messungen an beliebigen Stellen innerhalb der Prozesse vornehmen zu können bieten sowohl die Business Flows als auch die Mediation Flows die Erzeugung von Ereignissen (Events) an. Diese Events können quasi überall innerhalb der Prozesse generiert werden, und können neben ihrem Entstehungszeitpunkt auch Informationen zu den Werten von Variablen bzw. Nachrichten innerhalb der Prozesse liefern. Aus den Business Flows heraus können solche Events im AuditLog gespeichert werden, einer Datenbank, die diese Informationen dauerhaft speichert. Alternativ können auch Events für die Common Event Infrastructure (CEI) erzeugt werden. Diese stellt eine Reihe von Schnittstellen zur Verfügung, mit der Events, die nach dem Standard der Common Base Events [CBE] erzeugt wurden, verarbeitet werden können. Diese Events können dann von der CEI weiterverteilt, bzw. in ihrer internen Datenbank gespeichert werden.

Praktisch alle Komponenten eines Business Prozesses, wie z. Bsp. Assigns oder Invokes, sind in der Lage solche Events zu feuern (bspw. Entry- und Exit-Events). Damit können die Messungspunkte quasi an jeder Stelle des Verarbeitungsablaufs gesetzt werden. Die Primitives des Mediation Flows sehen eine solche Event-Erzeugung nicht vor. Es existiert allerdings ein spezieller Mediation Primitive, der Event-Emitter, der diese Events generieren und der an jeder Stelle innerhalb des Mediation Flows eingebaut werden kann.

Die Flexibilität dieser Events hat aber auch ihren Preis. Die Eventgenerierung ist ein persistenter Vorgang und entsprechend aufwendig. Die Events der CEI sind zur Überwachung der Abläufe innerhalb von Prozessen gedacht, aber prinzipiell nicht zur Performancemessung. Die Erzeugung der Events, das Verarbeiten durch die CEI sowie das Speichern in der Datenbank benötigt Ressourcen, was sich auf das Laufzeitverhalten der Prozesse spürbar auswirken kann. Der Einfluss der Messwerkzeuge auf das Verhalten der Testobjekte ist ein generelles Problem, aber von den hier beschriebenen Methoden wirkt sich die Verwendung der CEI am deutlichsten aus. Trotzdem wurden die Events der CEI für die Messungen verwendet. Dies liegt zum einen daran, dass die Flexibilität der Messpunkte, wie oben beschrieben, nur auf diese Art zu erreichen war. Zum anderen kann diese Meßmethode für alle betrachteten Prozessarten verwendet werden. Die Events des AuditLogs stehen für den Mediation Flow nicht zur Verfügung. Die Events der CEI können jedoch in allen Prozessarten erzeugt werden und liefern auf diese Art vergleichbare Ergebnisse. Während die Verwendung der CEI die quantitativen Ergebnisse der Tests negativ beeinflusst, werden die qualitativen Aussagen der Ergebnisse dadurch nicht beeinträchtigt, da in allen Prozesstypen dasselbe Messwerkzeug verwendet wird, und durch die Flexibilität der Messpunkte auch an den jeweils einander entsprechenden Stellen innerhalb der Prozesse gemessen werden kann.

### 4.3. Testfälle und Testmethodik

Durch den Aufbau der Testszenarien sind im wesentlichen drei Parameter für den Aufruf der einzelnen Prozesstypen einstellbar. Dies ist zunächst einmal die Anzahl der Threads, die parallel den jeweiligen Prozesstyp aufrufen. Als zweites kann bei jeder Aufrufnachricht eines Prozesses die Anzahl der Aufrufe der Bankdienste angegeben werden, wie das in der Vorstellung der Testszenarios bereits erwähnt wurde. Zuletzt kann durch das Payloadelement die Nachrichtengröße beliebig angepasst werden, um die Auswirkungen großer Nachrichten auf den jeweiligen Prozesstyp zu prüfen.

Für jeden der Parameter wurden fünf Standardwerte festgelegt, die in den Testfällen verwendet werden (Tabelle 4.1).

Tabelle 4.1.: Parameter

Parameter	Minimum	Low	Medium	High	Maximum
Threads	1	20	30	40	40
Serviceaufrufe	3	10	20	30	30
Payloadgröße (kb)	10	50	200	500	500

Aus der Kombination dieser Parameter sind schließlich die verschiedenen Testfälle entworfen worden, die mit jedem Prozesstyp durchgeführt wurden (Tabelle 4.2 auf der nächsten Seite). Wie man in der Übersicht sieht, bleibt die Anzahl der Gesamtaufrufe bei jedem Testfall gleich. Der jeweilige Prozess wird also insgesamt 500 mal aufgerufen. Der Minimumtest ist ein einfacher sequentieller Aufruf des Prozesses mit drei Serviceaufrufen und minimaler Payload.

Der Maximumtest ruft die Prozesse mit 40 parallelen Threads und maximalen Parameterwerten auf. Er stellt die höchste Belastung für die Prozesse dar.

Die verbleibenden Mediumtests verwenden jeweils die Mediumwerte als Parameter, und verändern immer nur den im Namen des Testfalls angegebenen Parameter auf den entsprechenden Wert. Da die Parameter für die Testfälle Mediumtestfall-Threads-Medium, Mediumtestfall-Serviceaufrufe-Medium und Mediumtestfall-Payload-Medium identisch sind, ergeben sich 9 unterschiedliche Testfälle. Für die letztgenannten Testfälle wurde entsprechend nur jeweils ein Test pro Prozesstyp durchgeführt.

Gemessen wurden für diese Testfälle die komplette Prozessdauer, also ab Eingang der Aufrufnachricht in den Prozess bis zum Ende des Prozesses, unmittelbar bevor die Ergebnismessage zurückgegeben wurde.

Die einzelnen Testfälle wurden nacheinander für jeden Prozesstyp getestet, wobei die Vorgehensweise immer gleich war.

Vor Beginn jedes Tests wurde zunächst der Server neu gestartet. Dann wurde der jeweilige Prozess zunächst durch je 30 parallele Threads insgesamt 200 mal aufgerufen. Die übrigen Parameter hatten Minimumwerte. Dies wurde zwei mal gemacht. Schließlich wurde das Ganze mit 40 parallelen Threads wiederholt. Hierdurch wurde gewährleistet, dass die notwendigen Initialisierungsmaßnahmen bei jedem Prozesstyp bereits durchgeführt sind

Tabelle 4.2.: Testfälle

Testfall	Gesamaufrufe	Threads	Serviceaufrufe	Payload (kb)
Minimumtestfall	500	1	3	10
Mediumtestfall-Theads-Low	500	20	20	200
Mediumtestfall-Theads-Medium	500	30	20	200
Mediumtestfall-Theads-High	500	40	20	200
Mediumtestfall-Serviceaufrufe-Low	500	30	10	200
Mediumtestfall-Serviceaufrufe-Medium	500	30	20	200
Mediumtestfall-Serviceaufrufe-High	500	30	30	200
Mediumtestfall-Payload-Low	500	30	20	50
Mediumtestfall-Payload-Medium	500	30	20	200
Mediumtestfall-Payload-High	500	30	20	500
Maximumtestfall	500	40	30	500

und die Tests nicht beeinflussen. Dies ist vor allem deshalb von Bedeutung, da durch die Verwendung der CEI als Messwerkzeug nur innerhalb des Prozesses gemessen wird. Die Zeit, die der Server für das Erstellen der Instanzen benötigt, wird nicht berücksichtigt. Daher ist es wichtig, dass derlei vorbereitende Maßnahmen möglichst schon abgeschlossen sind. Anschließend wurden die dabei erzeugten Events aus der Datenbank gelöscht und der eigentliche Test gestartet. Jeder Test eines Testfalls bestand aus fünf Durchläufen. Bei jedem Durchlauf wurde der jeweilige Prozessstyp so aufgerufen, wie es der zugeordnete Testfall vorgibt. Insgesamt wurden bei jedem Test also 2500 Aufrufe des jeweiligen Prozesses durchgeführt, da bei jedem Durchlauf die vorgegebenen 500 Aufrufe gemacht werden. Durch die viermalige Wiederholung jedes Durchlaufes sollten belastbare Ergebnisse erzielt und die Auswirkungen von Ausreißerwerten, die sich extrem vom Durchschnitt abheben, abgemildert werden.

Nach Abschluss jedes Tests wurden die Ergebnisse aus der CEI-Datenbank ausgelesen und in eine Datei kopiert. Dadurch konnte die Datenbank der CEI vor jedem Test stets ganz gelöscht werden, und ihre Größe sich nicht auf die Testergebnisse auswirken.

# Testergebnisse

---

## 5.1. Minimumtestfall

Der Minimumtestfall soll einen grundlegenden Überblick über das Laufzeitverhalten der unterschiedlichen Prozesstypen geben. Zu diesem Zweck wurden sehr niedrige Werte für die drei Parameter verwendet (s. Tabelle 5.1). Falls einer der Prozesstypen spezielle Schwierigkeiten mit einem der Parameter haben sollte, bspw. mit einer hohen Payload, wurde er in diesem Testfall also nicht benachteiligt. Die Prozessaufrufe wurden jeweils nur von einem Thread durchgeführt, fanden also rein sequentiell statt. Die Serviceaufrufe umfassten genau jeden der drei Bankdienste einmal.

Tabelle 5.1.: Parameter Minimumtest

Testfall	Gesamtaufrufe	Threads	Serviceaufrufe	Payload (kb)
Minimumtestfall	500	1	3	10

### 5.1.1. Überblick über die Ergebnisse

Tabelle 5.2 auf der nächsten Seite gibt einen Überblick über die Ergebnisse des Tests. Da fünf Durchläufe bei jedem Test eines Prozesstyps durchgeführt wurden, müssen die Ergebnisse aggregiert werden, um sie geeignet darstellen zu können. Dazu wurden jeweils die fünf Werte aus den fünf Durchläufen aggregiert, die nach der zeitlichen Abfolge zueinander gehören, d. h. das erste Aggregat ist das Ergebnis der jeweils ersten Resultate der fünf Durchläufe, das zweite Aggregat das Ergebnis der fünf zweiten Resultate, usw. Als Aggregatsfunktionen wurde der Durchschnitt, das Minimum sowie das Maximum der fünf Werte verwendet. Aus den 2500 Testwerten der 5 Durchläufe ergeben sich somit 500 Durchschnitts-, 500 Minimum- und 500 Maximumwerte. Die Spalten geben für jeden

Prozesstyp den Gesamtdurchschnitt, das Minimum sowie das Maximum aller Durchläufe an. Um ein Maß für die Streuung der Durchschnittswerte angeben zu können, wurden für die 500 Durchschnittswerte die Standardabweichung  $\sigma$  berechnet. Außerdem ist die durchschnittliche Gesamtdauer der Durchläufe sowie der daraus abgeleitete Durchsatz enthalten. Alle Werte sind in Millisekunden mit Ausnahme der Gesamtdauer.

Tabelle 5.2.: Testergebnisse Minimumtest

Prozesstyp	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Microflow	54,91	31,00	469,00	13,67	38,24	13,07
Mediationflow	58,90	31,00	1500,00	49,86	41,15	12,15
Macroflow	671,39	375,00	6421,00	183,26	354,21	1,41

Den Werten der Durchschnittsdauer in Tabelle 5.2 zu Folge ist der Microflow etwas schneller als der Mediationflow. Dies relativiert sich aber, wenn man die anderen Werte mitberücksichtigt. Die minimale Dauer war bei beiden Prozesstypen sogar identisch, allerdings unterscheidet sich die maximale Dauer enorm. Wie dann zu erwarten ist auch die Standardabweichung beim Mediationflow erheblich höher als beim Microflow. Da die Aufrufe sequentiell stattfanden findet sich die höhere Durchschnittsdauer direkt in der Gesamtdauer und damit in den Transaktionen pro Sekunde wieder.

Wenig überraschend ist, dass der Macroflow mit seinen Werten deutlich über den anderen Prozesstypen liegt, in diesem Fall etwa um den Faktor 12. Obwohl Micro- und Macroflow praktisch das selbe Prozessmodell zu Grunde liegt, unterscheidet sich die Dauer beider Prozesse erheblich, da die Ausführung gänzlich anders erfolgt. Dies zeigt sich auch in der Abbildung 5.1 auf der nächsten Seite, die eine Übersicht über die 500 Durchschnittswerte aus den 5 Durchläufen bietet.

Die Skala in Abbildung 5.1 auf der nächsten Seite ist auf Grund der Werte des Macroflows für die anderen beiden Prozesstypen nicht optimal, aber hier zeigt sich bereits, dass die hohe Maximumdauer des Mediationflow offenbar auf einigen wenigen Extremwerten beruht. Deutlich extremer sind die Schwankungen beim Macroflow. Natürlich muss berücksichtigt werden, dass die Werte bei diesem viel höher liegen und die Abweichungen entsprechend größer sind. Da sich durch die Ergebnisse des Macroflows die Werte der anderen beiden Prozesstypen schlecht darstellen lassen, zeigt Abbildung 5.2 auf der nächsten Seite die Werte für den Micro- und Mediationflow noch einmal mit einer anderen Skala, um die Unterschiede besser erkennen zu lassen.

In Abbildung 5.2 auf der nächsten Seite sieht man nun deutlicher, dass der Microflow durchaus auch Schwankungen in seiner Ausführungszeit hat. Die Anzahl der einzelnen Spitzen ist sogar größer als beim Mediationflow, dessen Linie insgesamt recht gleichmäßig

## 5. Testergebnisse

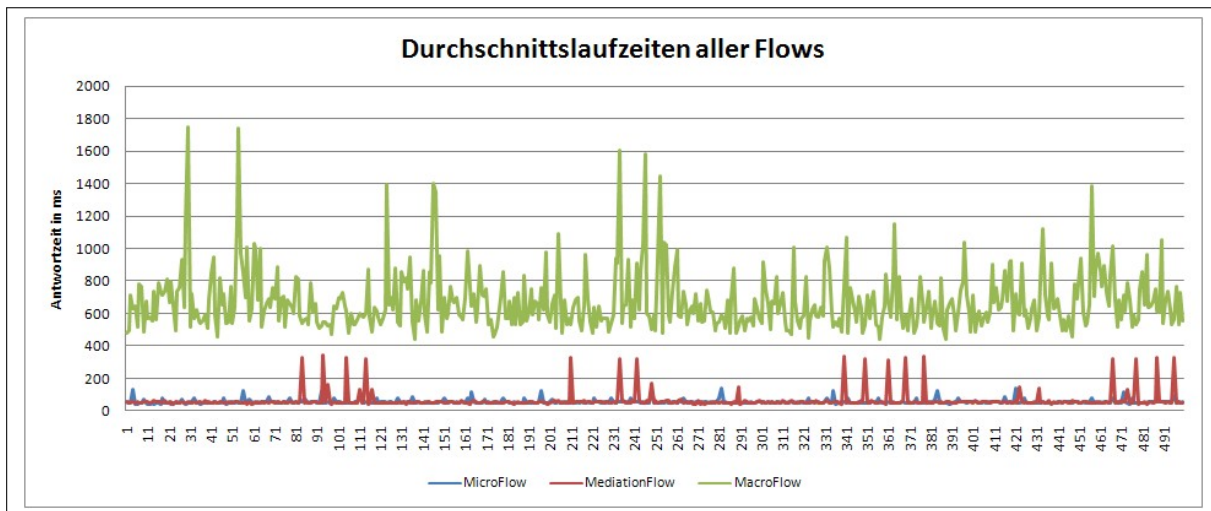


Abbildung 5.1.: Minimumtest - Antwortzeiten aller Prozesstypen

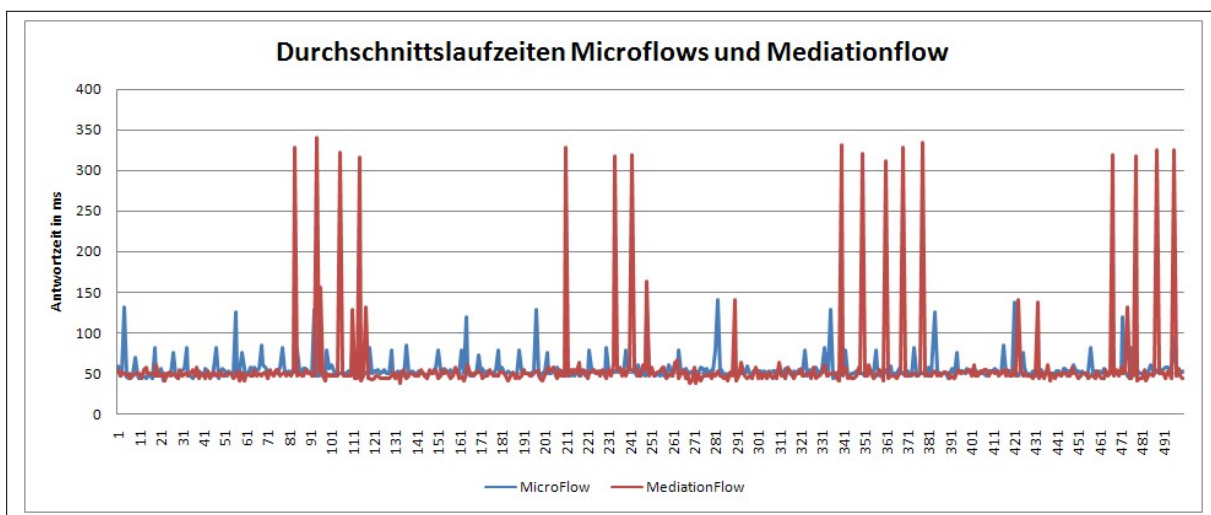


Abbildung 5.2.: Minimumtest - Antwortzeiten Micro- und Mediationflow

verläuft. Allerdings sind seine Extremwerte sehr hoch, was die hohe Standardabweichung und die Maximalwerte erklärt. Diese extremen Werte sind dann auch für das etwas schlechtere Abschneiden beim Durchschnittswert verantwortlich. Die Gründe für diese einzelnen Extremwerte liegen wahrscheinlich in der Garbage Collection. Der Mediationflow benötigt viel Speicher und ist entsprechend anfällig für Verzögerungen, die sich durch die Speicherbereinigung ergeben. Der regelmäßige Abstand zwischen den einzelnen Gruppen von Spitzen beim Mediationflow deutet ebenfalls auf die Garbage Collection hin. Abbildung 5.3 auf der nächsten Seite zeigt schließlich noch die Abweichung der 500 Durch-

schnittswerte vom Gesamtdurchschnitt. Dies soll, als Ergänzung zur oben angegebenen Standardabweichung, einen Überblick darüber geben, wie groß die Streuung der einzelnen Ergebnisse ist.

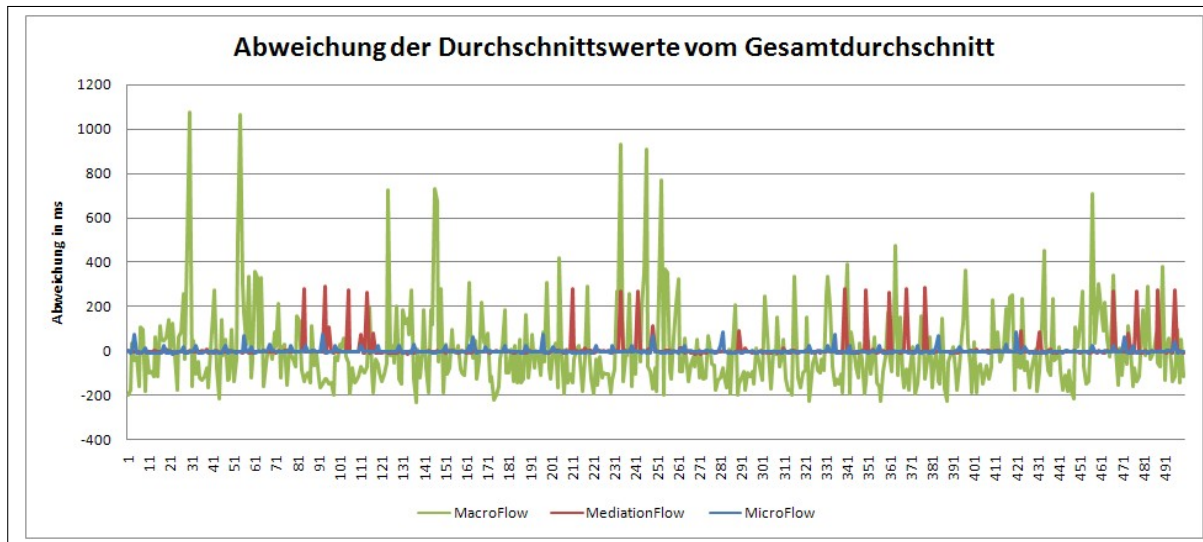


Abbildung 5.3.: Minimumtest - Abweichungen aller Prozesstypen

Interessant an der Darstellung in Abbildung 5.3 ist, dass die Werte von Micro- und Mediationflow kaum den Durchschnitt unterbieten. Die deutlichen Extrema, die man bei den beiden Prozesstypen erkennen kann, liegen allesamt über dem Schnitt. Anders ist es beim Macroflow. Auch hier liegen die starken Extrema über dem Durchschnitt, aber die durchschnittliche Laufzeit wird häufig unterboten. Daher muss der Wert der Standardabweichung beim Macroflow anders interpretiert werden als bei den anderen beiden Prozesstypen. Die Standardabweichung beim Micro- und Mediationflow geht quasi komplett auf Werte zurück, die eine Verschlechterung des Durchschnitts darstellen. Dagegen zeigt die Standardabweichung beim Macroflow lediglich, dass eine hohe Streuung vorliegt, die sowohl eine Verbesserung als auch eine Verschlechterung des Durchschnittswertes beinhaltet.

Die Resultate des Minimumtestfalles zeigen zunächst einmal die grundsätzliche Vergleichbarkeit von Micro- und Mediationflow. Selbst unter Berücksichtigung der Extremwerte liefern beide sehr ähnliche durchschnittliche Laufzeiten. Die deutlichsten Unterschiede zwischen diesen beiden Prozesstypen lagen in diesem Testfall in den Abweichungen vom Durchschnittswert. Während der Microflow relativ viele Extrema in seiner Kurve aufweist, sind diese Werte aber nur in einem relativ kleinen Wertebereich. Der Mediationflow hingegen zeigt eine vergleichsweise flache Kurve, die allerdings regelmäßig starke Spitzen aufweist, die wohl auf die Garbage Collection zurückzuführen sind.

Der Macroflow ist nur begrenzt mit den anderen beiden Prozesstypen zu vergleichen, da er

## 5. Testergebnisse

---

vollständig andere Qualities-of-Service beinhaltet. Wie zu erwarten hat er eine relativ lange Dauer, die zusätzlich deutlich schwankt, sowohl positiv wie negativ.

## 5.2. Mediumtestfall

Beim Mediumtestfall wurden die Parameter so gewählt, dass sie eher realen Anforderungen entsprechen (s. Tabelle 5.3). Während die Anzahl der Gesamtaufrufe gleich bleibt, wurden sie nun von 30 Threads durchgeführt. Dadurch fanden die Anfragen nicht mehr sequentiell, sondern parallel statt. Gleichzeitig wurde die Bankenliste auf 20 erhöht, um eine größere Zahl von externen Aufrufen durch jeden Prozess zu simulieren. Die Payload wurde ebenfalls erhöht, um die Auswirkungen, die bei einer größeren Menge an Eingabedaten entstehen, zu berücksichtigen.

Tabelle 5.3.: Parameter Mediumtest

Testfall	Gesamtaufrufe	Threads	Serviceaufrufe	Payload (kb)
Mediumtestfall	500	30	20	200

### 5.2.1. Überblick über die Ergebnisse

Die Daten für diesen Testfall wurden prinzipiell auf die selbe Weise berechnet, wie dies beim Minimumtestfall geschehen ist. Allerdings muss beachtet werden, dass durch die parallele Art der Prozessaufrufe die Abfolge der Einzeldaten nicht mehr so eindeutig ist, wie im sequentiellen Fall. Die Daten sind wie erwähnt nach ihrer zeitlichen Abfolge einander zugeordnet, und zwar nach dem Startzeitpunkt des Prozesses. Besonders zu Beginn eines Durchlaufes starten hier jedoch einige Prozesse gleichzeitig, wodurch die Zuordnung zwischen den Daten der einzelnen Durchläufe nicht mehr eindeutig sein kann. Daher sind die Daten hier zusätzlich nach ihrem Endzeitpunkt sortiert, was in den allermeisten Fällen eine eindeutige Sortierung hervorbringt.

Tabelle 5.4.: Testergebnisse Mediumtest

Prozesstyp	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Microflow	877,14	156,00	2500,00	155,39	26,86	18,61
Mediationflow	2502,56	203,00	6891,00	553,64	67,59	7,40
Macroflow	18934,13	9937,00	33750,00	1210,81	329,45	1,52

Tabelle 5.4 zeigt die Übersicht über die Messergebnisse. Die durchschnittliche Dauer ist bei allen Prozesstypen deutlich gestiegen, am wenigsten beim Microflow, wo der Unterschied etwa den Faktor 16 beträgt. Hätte man die deutlichste Steigerung wohl beim Macroflow erwartet, zeigt der Mediationflow den höchsten Anstieg, der etwa den Faktor 43 besitzt, und damit mehr als doppelt so hoch ist wie beim Mediationflow. Die Dauer des Macroflows ist hingegen nur um den Faktor 28 gestiegen. Lagen der Micro- und der Mediationflow beim

## 5. Testergebnisse

Minimumtest noch quasi gleichauf, benötigt der Mediationflow nun fast dreimal so lange wie der Microflow. Im Vergleich dazu ist der Minimumwert des Mediationflow nur moderat gestiegen. Mit einem Faktor von 6,5 ähnelt er in diesem Punkt eher der Entwicklung des Microflows mit einem Faktor von 5,0. Ähnliches gilt für die Maximumwerte, die mit einem Faktor von 5,3 (Microflow) und 4,6 ebenfalls recht nahe beieinander liegen.

Um die Auswirkungen der parallelen Aufrufe zu beurteilen ist die Gesamtdauer interessant. Beim Microflow ging sie deutlich zurück (Minimumtest 38,24 sec). Dieser Prozesstyp konnte am besten von den parallelen Aufrufen profitieren, so dass er trotz längerer Durchschnittszeit den Test früher beenden konnte. Auch der Macroflow konnte seine Gesamtlaufzeit etwas verringern (Minimumtest 354,21). Einzig der Mediationflow konnte aus dem parallelen Aufruf offenbar keine Vorteile ziehen, so dass sich seine Gesamtlaufzeit sogar deutlich erhöht hat (Minimumtest 41,15 sec.).

Abbildung 5.4 zeigt die 500 Durchschnittswerte aus den 5 Durchläufen der Tests für den Mediumtestfall. Hier wurde ein Punktdiagramm gewählt, um darstellen zu können, welche Prozesse gleichzeitig starten. Daher gibt die X-Achse die Startzeit der Prozesse an, während die Y-Achse wie gewohnt die Dauer der Prozesse wiedergibt.

Besonders beim Macroflow sieht man in Abbildung 5.4 deutlich die Gruppen von

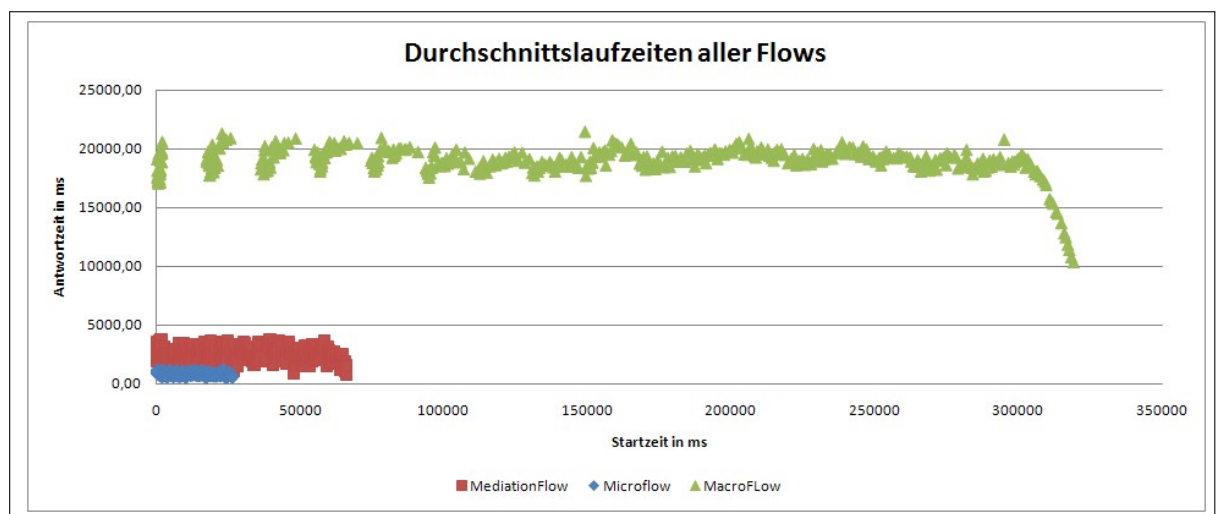


Abbildung 5.4.: Mediumtest - Antwortzeiten aller Prozesstypen

Prozessen, die gemeinsam starten. Dass sich die Punkte innerhalb der Gruppe auf unterschiedlicher Höhe befinden, zeigt, dass die Dauer der einzelnen Prozesse unterschiedlich lang ist. Entsprechend fangen die Gruppen im Laufe des Tests an, auseinanderzufallen, bis sie schließlich nicht mehr erkennbar sind. Deutlich sieht man, dass der parallele Aufruf die Prozesse bremst. Wenn gegen Ende des Tests die Zahl der parallelen Prozesse nach und nach abnimmt, sinkt bei den verbleibenden Prozessen die durchschnittliche Dauer schnell ab. Anders als im Liniendiagramm ist hier auch die unterschiedliche Gesamtdauer der

Prozesstypen ablesbar, womit der Unterschied der Laufzeiten der Prozesse noch einmal deutlich wird. Um die Ergebnisse für Micro- und Mediationflow besser darstellen zu können, zeigt Abbildung 5.5 diese Werte in einer für diese Prozesstypen geeigneteren Skala.

Während man in Abbildung 5.5 beim Microflow am Anfang die Gruppen der par-

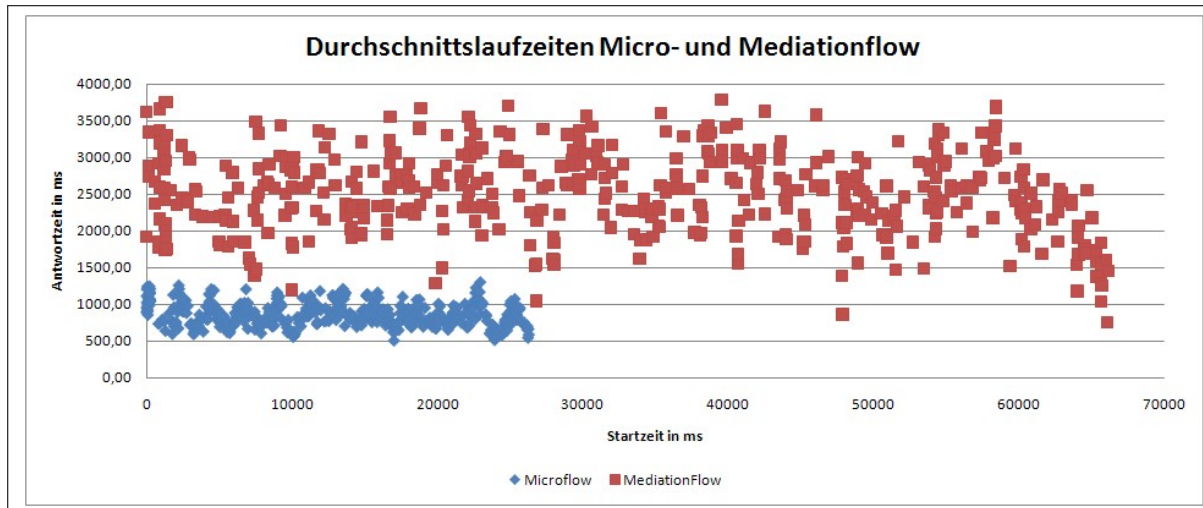


Abbildung 5.5.: Mediumtest - Antwortzeiten Micro- und Mediationflow

allel gestarteten Prozesse noch erkennen kann, lassen sich beim Mediationflow selbst zu Beginn keine einheitlichen Gruppen ausmachen. Es finden also kaum synchrone Starts statt. Beim Microflow sind diese Gruppen sogar noch nach Beginn des Test erkennbar. Auffällig ist die starke Streuung der Werte des Mediationflow.

Um einen besseren Vergleich mit den Ergebnissen des Minimumtests zu ermöglichen, sind in Abbildung 5.6 auf der nächsten Seite die Ergebnisse für den Micro- und den Mediationflow zusätzlich als Liniendiagramm dargestellt, wobei hier die Informationen über den parallelen Ablauf allerdings verloren gehen. Auch wird dort der Anschein einer gleichen Gesamtdauer erweckt, jedoch ist auf der Y-Achse nicht länger die Zeit abgebildet, sondern lediglich die Startnummer des jeweiligen Prozesses.

In Abbildung 5.6 auf der nächsten Seite sieht man noch einmal den deutlich „ruhigeren“ Verlauf des Microflow im Gegensatz zu den Schwankungen der Werte beim Mediationflow. Es kommt bei den Durchschnittswerten des Mediationflow nur in Ausnahmefällen vor, dass sie in den Wertebereich des Microflow hinunterreichen. Der Durchschnittswert des Mediationflow ergibt sich also nicht auf Grund weniger sehr hoher Werte, die den Durchschnitt nach oben ziehen, sondern ist das Ergebnis des insgesamt sehr extremen Verlaufs des Prozesstyps. Deutlicher werden die Schwankungen noch einmal in der Abbildung 5.7 auf der nächsten Seite dargestellt, die abschließend noch einen Überblick über die Abweichung der Durchschnittswerte vom Gesamtdurchschnitt gibt.

## 5. Testergebnisse

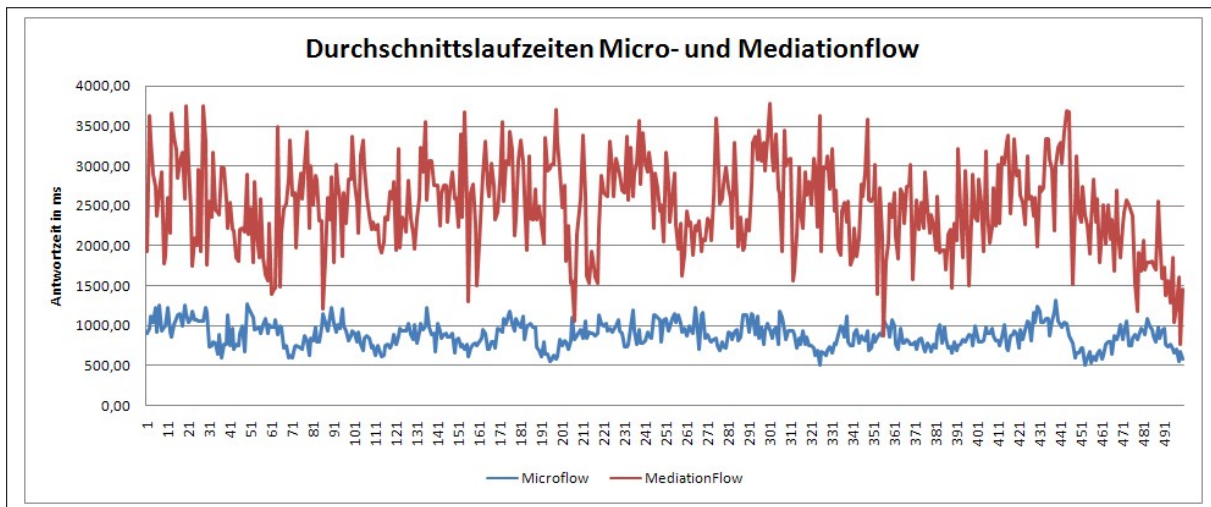


Abbildung 5.6.: Mediumtest - Antwortzeiten aller Prozesstypen

In Abbildung 5.7 haben sich die Kurven in ihrem Aussehen gegenüber dem Mini-

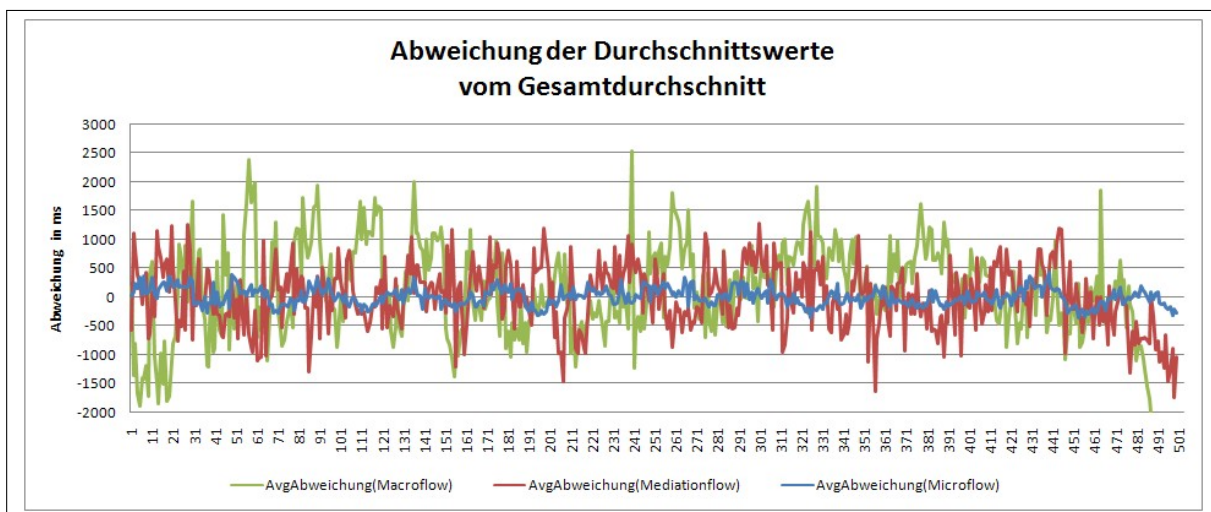


Abbildung 5.7.: Mediumtest - Abweichungen aller Prozesstypen

umtest insofern geändert, als dass es jetzt auch deutliche Abweichungen nach unten gibt, also mehrere Werte deutlich unter dem Durchschnitt liegen. War dies beim Macroflow auch schon im Minimumtest der Fall, waren dort die starken Extrema deutlich im positiven Bereich, während dies hier schon ausgewogener ist. Ähnliches gilt auch für die anderen beiden Prozesstypen, bei denen im Minimumtest die Abweichungen nur im positiven Fall (also bei einer Verschlechterung gegenüber dem Durchschnitt) deutlich erkennbar waren. Hier gibt es nun klar erkennbare Abweichungen, sowohl positiv wie negativ. Die Streuung

der Laufzeiten ist also stärker geworden, und anstatt wie im Minimumtest eine große Anzahl sehr ähnlicher Werte zu haben, die beim Micro- und Mediationflow den Durchschnittswert ergeben haben, bildet er sich hier nun aus sehr unterschiedlichen Werten. Das erklärt sich vor allem durch den parallelen Aufruf. War beim Minimumtest jeder Aufruf im Prinzip identisch zu seinem Vorgänger, und lieferte daher die gleichen Ergebnisse, beeinflussen sich hier nun die Prozesse gegenseitig, wodurch sich die Werte viel stärker verteilen.

Bei den Ergebnissen des Mediumtestfalles zeigt sich, dass alle drei Prozesstypen klar von den Änderungen der Parameter beeinträchtigt wurden. Besonders beim Mediationflow hat sich die durchschnittliche Laufzeit klar verschlechtert. Lag sie beim Minimumtest mit der des Microflow noch quasi gleich auf, beträgt sie jetzt etwa das Dreifache. Im Vergleich dazu haben sich die Minimum- und Maximumwerte bei Micro- und Mediationflow recht ähnlich entwickelt, der relative Unterschied ist im Vergleich zum Minimumtest also deutlich weniger gewachsen. Gleiches gilt für die Standardabweichung.

Der Macroflow hat sich trotz seiner speziellen Eigenschaften weniger verschlechtert als der Mediationflow. Bei ihm ist das gleichmäßige Verhalten im Prozessablauf auffällig, der sich in den klar abgegrenzten Gruppen gemeinsam gestarteter Prozesse zeigt, die sich erst im Laufe des Tests langsam auflösen. Bei ihm ist auch am deutlichsten die Beeinträchtigung durch die parallel laufenden Prozesse zu erkennen, wenn nämlich am Ende des Tests die Werte deutlich sinken. Da dies mit dem Auslaufen der Prozesse zusammenfällt, die nicht mehr neu gestartet werden, kann dies direkt auf die Beeinflussung durch die anderen Prozesse zurückgeführt werden.

Allgemein kann zur Parallelität gesagt werden, dass der Microflow diese am Besten nutzen kann, wie man an der gesunkenen Gesamtdauer erkennt. Auch der Macroflow vermag aus den parallelen Prozessen für den Durchsatz Vorteile zu ziehen. Einzig der Mediationflow kann die zusätzlichen Prozesse nicht positiv nutzen und zeigt eine deutlich erhöhte Gesamtdauer.

### 5.3. Maximumtestfall

Beim Maximumtestfall wurden alle Parameter auf die im Vorfeld festgelegten Maximalwerte gesetzt. Im Verlauf der Tests zeigte sich aber, dass beim Mediationflow Abstriche gemacht werden mussten. Beim parallelen Aufruf durch 40 Threads stürzte der Server reproduzierbar ab. Grund dafür war ein Überlauf des Java-Heaps. Dies lies sich auch nicht durch eine Vergrößerung des zugewiesenen Speichers beheben, da es bei einer größeren Java-Heap-Size zu Fragmentierungsfehlern kam. Die genaue Heapgröße ist in der Beschreibung der Servereinstellungen im Anhang einsehbar. Die anderen beiden Prozesstypen konnten den Test jedoch problemlos abschließen. Aus diesem Grund wurden ersatzweise für den Mediationflow die Ergebnisse aus den Payloadtest verwendet, der die höchsten Anforderungen der erfolgreich durchgeführten Testfälle an den Mediationflow stellt. Es muss also bei den folgenden Ergebnissen bedacht werden, dass die Anforderungen an den Mediationflow geringer waren, als an die anderen beiden Prozesstypen (s. Tabelle 5.5).

Tabelle 5.5.: Parameter Maximumtest

Testfall	Prozesstyp	Gesamtaufrufe	Threads	Serviceaufrufe	Payload (kb)
Maximumtestfall	Microflow	500	40	30	500
Maximumtestfall	Macroflow	500	40	30	500
Maximumtestfall	Mediationflow	500	30	20	500

#### 5.3.1. Überblick über die Ergebnisse

Tabelle 5.6.: Testergebnisse Maximumtest

Prozesstyp	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Microflow	1083,31	188,00	4250,00	333,76	45,46	11,00
Mediationflow	8901,30	344,00	32203,00	2010,95	258,86	1,93
Macroflow	41825,62	13562,00	62297,00	3286,91	548,66	0,91

Wie nach den Ergebnissen des Mediumtests zu erwarten war, wirken sich die hohen Parameterwerte am wenigsten auf den Microflow aus (s. Tabelle 5.6). Tatsächlich hat sich die durchschnittliche Dauer gegenüber dem Ergebnis des Mediumtests lediglich um den Faktor 1,2 erhöht, ähnlich die minimale Dauer. Selbst die maximale Dauer hat sich nicht einmal verdoppelt. Ganz anders beim Mediationflow, bei dem die durchschnittliche Dauer um mehr als das Dreifache gewachsen ist (Faktor 3,6). Während seine minimale Dauer sich in diesem Test nur wenig erhöht hat, ist die maximale Dauer deutlich gestiegen (ca. Faktor 4,7), was bereits auf eine große Streuung hinweist. Tatsächlich ist auch die Standardabweichung um den Faktor 3,6 gestiegen.

Der Macroflow hat sich ebenfalls klar verschlechtert, aber mit einer Durchschnittsdauer,

die etwas über dem Doppelten des Wertes aus dem Mediumtest liegt, hat sie sich deutlich besser entwickelt als der Mediationflow. Bei den Minimum- und Maximumwerten fällt die ähnliche Entwicklung zum Microflow auf. Das Wachstum der Minimumwerte liegt mit 1,2 beim Microflow bzw. 1,4 beim Macroflow sehr dicht beieinander, die Wachstumswerte für den Maximumwert mit 1,8 (Microflow) und 1,7 (Macroflow) sogar noch dichter. Das der Durchschnittswert beim Macroflow aber stärker als beim Microflow gewachsen ist, zeigt, dass mehr Werte des Macroflows im oberen Bereich der Streuung liegen als noch beim Mediumtest.

Ein Blick auf den Durchsatz bzw. die Gesamtdauer zeigt, dass keiner der Prozesstypen mehr Vorteile aus der gestiegenen Anzahl der parallelen Aufrufe ziehen konnte. Bei allen ging der Durchsatz deutlich zurück. Während der Durchsatz bei Micro- und Macroflow etwa um den selben Faktor zurückging (1,7), konnte der Mediationflow die zusätzliche parallele Verarbeitung am wenigsten nutzen und hat seine Gesamtdauer deutlich erhöht (Faktor 3,9 gegenüber 1,7 bei Micro- und Mediationflow). Verdeutlicht wird dies in der Abbildung 5.8, die die Durchschnittslaufzeiten als Punktdiagramm darstellt.

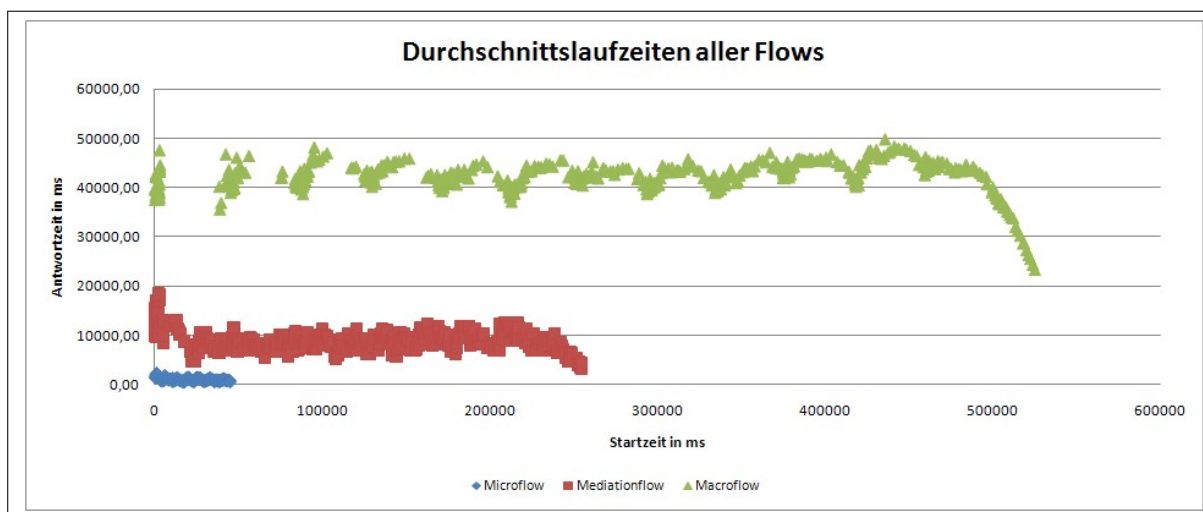


Abbildung 5.8.: Maximumtest - Antwortzeiten aller Prozesstypen

Wie schon bei Mediumtest lassen sich beim Macroflow auf Grund der langen Laufzeit der einzelnen Prozesse die gemeinsam gestarteten Prozesse am Besten in Abbildung 5.8 erkennen. Auch hier zerfallen die Gruppen recht bald, doch lassen sie sich fast bis zum Ende zumindest im Ansatz erkennen. Hierzu muss gesagt werden, dass das Zerfallen der Gruppen auch teilweise von den Durchschnittswerten verursacht wird. In den nicht aggregierten Daten der einzelnen Testläufe sind die Gruppen noch etwas kompakter und zerfallen langsamer. Auch Micro- und Mediationflow verhalten sich grundsätzlich wie im Mediumtest, wie in Abbildung 5.9 auf der nächsten Seite deutlicher zu sehen ist.

## 5. Testergebnisse

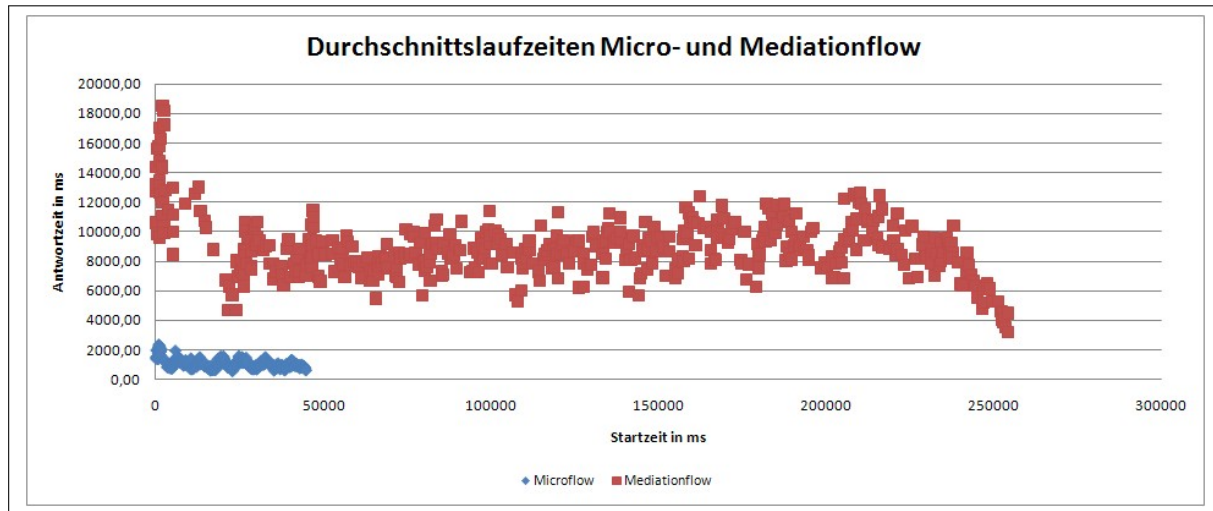


Abbildung 5.9.: Maximumtest - Antwortzeiten Micro- und Mediationflow

Wie bereits beim Mediumtest sind beim Mediationflow die Werte der ersten Gruppe in Abbildung 5.9 stark gestreut, und beinhalten auch den Maximumwert. Anschließend zerfällt die Gruppe und es lassen sich keine gemeinsam gestarteten Prozesse mehr erkennen. Die extremen Werte zu Beginn dürften wie beim Mediumtest auf das Starten vieler Prozesse zum selben Zeitpunkt zurückzuführen sein. Im weiteren Verlauf des Tests werden die Prozesse auf Grund ihrer unterschiedlichen Dauer nicht mehr gleichzeitig gestartet. Das wirkt sich offenbar deutlich auf die Dauer aus, die nach dem Beginn des Tests, wenn auch mit großer Streuung, in einem niedrigeren Bereich bleibt. Am Ende, wenn die Anzahl der parallelen Prozesse immer mehr abnimmt, geht die Dauer der zuletzt gestarteten Prozesse schließlich deutlich zurück.

Um den Verlauf der Microflow-Prozesse ebenfalls besser beurteilen zu können, sind deren Durchschnittswerte in Abbildung 5.10 auf der nächsten Seite noch einmal mit einer besseren Skala dargestellt.

Auch in Abbildung 5.10 auf der nächsten Seite sieht man, ähnlich wie beim Mediationflow, dass die höchsten Werte unmittelbar beim Start des Tests auftauchen, wenn also viele Prozesse gleichzeitig gestartet werden. Anschließend variieren die Werte sehr, und bilden ein Art Wellenmuster, in dem man, anders als beim Macroflow, keine eindeutigen Gruppen erkennen kann. Stattdessen oszilliert die Dauer der Prozesse bis zum Ende des Tests.

In der Abbildung 5.11 auf der nächsten Seite sind die Werte noch einmal in ihrer sequentiellen Abfolge als Liniendiagramm dargestellt. Besonders beim Mediationflow lassen sich die hohen Anfangswerte und die nachfolgende starke Streuung deutlich erkennen. Hier

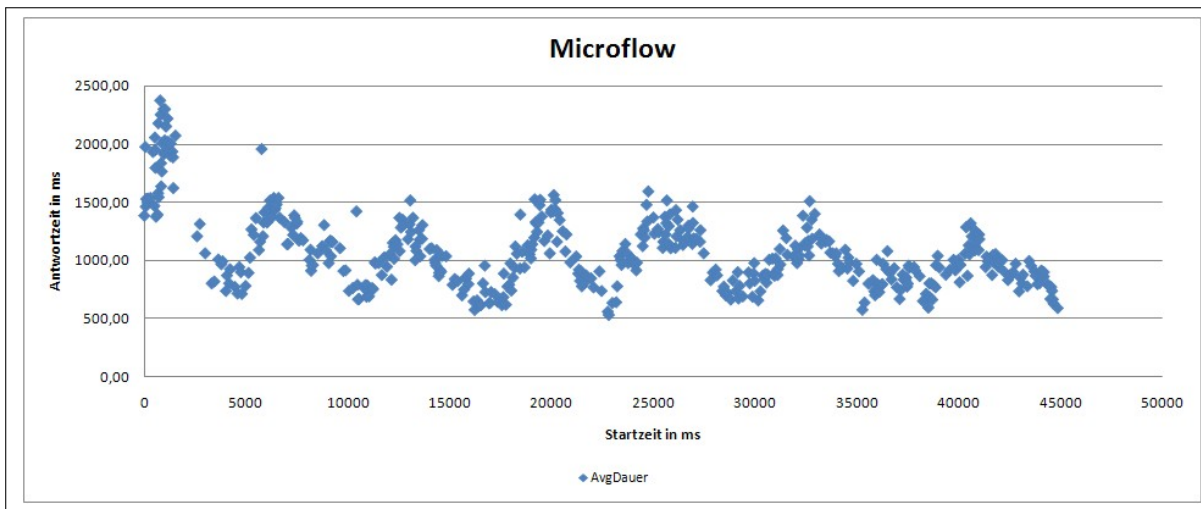


Abbildung 5.10.: Maximumtest - Antwortzeiten Microflow

wird auch deutlich, dass diese Extremwerte beim Mediationflow zu Beginn des Tests kein einfaches lineares Ansteigen darstellen. Stattdessen liegen die maximalen Werte etwa in der Mitte der Gruppe der 40 gemeinsamen Aufrufe. Für die letzten Prozesse der ersten Gruppe geht die Dauer sogar deutlich zurück. Das lässt darauf schließen, dass die hohe Dauer nicht einfach durch das parallele Ausführen der Prozesse verursacht wird, sondern konkret durch das gemeinsame Starten. Wahrscheinlich kommt es bei der Instantiierung der Prozesse zu einem gewissen Engpass, der sich deutlich auf die Gesamtdauer auswirkt.

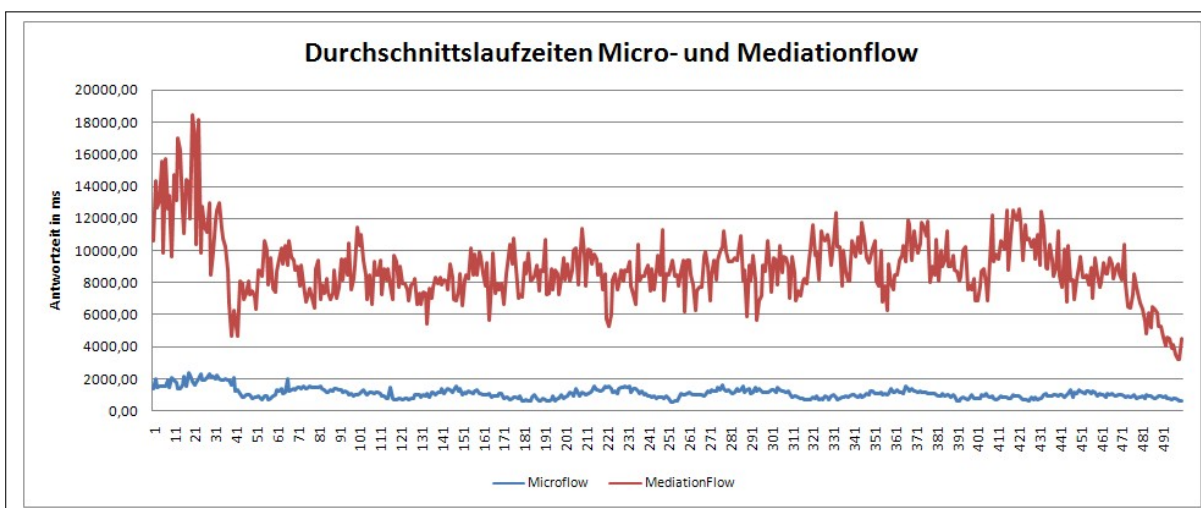


Abbildung 5.11.: Maximumtest - Antwortzeiten Mediationflow

## 5. Testergebnisse

Verdeutlicht wird dies noch einmal in Abbildung 5.12, die die Abweichungen der Durchschnittswerte vom Gesamtdurchschnitt wiedergibt. Hierbei wurde wegen des starken Abfallens der Werte des Macroflows zum Ende des Tests die y-Achse auf -2000 beschränkt. Während der Microflow sich im Vergleich zu den beiden anderen wieder sehr ruhig verhält, hat der Macroflow zu Beginn vor allem Abweichungen unterhalb des Durchschnittswertes und verhält sich damit genau andersherum als der Mediationflow.

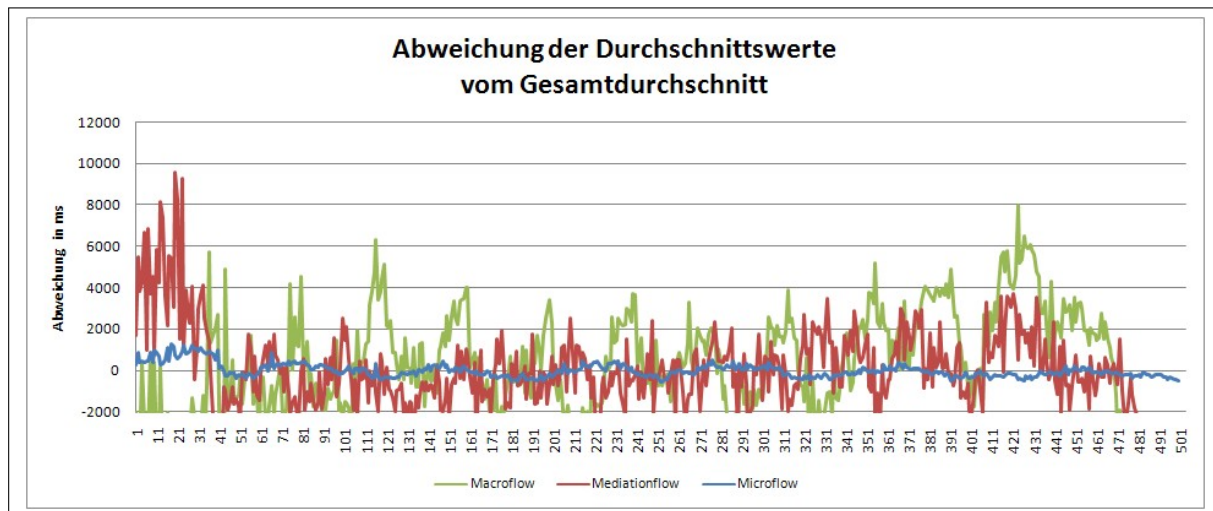


Abbildung 5.12.: Maximumtest - Abweichungen aller Prozesstypen

Der Maximumtestfall zeigt einige deutliche Unterschiede zum Mediumtestfall. So verändert der Microflow seine Durchschnittsdauer nur noch geringfügig. Allerdings ist der Unterschied in den Parametern Threads und Serviceaufrufe auch längst nicht mehr so stark wie beim Wechsel zwischen Minimum- und Mediumtestfall. Im Vergleich mit den Ergebnissen der später vorgestellten Testfälle fällt auf, dass die Durchschnittsdauer des Microflows hier interessanterweise nicht die höchste aller Testfälle ist. Beim Hightest des Serviceaufrufes wird eine höhere erreicht, allerdings ist dort die Gesamtdauer niedriger als hier (s. Kapitel 5.5 auf Seite 69). Dieses seltsame Verhalten scheint mit der Payloadgröße zusammenzuhängen und wird im Abschnitt zum Payloadtestfall näher beschrieben (Kapitel 5.6 auf Seite 77). Beim Mediationflow wächst die durchschnittliche Dauer hingegen deutlicher als beim Microflow, so dass seine Laufzeit jetzt mehr als das Achtfache des Microflows beträgt. Der Macroflow hat sich in seinen Werten ebenfalls deutlich verschlechtert, jedoch weniger stark, als dies beim Mediationflow der Fall ist.

Für die Gesamtdauer haben die zusätzlichen, parallelen Prozesse keine Vorteile mehr gebracht. Bei allen Prozesstypen ist der Durchsatz zurückgegangen. Welche der Parameter darauf den stärksten Einfluss haben, wird in den folgenden Testfällen untersucht.

## 5.4. Threadtestfälle

Bei den Threadtestfällen ging es darum, speziell die Auswirkungen paralleler Last auf die einzelnen Prozesstypen zu prüfen. Zwar wurden bei den vorangegangenen Testfällen bereits die Threadzahl erhöht, aber dort lässt sich schwer unterscheiden, welche der Veränderungen auf welchen Parameter zurückzuführen ist. Zu diesem Zweck wurden die Parameter für die Serviceaufrufe und die Payload in diesem Testfall auf die Mediumwerte gesetzt, und einzig die Anzahl der Threads, die die Aufrufnachrichten an die Prozesse schicken, verändert. Wie man den Werten in Tabelle 5.7 entnehmen kann, entspricht der Threadtestfall Medium mit seinen Werten dem bereits vorgestellten Mediumtestfall. Zum Vergleich wurden hier noch einmal Tests mit 20 Threads sowie mit 40 Threads gemacht.

Tabelle 5.7.: Parameter Threadtest

Testfall	Gesamtaufrufe	Threads	Serviceaufrufe	Payload (kb)
Threadtestfall Low	500	20	20	200
Threadtestfall Medium	500	30	20	200
Threadtestfall High	500	40	20	200

### 5.4.1. Überblick über die Ergebnisse

Die Tabellen 5.8, 5.9 auf der nächsten Seite und 5.10 auf der nächsten Seite geben einen Überblick über die Resultate der durchgeführten Testläufe. Da hier wiederum parallele Aufrufe stattfanden, wurden die Werte auf dieselbe Weise berechnet, wie dies bereits beim Mediumtestfall beschrieben wurde.

Tabelle 5.8.: Testergebnisse Threadtest Microflow

Threads	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	716,50	156,00	2422,00	147,43	27,53	18,17
Medium	877,14	156,00	2500,00	155,39	26,86	18,61
High	1045,37	171,00	4984,00	228,75	26,65	18,76

Der Microflow reagiert recht gleichmäßig auf die Erhöhungen (s. Tabelle 5.8). Beide Male wächst die durchschnittliche Dauer um den Faktor 1,2. Die minimale Dauer verändert sich bei der ersten Erhöhung gar nicht, und wächst auch bei der zweiten Erhöhung kaum. Die Maximalwerte erhöhen sich ebenfalls nur beim zweiten Mal, jedoch dort um den Faktor 2, was bereits ein starke Streuung andeutet. Die zusätzliche parallele Ausführung macht sich beim Durchsatz kaum bemerkbar, hier schlägt die höhere Prozessdauer stärker durch.

## 5. Testergebnisse

Immerhin verkürzt sich die Gesamtdauer bei beiden Erhöhungen, wenn auch nur minimal.

Tabelle 5.9.: Testergebnisse Threadtest Mediationflow

Threads	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	1365,30	172,00	4531,00	388,00	56,84	8,80
Medium	2502,56	203,00	6891,00	553,64	67,59	7,40
High	4366,96	297,00	13234,00	824,76	82,16	6,09

In Tabelle 5.9 ist beim Mediationflow zunächst einmal die höhere Durchschnittsdauer gegenüber dem Microflow auffällig. Bereits beim Lowtest liegt sie fast doppelt so hoch. Auch bei der weiteren Entwicklung steigen die Werte des Mediationflow stärker als beim Microflow. Die durchschnittliche Dauer steigt um den Faktor 1,8 und die Änderungen der minimalen und maximalen Dauer sind deutlich sichtbar. Auffallend ist, dass sich schon bei der ersten Erhöhung der Durchsatz verschlechtert. Die zusätzliche Belastung durch die parallel auszuführenden Prozesse beeinträchtigt die durchschnittliche Dauer also stärker, als dies durch die Parallelität wieder ausgeglichen werden kann. Die Folge ist eine höhere Gesamtdauer. Beim Test mit den High-Parametern steigt die durchschnittliche Dauer noch einmal um den Faktor 1,7, also etwas schwächer als bei der ersten Erhöhung. Dies macht auch Sinn, da die relative Erhöhung beim zweiten Mal geringer ausfällt. Die maximale Dauer wächst fast um das Doppelte des Medium-Wertes. Gleichzeitig sinkt der Gesamtdurchsatz noch einmal. Der Mediationflow kann also aus zunehmender Parallelität keine Vorteile ziehen.

Tabelle 5.10.: Testergebnisse Threadtest Macroflow

Threads	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	12844,91	7062,00	21921,00	753,77	334,40	1,50
Medium	18934,13	9937,00	33750,00	1209,81	329,45	1,52
High	24980,93	12625,00	39890,00	1822,43	32,59	1,53

Die Ergebnisse des Macroflows in Tabelle 5.10 sind bemerkenswert stabil. Die durchschnittliche Dauer wächst mit Faktoren von 1,5 (Low auf Medium) bzw. 1,3 (Medium auf High) nur wenig stärker als der Microflow, und sogar schwächer als bei den Erhöhungen des Mediationflow. Die Minimum- und Maximumwerte wachsen jeweils um etwa dieselben Faktoren, was darauf hindeutet, dass sich der Wertebereich der Streuung in Grenzen hält. Ähnlich wie beim Microflow steigt der Durchsatz bei den einzelnen Erhöhungen, wenn auch nur wenig. Insgesamt ist die Ähnlichkeit zur Entwicklung des Microflow deutlich, wenngleich die Werte natürlich auf einem weit höheren Niveau liegen.

Die Abbildungen 5.13 auf der nächsten Seite, 5.15 auf Seite 64 und 5.17 auf Seite 65 geben die durchschnittlichen Antwortzeiten wieder. Wie beim Mediumtestfall

werden hier Punktdiagramme verwendet, um den parallelen Beginn der Prozesse zu veranschaulichen. Die verwendeten Skalen sind jeweils für den einzelnen Prozesstyp optimiert, was bei Vergleichen beachtet werden muss. Um einen besseren Vergleich der einzelnen Laufzeiten innerhalb der Prozesstypen zu ermöglichen, sind mit den Abbildungen 5.14, 5.16 und 5.18 jeweils unter den Punktdiagrammen die Durchschnittswerte zusätzlich als Liniendiagramm dargestellt. Da hier auch wiederum der Startzeitpunkt keine Rolle spielt, muss berücksichtigt werden, dass die Tests für den Low-, Medium- und High-Fall nicht gleich lang gedauert haben, obwohl die Liniendiagramme dies auszusagen scheinen. Hier sind jedoch lediglich sequentiell die Dauer der einzelnen Aufrufe als Werte eingezeichnet. Wie aus den Punktdiagrammen hervorgeht, ist die Gesamtdauer der einzelnen Tests durchaus unterschiedlich.

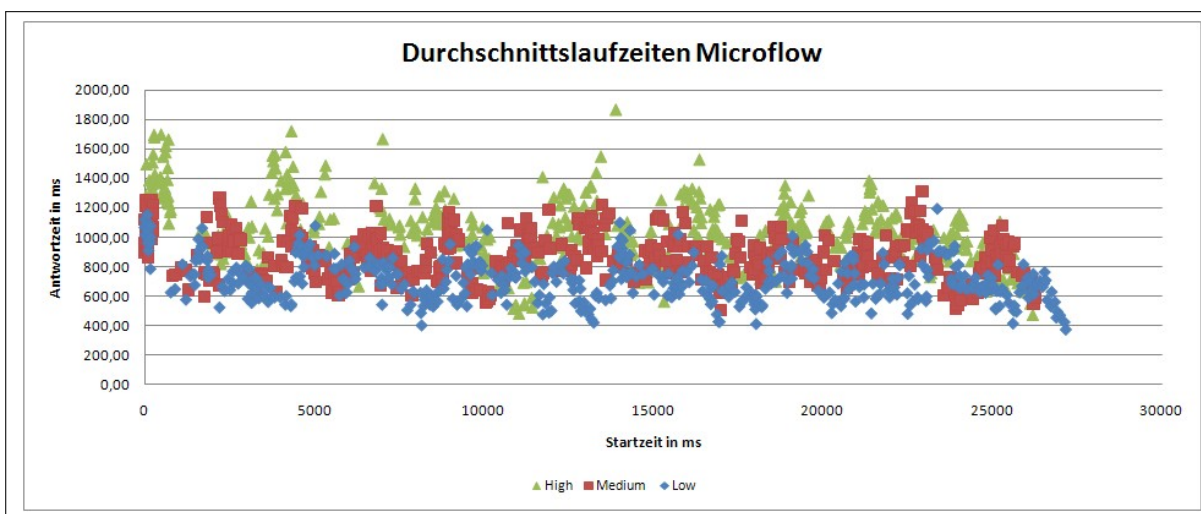


Abbildung 5.13.: Threadtest - Antwortzeiten Microflow

Das Offensichtlichste an der Darstellung 5.13 der Microflowergebnisse ist die große Nähe der Werte untereinander. Besonders die Werte aus den Low- und Mediumtests liegen teilweise sehr eng beieinander und unterscheiden sich erst in den höheren Werten des Mediumtests. Die erste Gruppe der Low-Werte liegt sogar innerhalb des Bereichs der Startwerte aus dem Mediumtest. Hier unterscheiden sie sich also vorrangig in der Streuung. Deutlicher ist der Unterschied zum Hightest, bei dem auch die Streuung stärker auffällt. Hier sieht man auch die Ähnlichkeit der Minimumwerte aus den verschiedenen Tests.

Das spiegelt sich auch im Liniendiagramm der Abbildung 5.14 auf der nächsten Seite wieder, wo die Kurven des Low- und Mediumtests praktisch im selben Bereich verlaufen und lediglich die Kurve des Hightest sich deutlicher nach oben absetzt.

In der Abbildung 5.15 auf der nächsten Seite für den Macroflow zeigen sich, im Gegensatz zum Microflow, die deutlich unterschiedlichen Ergebnisse der verschiedenen Tests. Außerdem lässt sich die unterschiedliche Gesamtlaufzeit sehr gut erkennen. Auffallend

## 5. Testergebnisse

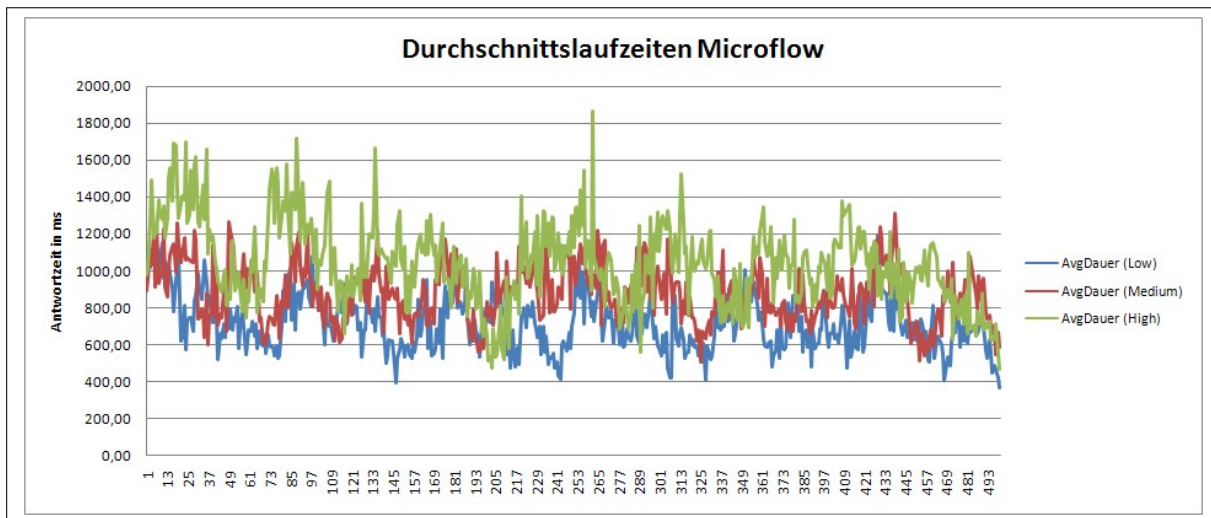


Abbildung 5.14.: Threadtest - Antwortzeiten Microflow

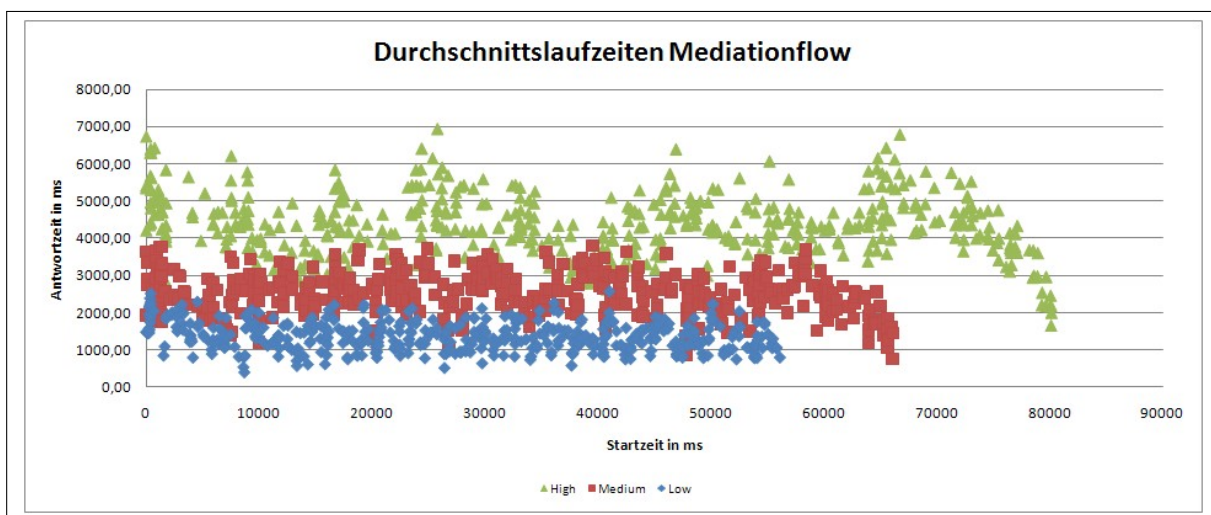


Abbildung 5.15.: Threadtest - Antwortzeiten Mediationflow

sind wiederum die hohen Werte am Anfang, die schon aus den vorhergehenden Tests bekannt sind. Bei den Werten des Highttests zeigt sich klar, dass die Streuung stark zugenommen hat. Bei den hinteren Werten des Hightests kann man wiederum das Absinken der Werte als Folge davon erkennen, dass die Gesamtzahl der parallel laufenden Prozesse zum Ende des Tests hin abnimmt.

Im Liniendiagramm 5.16 auf der nächsten Seite kann man diese Entwicklung der Highkurve ebenfalls gut beobachten. Hier sieht man auch deutlich, dass sich die Werte der Highkurve und der Mediumkurve nur an wenigen Stellen schneiden können. Trotz

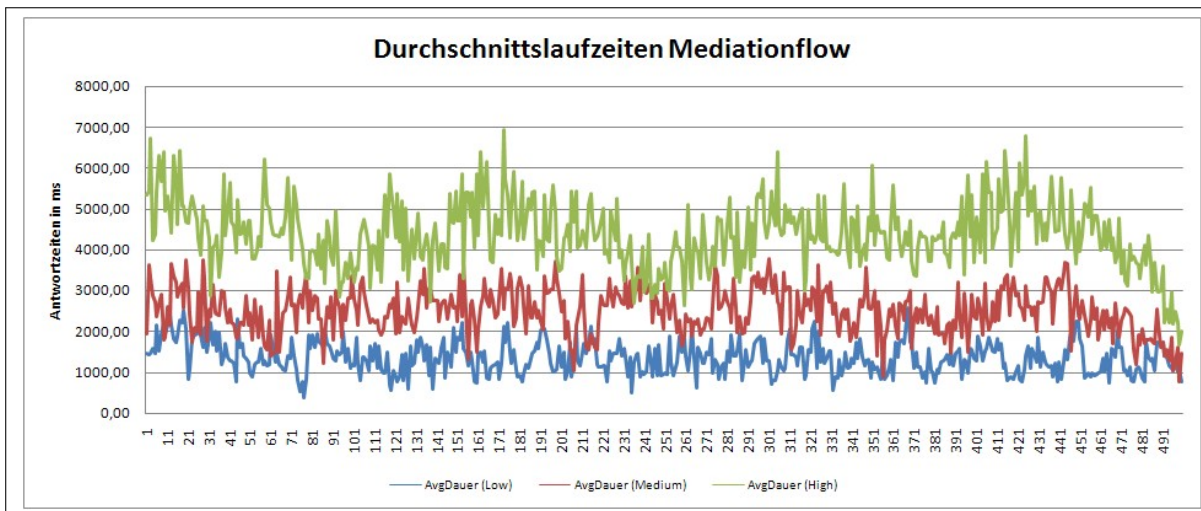


Abbildung 5.16.: Threadtest - Antwortzeiten Mediationflow

der starken Streuung des Hightests liegen also nahezu alle Werte über dem Maximum des Mediumtests. Bei den Kurven des Medium- und des Lowtests hingegen liegen die Minima der Mediumkurve recht oft in einem Bereich, der unter dem der Maxima der Lowkurve liegt. Trotz klar unterschiedlicher Durchschnittswerte können Mediationflowprozesse, die mit Lowparametern gestartet wurden, vereinzelt eine ähnliche Dauer erreichen, wie jene Prozesse, die mit Mediumparametern gestartet wurden.

Die Abbildung 5.17 zeigt den sehr stabilen Verlauf des Macroflows. Im Gegensatz

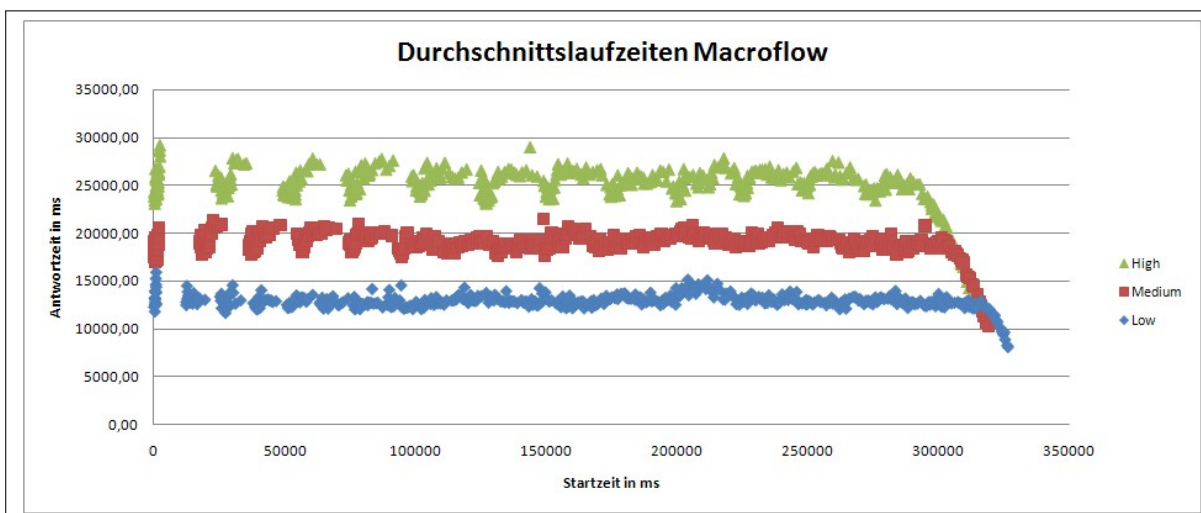


Abbildung 5.17.: Threadtest - Antwortzeiten Macroflow

## 5. Testergebnisse

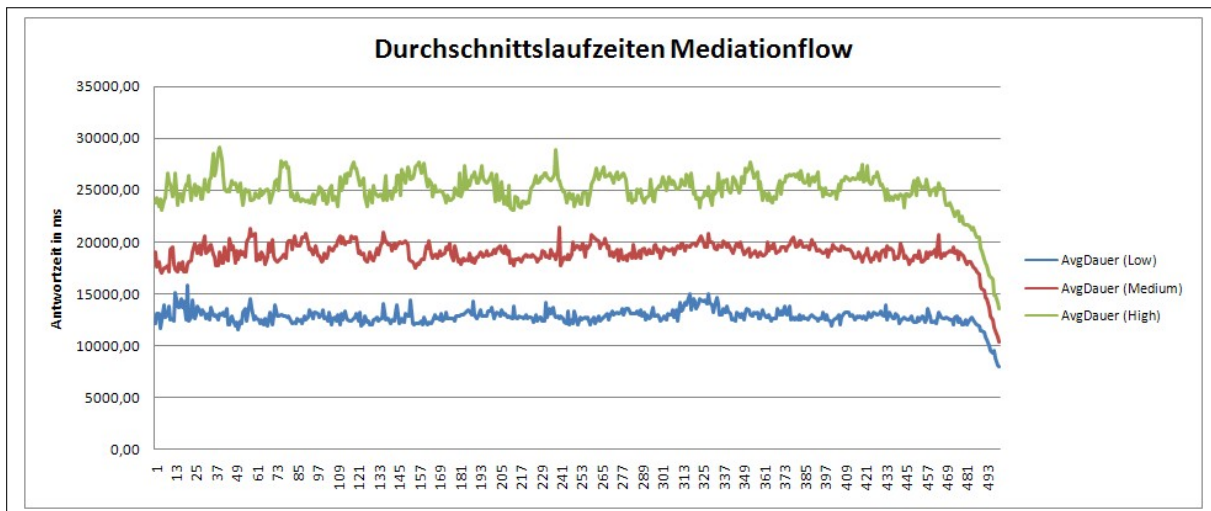


Abbildung 5.18.: Threadtest - Antwortzeiten Macroflow

zu den anderen beiden Prozesstypen hat man klar getrennte Verläufe. Deutlich kann man wieder die Gruppen der gemeinsamen Aufrufe zu Beginn der Tests erkennen. Durch die längere Dauer bei den Medium- und Hightests sind hier die Gruppen länger erkennbar, während sich beim Lowtest durch die Streuung und die kurze Laufzeit schnell keine klare Abgrenzung mehr ausmachen lässt. Man sollte beachten, dass die Gruppen im Diagramm durch die Threadzahl bestimmt werden. Im Hightest umfasst eine solche Gruppe also 40 Werte während sie im Lowtest nur 20 umfasst. Das beeinflusst natürlich dann auch die Streuung innerhalb dieser Gruppen. Deutlich sieht man am Ende den klaren Abfall der Werte bei allen drei Tests, der wieder auf das Zurückgehen der parallel laufenden Prozesse zurückzuführen ist. Am Ende der Tests lassen sich sogar die Unterschiede in der Gesamtdauer erkennen, und man sieht das die globalen Minima der Durchschnittswerte stets die zuletzt gestarteten Prozesse sind.

Abbildung 5.18 verdeutlicht noch einmal den sehr ähnlichen Verlauf der drei Tests. Die drei Kurven heben sich klar voneinander ab. Auch durch die Streuung gerät keine in den Wertebereich der anderen Kurven. Erst am Schluss nähern sie sich einander etwas an, jedoch sieht man deutlich die unterschiedlichen Minimumwerte.

Zum Abschluss der Betrachtung der Threadtestfälle zeigen die Diagramme 5.19, 5.20 und 5.21 noch die Abweichungen der einzelnen Durchschnittswerte vom Gesamtdurchschnitt, um wiederum die Streuung der Antwortzeiten zu verdeutlichen.

Bei der Abbildung 5.19 auf der nächsten Seite zeigt sich für den Microflow die bereits beschriebene Ähnlichkeit von Low- und Mediumtest. Für den Mediationflow sieht man in Abbildung 5.20 auf der nächsten Seite, dass die Streuung der Werte beim Lowtest zwar geringer ist, aber in vielen Abschnitten der des Mediumtests ähnelt. Bei beiden Prozesstypen sticht der Hightest bei den Abweichungen deutlich hervor.

Beim Macroflow liegen in Abbildung 5.21 auf Seite 68 alle drei Kurven sehr eng beieinander.

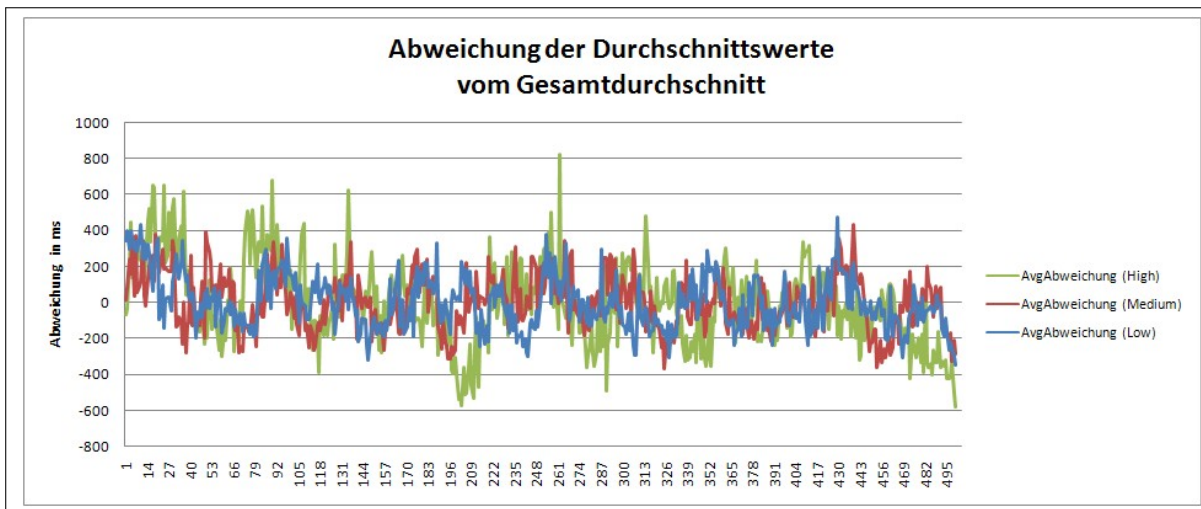


Abbildung 5.19.: Threadtest - Abweichungen Microflow

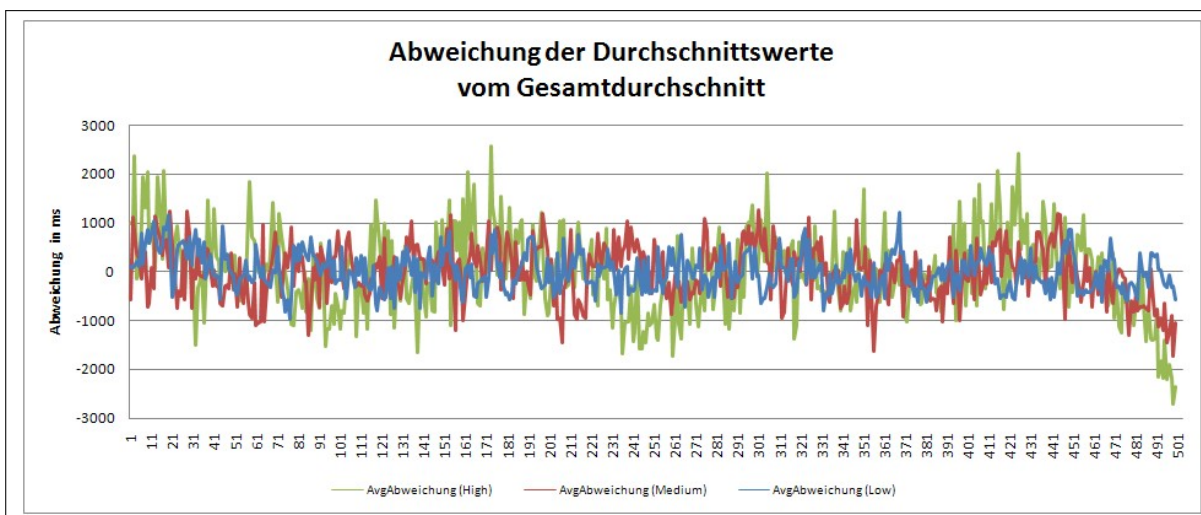


Abbildung 5.20.: Threadtest - Abweichungen Mediationflow

Bemerkenswert ist hier, dass sich alle Kurven des Macroflow größtenteils im Bereich zwischen -2000 und +2000 Millisekunden bewegen, und damit in einem Bereich, der nur doppelt so groß ist wie jener, in dem sich die Kurve des Mediationflow hauptsächlich befindet. Angesichts der viel höheren Durchschnittswerte des Macroflow unterstreicht das noch einmal, wie gering dessen Streuung tatsächlich ist.

Im Threadtestfall zeigt sich, dass die Anzahl der aufrufenden Threads auf den Microflow relativ geringe Auswirkungen zu haben scheint. Die durchschnittliche Dauer

## 5. Testergebnisse

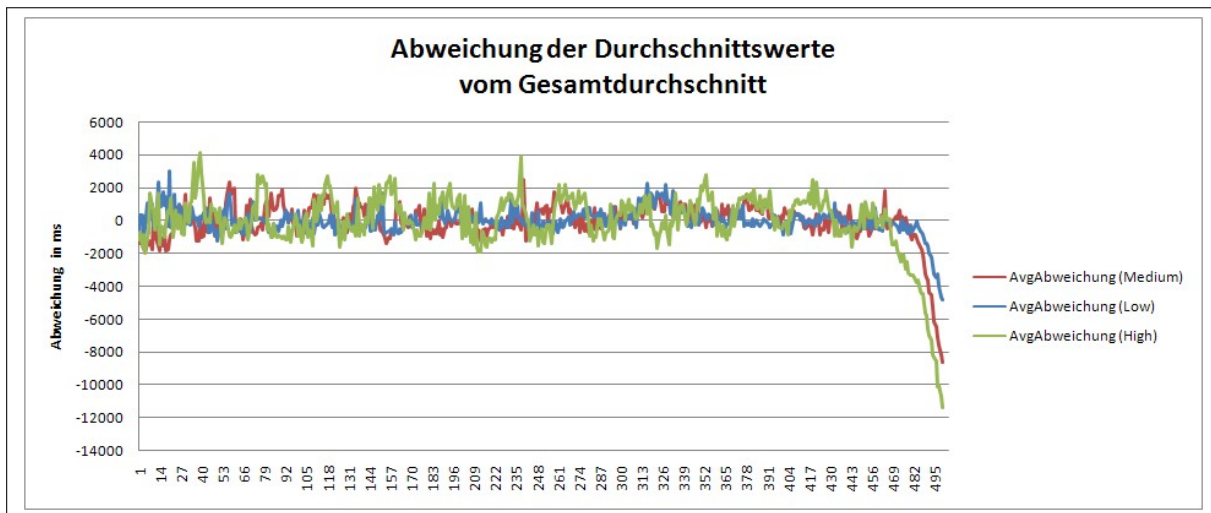


Abbildung 5.21.: Threadtest - Abweichungen Macroflow

verschlechtert sich jeweils nur sehr gering. Die parallele Verarbeitung der Anfragen kann dies jedoch kompensieren, sodass sich die Gesamtlaufzeit bei beiden Erhöhungen leicht verbessert, wenn auch nur geringfügig.

Genau gegensätzlich ist die Lage beim Mediationflow. Es steigt nicht nur die durchschnittliche Dauer deutlich, sondern gleichzeitig verschlechtert sich der Durchsatz gravierend. Bereits bei diesen Threadzahlen ist die Beeinträchtigung des einzelnen Prozesses also größer, als das durch die parallele Bearbeitung der Anfragen wieder ausgeglichen werden kann. Der Mediationflow kann mit einer großen Anzahl paralleler Anfrage also nur schlecht umgehen und der Threadparameter zeigt auch insgesamt die stärksten Auswirkungen auf das Laufzeitverhalten des Mediationflows.

Der Macroflow zeigt ein sehr ähnliches Verhalten wie der Microflow, wenn auch die Werte auf einem anderen Niveau liegen. Die Steigerung der Durchschnittsdauer liegt nur wenig über der des Microflow und auch hier sieht man die positiven Auswirkungen der zusätzlichen parallelen Prozesse auf Gesamtdauer bzw. Durchsatz. Gerade beim Macroflow werden jedoch auch die negativen Auswirkungen der zusätzlichen Threads auf die Durchschnittsdauer sichtbar, wenn die Werte der Prozesse am Ende des Tests deutlich nach unten gehen, sobald keine neuen Prozesse mehr gestartet werden.

## 5.5. Serviceaufruftestfälle

Bei den Serviceaufruftestfällen werden speziell die Auswirkungen der externen Dienste untersucht, die von den einzelnen Prozessen aufgerufen werden. Durch das Testszenario sind bereits elf Dienstaufrufe vorgegeben, die von den einzelnen Prozessen durchgeführt werden. Die Verwendung einer Recipient List bei den Bankdiensten ermöglicht es, speziell die Bankdienste beliebig häufig aufzurufen. Hierdurch lässt sich die Verwendung einer großen Anzahl von externen Diensten durch einen Prozess simulieren. Als Mittelwert für den Serviceaufruf-Parameter wurde wiederum der Wert aus dem Mediumtestfall verwendet, die Bankliste umfasst für den Medium-Fall also 20 Einträge. Im Low-Fall werden insgesamt 10, im High-Fall 30 Bankdienste aufgerufen. Die übrigen Parameter entsprechen ihrem Mediumwert (s. Tabelle 5.11).

Tabelle 5.11.: Parameter Serviceaufruftest

Testfall	Gesamtaufrufe	Threads	Serviceaufrufe	Payload (kb)
Threadtestfall Low	500	30	10	200
Threadtestfall Medium	500	30	20	200
Threadtestfall High	500	30	30	200

### 5.5.1. Überblick über die Ergebnisse

Die Resultate der Tests sind im Überblick in den Tabellen 5.12, 5.13 auf der nächsten Seite und 5.14 auf Seite 71 dargestellt. Die Werte wurden auf die selbe Weise, wie bei den Threadtestfällen berechnet.

Die Tabelle 5.12 zeigt die Ergebnisse für den Microflow. Die Veränderungen der

Tabelle 5.12.: Testergebnisse Serviceaufruftest Microflow

Serviceaufr.	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	520,42	79,00	3062,00	150,61	22,60	22,13
Medium	877,14	156,00	2500,00	155,39	26,86	18,61
High	1241,96	265,00	2953,00	206,76	31,80	15,72

durchschnittlichen Laufzeit liegen bei den Erhöhungen in diesem Testfall über denen des Threadtestfalles. Beim Medium-Test erhöht sich die durchschnittliche Laufzeit gegenüber dem Low-Test hier um den Faktor 1,7 (Threadtest: 1,2), bei der Erhöhung auf High noch einmal um den Faktor 1,4 (Threadtest 1,2). Offenbar wirkt sich die Veränderung der Serviceaufrufe auf den Microflow stärker aus als die Veränderung der Threads. Dies zeigt sich auch an der Durchschnittsdauer der Lowtests. Beim Lowtest dieses Testfalles liegt der Wert des Microflow unter dem Wert des Lowtests des Threadtestfalles. Die niedrigere

## 5. Testergebnisse

Anzahl an Serviceaufrufen (10 hier gegenüber 20 beim Lowtest des Threadtestfalles) macht sich also stärker bemerkbar als die Beeinträchtigung durch die erhöhte Threadanzahl (hier 30 gegenüber 20 beim Lowtest des Threadtestfalles). Gleiches gilt für die Durchschnittsdauer des Hightests. Hier liegt entsprechend die Durchschnittsdauer dieses Testfalles höher als die im Threadtestfall. Die Durchschnittsdauer des Microflow reagiert also auf die Veränderungen der Anzahl der Serviceaufrufe stärker als auf Veränderungen der Anzahl der Threads. Ein Grund hierfür ist, dass die Aufrufe der Services innerhalb des Prozesses sequentiell erfolgen, und sich dadurch die erhöhte Serviceanzahl unmittelbar auf die Dauer des Prozesses auswirkt.

Auch die Minimumdauer steigt bei beiden Erhöhungen in diesem Testfall deutlich an. Der extreme Maximumwert beim Lowtest, der sogar über dem des Hightests liegt, ist auf einen einzelnen Ausreißer zurückzuführen. Im Durchschnitt steigen die Maximumwerte der Durchläufe passend zur Erhöhung der Serviceaufrufe. Dies zeigt sich auch an den Werten der Standardabweichung, die sich insbesondere zwischen Low- und Mediumtest kaum unterscheiden. Wie zu erwarten war, geht bei jeder Erhöhung der Durchsatz deutlich zurück. Der zusätzliche Aufruf von Services wird hier nicht durch zusätzliche parallele Verarbeitung ausgeglichen, womit sich die höhere Durchschnittsdauer direkt in einer höheren Gesamtdauer bemerkbar macht.

Beim Mediationflow (Tabelle 5.13) ist das Verhältnis zwischen Serviceaufrufen und

Tabelle 5.13.: Testergebnisse Serviceaufruftest Mediationflow

Serviceaufr.	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	1536,49	125,00	4859,00	441,68	52,85	9,46
Medium	2502,56	203,00	6891,00	553,64	67,59	7,40
High	3282,36	329,00	12172,00	591,29	75,65	6,61

Threadanzahl genau gegensätzlich zum Microflow. Das Ansteigen der Durchschnittswerte liegt mit einem Faktor von 1,6 beim Wechsel von Low auf Medium und 1,3 beim Wechsel von Medium auf High unter den Werten des Threadtestfalles (Faktor 1,8 und 1,7). Hier wirkt sich also das Ansteigen der Serviceaufrufe weniger aus, als das Ansteigen der Threads. Auch die Durchschnittsdauer des Lowtest weist auf dieses Verhalten hin. Der Wert des Lowtests dieses Testfalles liegen unter dem Wert des Lowtests des Threadtestfalles. Die höhere Threadanzahl beim Lowtests dieses Falles beeinträchtigt die Ergebnisse also mehr, als die niedrigere Anzahl der Serviceaufrufe das ausgleichen könnte. Bei den Ergebnissen des Hightests liegt dessen Wert in diesem Testfall entsprechend auch niedriger als beim Threadtestfall. Beim Mediationflow wird die Durchschnittsdauer also stärker von der Threadanzahl beeinflusst, als von der Anzahl der Serviceaufrufe.

Die Minimumdauer verändert sich hier ebenfalls nur sehr moderat. Abgesehen vom Hightest gilt dies auch für die Maximumdauer. Das sich im Hightest die Maximumdauer plötzlich verdoppelt, dürfte wieder auf wenige Ausreißer zurückzuführen sein, wie die nur mäßig angewachsene Standardabweichung belegt. Auch beim Mediationflow geht der Durchsatz, wie zu erwarten, mit höherer Serviceaufrufanzahl klar zurück.

Die Werte des Macroflow in Tabelle 5.14 verhalten sich, was die Beziehung zwi-

Tabelle 5.14.: Testergebnisse Serviceaufruftest Macroflow

Serviceaufr.	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	11614,40	7110,00	22797,00	717,52	206,36	2,42
Medium	18934,13	9937,00	33750,00	1209,81	329,45	1,52
High	26237,56	11672,00	41593,00	1851,93	451,78	1,11

schen Thread- und Serviceaufrufanzahl angeht, wie der Microflow. Das Wachstum der Durchschnittsdauer zwischen den einzelnen Tests liegt mit einem Faktor von 1,6 (Low-Medium) und 1,4 (Medium-High) sehr dicht an den Microflowwerten und insbesondere über den entsprechenden Werten aus dem Threadtestfall (1,5 und 1,3). Auch hier unterstützt der Durchschnittswert des Lowtests diese Sichtweise. Da im Lowtest dieses Testfalles weniger Serviceaufrufe enthalten sind als beim Threadtestfall, haben wir hier eine etwas niedrigere Durchschnittsdauer (12844,91 im Threadtestfall). Entsprechend liegt der Wert des Hightests in diesem Testfall mit der maximalen Anzahl an Serviceaufrufen über dem des Threadtestfalles. Die Serviceaufrufanzahl spielt also eine größere Rolle, als die Anzahl der parallel ablaufenden Prozesse.

Dies ist vor allem deshalb interessant, weil der Macroflow die Services parallel aufrufen kann, und man daher eine weniger starke Auswirkung erwartet hätte. Da jedoch alle Aufrufe an den selben Server gingen, kann hier ein Engpass entstanden sein. Minimum- und Maximumwert wachsen ebenfalls mit einem ähnlichen Verhältnis, in den Daten sind also keine Ausreißerwerte. Der Durchsatz verringert sich hier stärker, als dies bei den anderen beiden Prozesstypen der Fall ist. Die erhöhte Anzahl der Serviceaufrufe wirkt sich hier also stärker auf die Prozesse aus, als bei den anderen beiden Prozesstypen.

Die Diagramme 5.22 auf der nächsten Seite, 5.24 auf Seite 73 und 5.26 auf Seite 74 zeigen die Durchschnittswerte der Antwortzeiten wiederum als Punktdiagramm. Jeweils daran anschließend sind auf den Diagrammen 5.23, 5.25 und 5.27 die Werte noch einmal als Kurven dargestellt, um die Vergleichbarkeit zu erleichtern.

Die Werte der Tests des Microflow (Abb. 5.22 auf der nächsten Seite) zeigen wiederum die typische Wellenform, die mit dem gruppierten Aufruf durch die 30 Threads dieses Testfalles zusammenhängt. Die Wertebereiche der verschiedenen Tests sind gut erkennbar, wenn auch die hohen Werte eines Tests in den Bereich der niedrigen Werte des nächsten Tests übergehen. Besonders beim Low- und Hightest sieht man die hohen Maxima zu Beginn der Tests, die aus dem synchronen Aufruf der 30 Prozesse zu Beginn der Tests resultieren. Die gleichmäßige Entwicklung der Gesamtdauer lässt sich an den hinteren Werten gut ablesen, wo der letzte Wert des Mediumtests nur wenig näher am entsprechenden Wert des Low- als des Hightests liegt.

Im Liniendiagramm ( 5.23 auf der nächsten Seite) lässt sich noch einmal erkennen, wie nah die Extremwerte der verschiedenen Tests beieinander liegen.

In Abbildung 5.26 auf Seite 74 ist zunächst einmal die starke Streuung der Werte

## 5. Testergebnisse

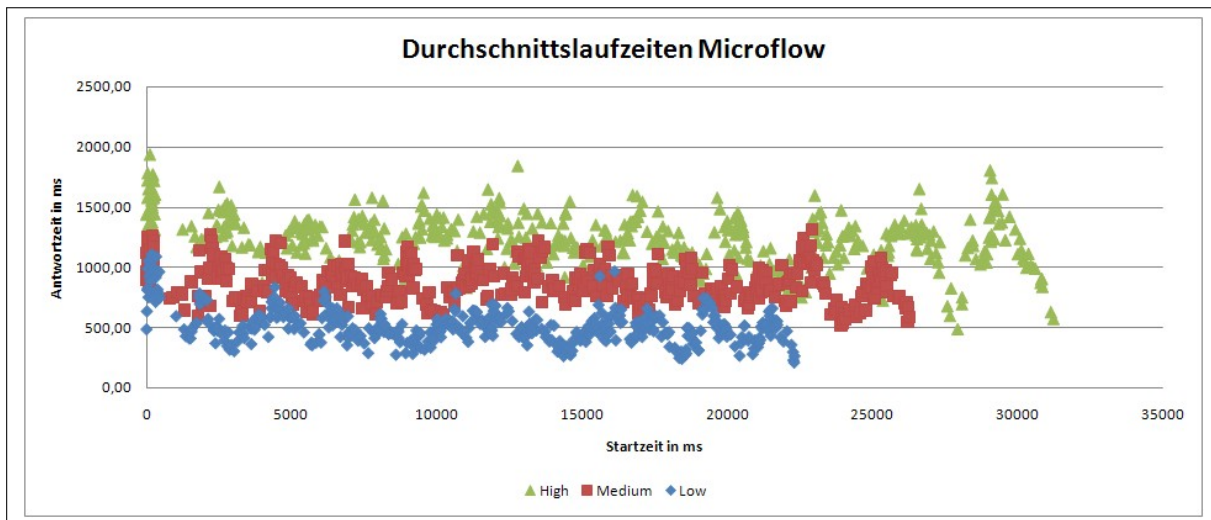


Abbildung 5.22.: Serviceaufruftest - Antwortzeiten Microflow

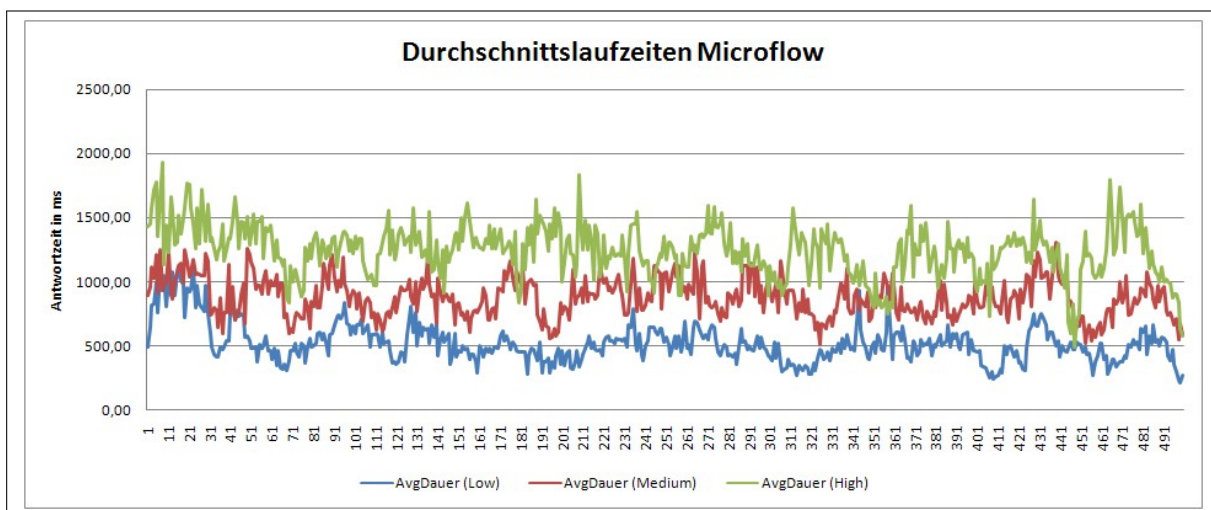


Abbildung 5.23.: Serviceaufruftest - Antwortzeiten Microflow

des Mediationflows auffällig. Obwohl die Durchschnittsdauer des Lowtests beinahe 1000 ms unter der des Mediumtests liegt, gibt es keine klare Trennlinie zwischen den Werten beider Prozesse. Die Verteilung der Minima und Maxima konzentriert sich nicht auf einen bestimmten Zeitabschnitt der Tests, wengleich es die klaren, lokalen Maxima zu Beginn und Minima am Ende der Tests gibt. Deutlich lässt sich wieder die unterschiedliche Gesamtdauer ablesen.

Im Liniendiagramm (Abb. 5.25 auf der nächsten Seite) zeigt sich die Streuung noch einmal deutlicher. Man erkennt, dass sogar vereinzelt Werte des Hightests in Wertebereichen liegen,

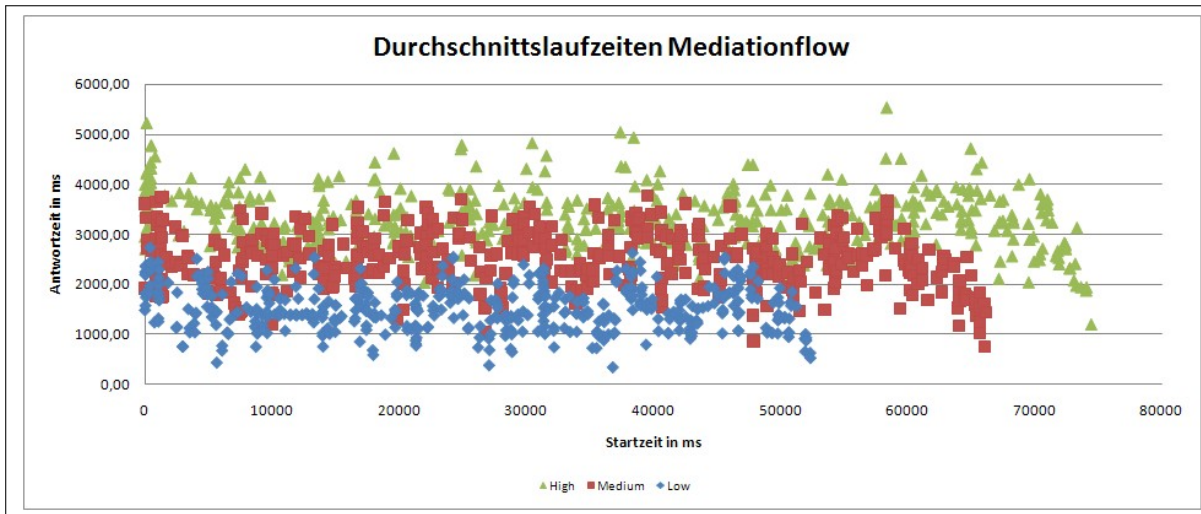


Abbildung 5.24.: Serviceaufruftest - Antwortzeiten Mediationflow

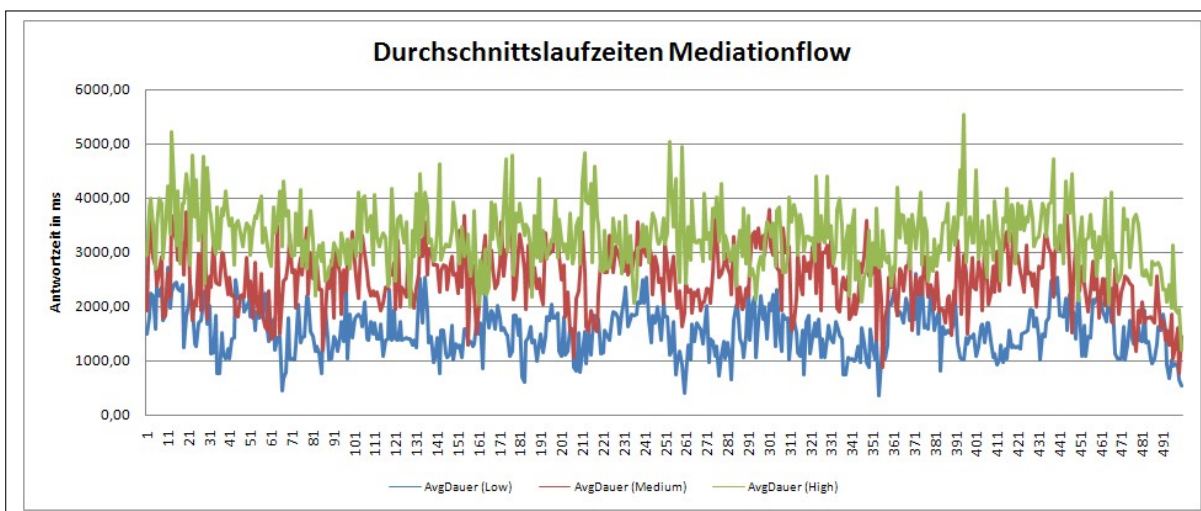


Abbildung 5.25.: Serviceaufruftest - Antwortzeiten Mediationflow

in denen auch Maximumwerte des Lowtests sind.

Der Macroflow (Abb. 5.26 auf der nächsten Seite ) zeigt wiederum das gewohnte Bild. Am Anfang der Tests die klar erkennbaren Gruppen, die sich allmählich auflösen, und am Ende der auffällige Abfall der Werte. Der recht gleichmäßige Anstieg bei den Erhöhungen ist ebenfalls gut erkennbar. Die Streuung ist für diese hohen Werte relativ gering, entsprechend sind die einzelnen Wertebereiche der Tests klar getrennt. Auch Ausreißerwerte sind kaum vorhanden.

## 5. Testergebnisse

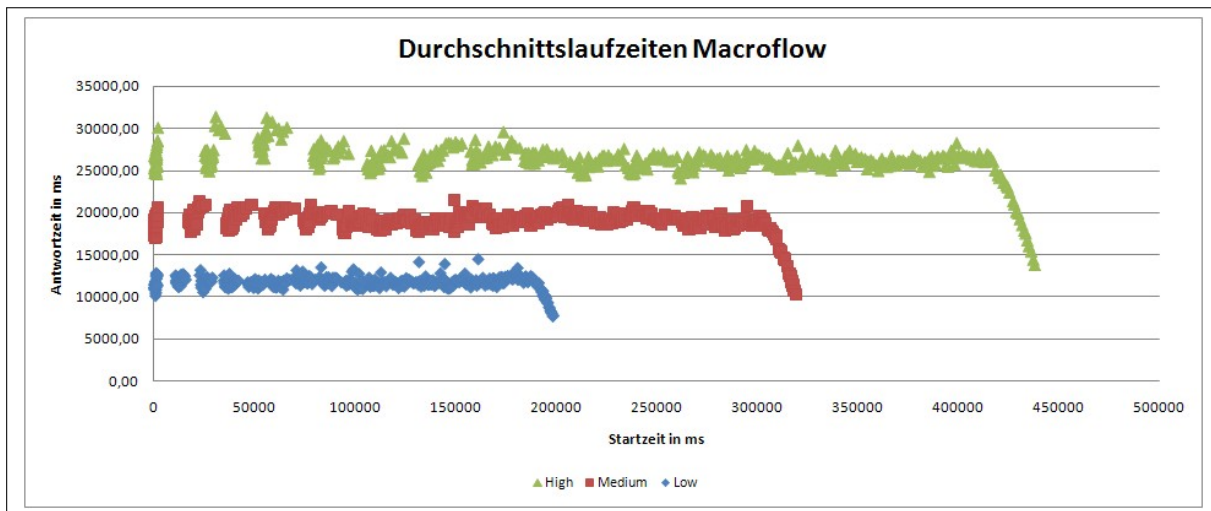


Abbildung 5.26.: Serviceaufruftest - Antwortzeiten Macroflow

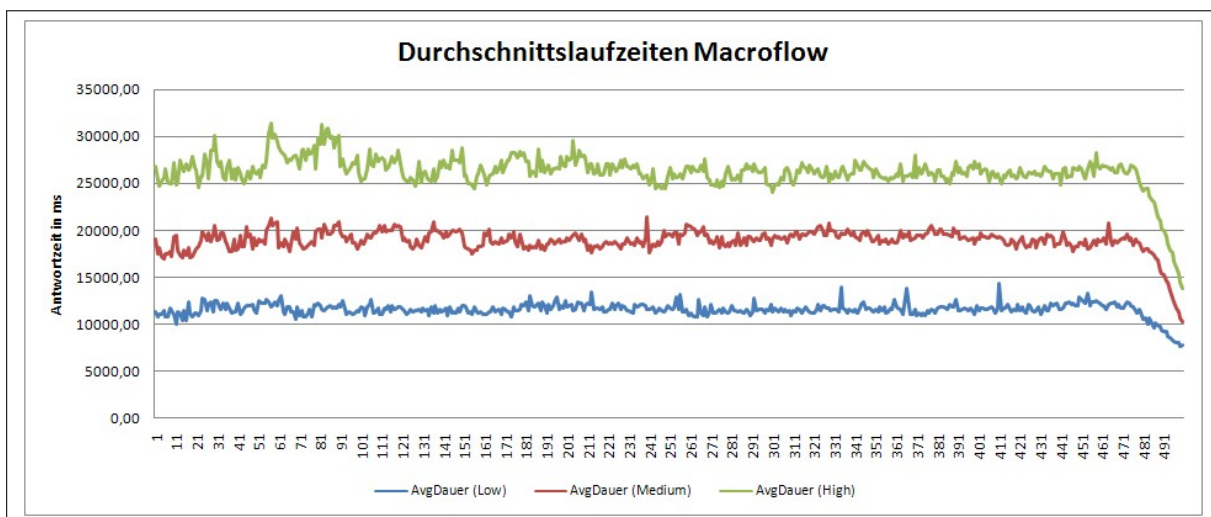


Abbildung 5.27.: Serviceaufruftest - Antwortzeiten Macroflow

Im Liniendiagramm (Abb. 5.27) sieht man vor allem beim Hightest genauer, dass die stärkste Streuung der Werte hauptsächlich am Anfang des Tests stattfindet. Der Verlauf wird gegen Ende eher ruhiger, bis die Werte schließlich zum Ende des Tests hin einbrechen.

Zur Ergänzung der Testdaten wird auf den Abbildungen 5.28, 5.29 und 5.30 wieder die Streuung der Durchschnittswerte für die einzelnen Prozesstypen dargestellt.

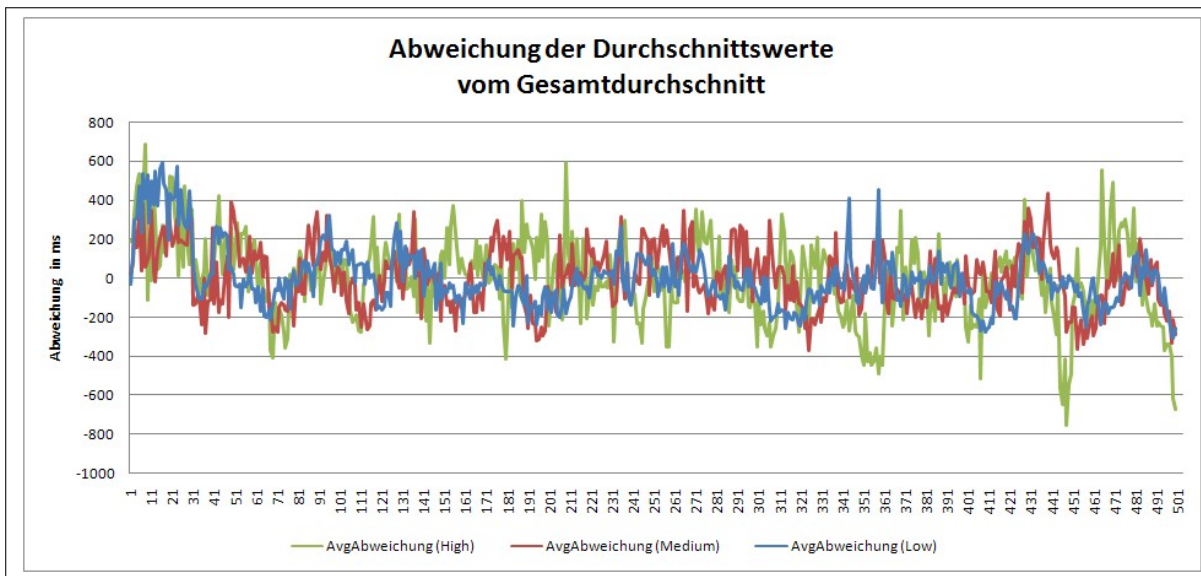


Abbildung 5.28.: Serviceaufruftest - Abweichungen Microflow

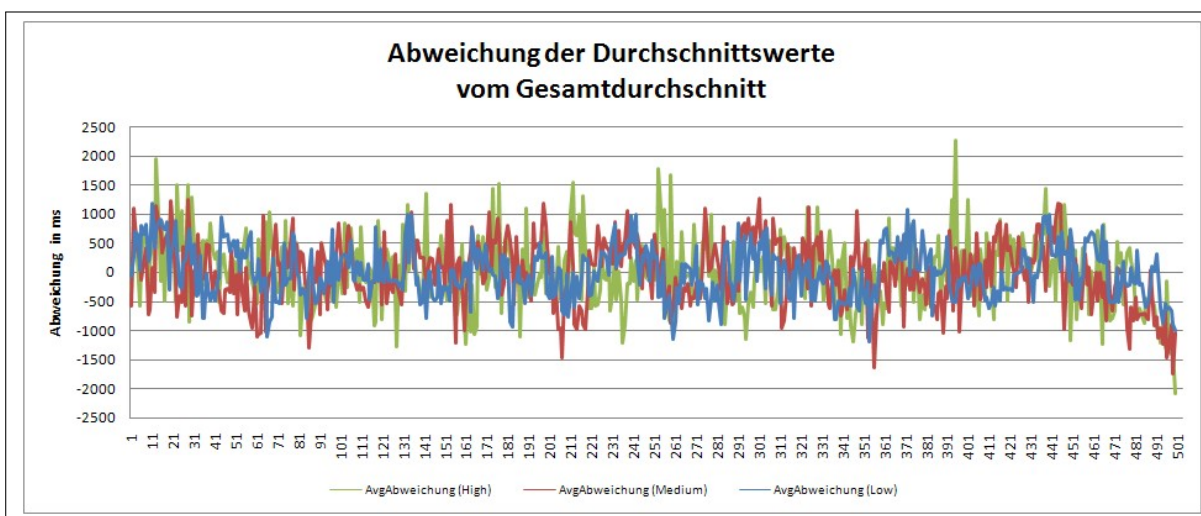


Abbildung 5.29.: Serviceaufruftest - Abweichungen Mediationflow

Beim Microflow erkennt man vor allem die sehr ähnliche Abweichung bei den unterschiedlichen Tests. Gerade am Anfang sieht man, dass die Abweichungen für den Lowtest in einem ähnlichen Bereich sind wie die des Hightests. Deutlich hervor treten die Abweichungen des Hightests gegenüber denen der anderen Prozesstypen erst im weiteren Verlauf des Tests. Auch beim Mediationflow bewegen sich die Abweichungen in sehr ähnlichen Wertebereichen. Während es beim Mediationtest vor allem einige starke Abweichungen unter dem Gesamtdurchschnitt gibt, sind beim Hightest die starken Extreme der Abweichungen über

## 5. Testergebnisse

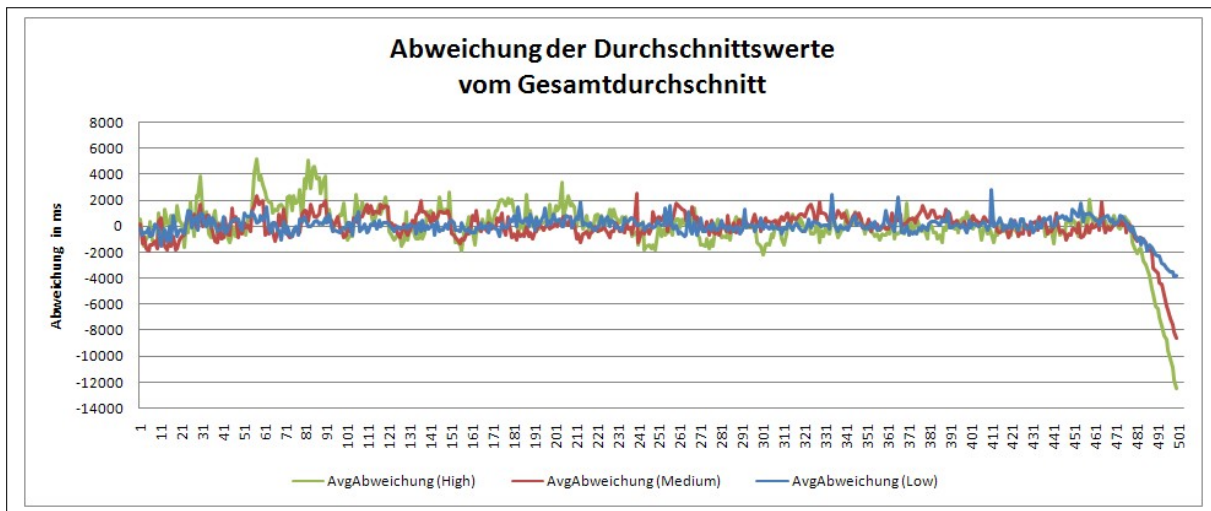


Abbildung 5.30.: Serviceaufruftest - Abweichungen Macroflow

dem Durchschnitt.

Beim Macroflow stechen am Auffälligsten die Abweichungen des Hightests hervor, bei dem teilweise mehrere Werte hintereinander deutlich über dem Gesamtdurchschnitt liegen. Klar sind natürlich die Abweichungen am Schluss zu erkennen, wenn die Durchschnittsdauer der Prozesse nach unten geht.

Der Serviceaufruftestfall zeigt, dass für den Microflow die Anzahl der Serviceaufrufe die wichtigste Rolle in Bezug auf sein Laufzeitverhalten spielt. Die Erhöhung der Durchschnittsdauer liegt klar über der, die beim Threadtest erzielt wurde. Da die Serviceaufrufe beim Microflow nur sequentiell erfolgen können, muss dies unmittelbar Auswirkungen auf die Durchschnittsdauer haben. In diesem Fall wirkt sich das wiederum direkt auf die Gesamtdauer aus, da keine Vorteile aus zusätzlicher Parallelität gezogen werden können.

Der Mediationflow zeigt ein sehr ähnliches Verhalten wie der Microflow. Die Erhöhung der Durchschnittswerte beider Prozesstypen liegt sehr dicht beieinander. Da sie auf einer gemeinsamen Architektur beruhen und die Serviceaufrufe ebenfalls sequentiell erfolgen ist dies auch wenig überraschend. Der Hauptunterschied besteht darin, dass für den Mediationflow die Anzahl der aufrufenden Threads eine größere Rolle spielt, da sie sich stärker auf seine Laufzeit auswirken, als dies bei den Serviceaufrufen der Fall ist.

Der Macroflow unterscheidet sich ebenfalls nur wenig vom Verhalten des Microflow. Die Steigerungswerte liegen auch sehr dicht aneinander, allerdings hätte man hier ein anderes Verhalten erwartet. Der Macroflow bietet die Möglichkeit, die Serviceaufrufe parallel durchzuführen, was in diesem Fall jedoch keine Verbesserung gebracht hat. Ein Grund hierfür könnte sein, dass der Tomcat-Server durch die parallelen Anfragen die gleichzeitig von parallelen Prozessen kamen, überfordert war, und sich daher keine Vorteile zeigen konnten.

## 5.6. Payloadtestfälle

Die Payloadtestfälle sollen schließlich die Auswirkungen großer Nachrichten auf die Dauer der Prozesse untersuchen. Dafür besitzt die Aufrufnachricht der drei Prozesstypen ein spezielles Element, welches die Payload enthält. Diese wird im Prozess zunächst nicht verwendet, sondern erst im letzten Teil des Prozesses einmal komplett kopiert. Die Mediumgröße, die bei den anderen Testfällen auch Verwendung fand, beträgt 200 kb. Als Minimum wurde 50 kb und als Maximum 500 gewählt (s. Tabelle 5.15).

Tabelle 5.15.: Parameter Payloadtest

Testfall	Gesamtaufrufe	Threads	Serviceaufrufe	Payload (kb)
Low	500	30	20	50
Medium	500	30	20	200
High	500	30	20	500

### 5.6.1. Überblick über die Ergebnisse

Die Tabellen 5.16, 5.17 auf der nächsten Seite und 5.18 auf der nächsten Seite geben wie gewohnt einen Überblick über die Testergebnisse.

Die Daten des Microflow in Tabelle 5.16 erscheinen zunächst sehr ungewöhnlich.

Tabelle 5.16.: Testergebnisse Payloadtest Microflow

Payload	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	1031,94	172,00	4640,00	193,98	20,32	24,60
Medium	877,14	156,00	2500,00	155,39	26,86	18,61
High	652,84	141,00	3656,00	161,47	41,59	12,02

Bei steigender Payloadbelastung nimmt die durchschnittliche Dauer ab, gleichzeitig steigt jedoch auch die Gesamtdauer und der damit einhergehende Durchsatz sinkt. Das klingt paradox und scheint zunächst einmal auf eine Vertauschung von Daten oder etwas ähnliches hinzuweisen. Jedoch wurden die Testfälle wiederholt und die Ergebnisse bestätigt. Tatsächlich sinkt die durchschnittliche Dauer mit wachsender Payload, und die Gesamtdauer steigt trotzdem. Der Widerspruch zwischen sinkender Durchschnittsdauer eines Prozesses und steigender Gesamtdauer lässt nur den Schluss zu, dass weniger Prozesse parallel ablaufen. Offenbar behindert die steigende Nachrichtengröße den Server beim Instantiieren paralleler Prozesse, was die steigende Gesamtdauer erklärt. Wie bereits gezeigt wirken sich parallel ablaufende Prozesse negativ auf die Durchschnittsdauer der Microflowprozesse aus. Da nun weniger Prozesse parallel aktiv sind, nimmt dadurch die Dauer der einzelnen Prozesse ab, was zu den obigen Ergebnissen führt.

## 5. Testergebnisse

Entsprechend sinkt analog zur Durchschnittsdauer die Minimumdauer. Auch die Maximumdauer nimmt im Durchschnitt ab, was sich jedoch an den Werten in der Tabelle, die jeweils das globale Maximum zeigen, nicht ablesen lässt.

Die Mediationflowdaten in Tabelle 5.17 sind weniger ungewöhnlich. Wie zu erwar-

Tabelle 5.17.: Testergebnisse Payloadtest Mediationflow

Payload	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	2119,73	219,00	12375,00	532,41	37,60	13,30
Medium	2502,56	203,00	6891,00	553,64	67,59	7,40
High	8901,30	344,00	32203,00	2010,95	258,86	1,93

ten, steigt bei der Erhöhung der Werte die Durchschnittszeit und die Gesamtlaufzeit. Wie sich hier die Erhöhung des Payloadparameter im Vergleich zu dem anderer Parameterdaten verhält, ist nicht eindeutig zu bestimmen. Obwohl der Anstieg der Parameterwerte mit 50 kb auf 200 kb den höchsten Anstieg eines Low- auf einen Mediumparameter bedeutet (Faktor 4 gegenüber Faktor 1,5 bei den Threads und Faktor 2 bei den Serviceaufrufen) ist das Wachstum der Durchschnittsdauer mit einem Faktor von 1,2 recht niedrig. Bei der zweiten Erhöhung ist der Faktor von 3,6 allerdings recht deutlich, auch wenn man die unterschiedlichen Relationen zwischen den Erhöhungen der Parameterwerte berücksichtigt. Offenbar kann der Mediationflow bis zu einem gewissen Grad eine höhere Payload gut verkraften, wird dann aber stark von ihr beeinträchtigt. So liegen auch beim Low- und Mediumtest die Minimumwerte sehr dicht beieinander. Der Mediumtest weist sogar ein niedrigeres globales Minimum auf als der Lowtest. Die durchschnittliche Minimumdauer der fünf Durchläufe wächst aber bei den Tests analog zur Erhöhung des Payloadparameter, so dass es sich hier wiederum um einen vereinzelt Ausreißer handelt. Gleiches gilt für den Maximumwert, der beim Lowtest ebenfalls weit vom Durchschnitt der Maximumwerte der fünf Durchläufe entfernt ist. Da auch hier wieder die Anzahl der parallel arbeitenden Threads gleich bleibt, erhöht sich mit der Durchschnittsdauer auch die Gesamtdauer. Bemerkenswert ist, dass sich zwischen Low- und Mediumtest die Gesamtdauer deutlich stärker wächst, als die durchschnittliche Dauer. Offenbar kommt es außerhalb des reinen Prozessablaufes zu Verzögerungen, die die Gesamtlaufzeit des Prozesses und damit die Prozessdauer beeinflussen.

Bei den Werten des Macroflow in Tabelle 5.18 tauchen die Probleme des Microflows nicht

Tabelle 5.18.: Testergebnisse Payloadtest Macroflow

Payload	Durch.-Dauer	Min.-Dauer	Max.-Dauer	$\sigma$ Durch.-Dauer	Ges.-dauer (sec)	tps
Low	17783,06	7625,00	28860,00	1215,26	305,69	1,64
Medium	18934,13	9937,00	33750,00	1209,81	329,45	1,52
High	23688,21	11547,00	50110,00	1974,36	415,35	1,20

auf. Wie man erwarten würde, steigt analog zur Payload die durchschnittliche Dauer und

die Gesamtdauer. Im Vergleich zu den Ergebnissen des Serviceaufruftests stellt man fest, dass sich die Payload weniger auf die Ergebnisse auswirkt als die Anzahl der Serviceaufrufe. Die Wachstumsfaktoren für die Durchschnittswerte liegen bei 1,1 (Low-Medium) und 1,3 (Medium-High) und damit höchstens gleichhoch wie die Faktoren aller anderen Tests, obwohl der Sprung bei den Parameterwerten in diesem Testfall viel stärker ist. Gleiches lässt sich wieder an den Werten des Lowtests ablesen. Trotz niedrigerer Payload als im Lowtest des Serviceaufruftestfalles ist die Durchschnittsdauer beim Lowtest dieses Testfalles höher, was auf die höheren Serviceaufrufe zurückzuführen ist. Das trifft auch beim Vergleich mit dem Threadtestfall zu, bei dem die Werte analog sind. Die Payload spielt für den Macroflow also auch eine weniger wichtige Rolle als die Anzahl der Threads.

Die Ergebnisse von Minimum- und Maximumdauer verhalten sich ähnlich wie die Ergebnisse der Durchschnittsdauer. Die Standardabweichung ist beim Low- und Mediumtest etwa gleich und steigt erst beim Hightest deutlich an. Da sich die Anzahl der Threads nicht verändert, sinkt in Folge der höheren Durchschnittsdauer der Gesamtdurchsatz.

Die Abbildungen 5.31, 5.33 auf der nächsten Seite und 5.35 auf Seite 82 zeigen wieder die Durchschnittswerte als Punktdiagramme, jeweils gefolgt von den zugehörigen Liniendiagrammen in den Abbildungen 5.32 auf der nächsten Seite, 5.34 auf Seite 81 und 5.36 auf Seite 82.

In Abbildung 5.31 zeigt sich die ungewöhnliche Situation, dass sich die Werte des

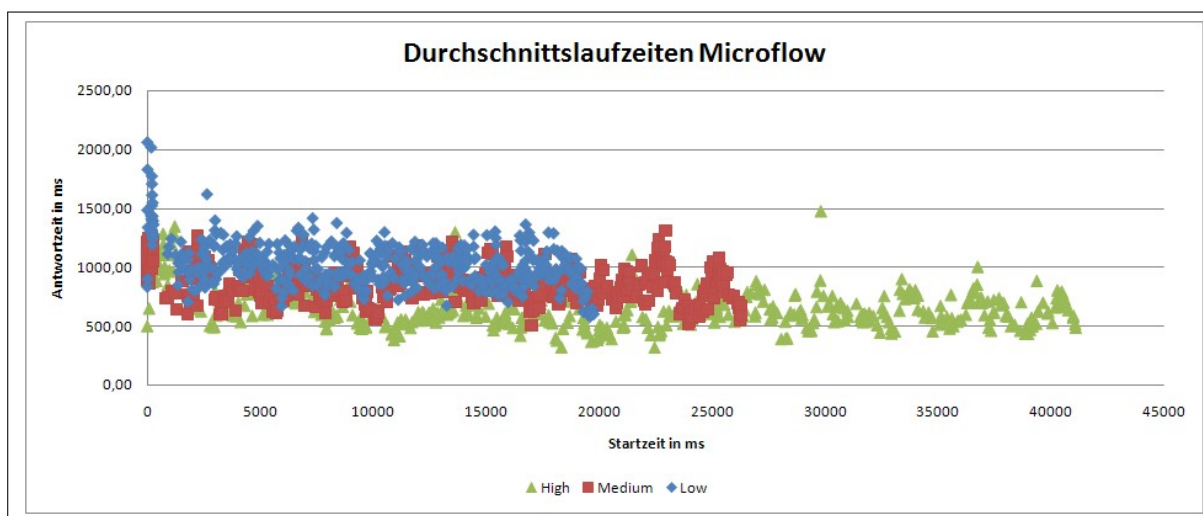


Abbildung 5.31.: Payloadtest - Antwortzeiten Microflow

Lowtests über den Werten des Hightests befinden. Wie deutlich zu sehen ist, konzentriert sich die Anzahl der Aufrufe (die in allen Fällen 500 beträgt) über unterschiedlich lange Abschnitte der X-Achse. Daran sieht man, dass deutlich mehr Prozesse des Lowtests parallel aktiv sein müssen, als dies beim Hightest der Fall ist. Der Mediumtest liegt sowohl in Laufzeit als auch in Durchschnittsdauer zwischen den Werten der anderen beiden, allerdings

## 5. Testergebnisse

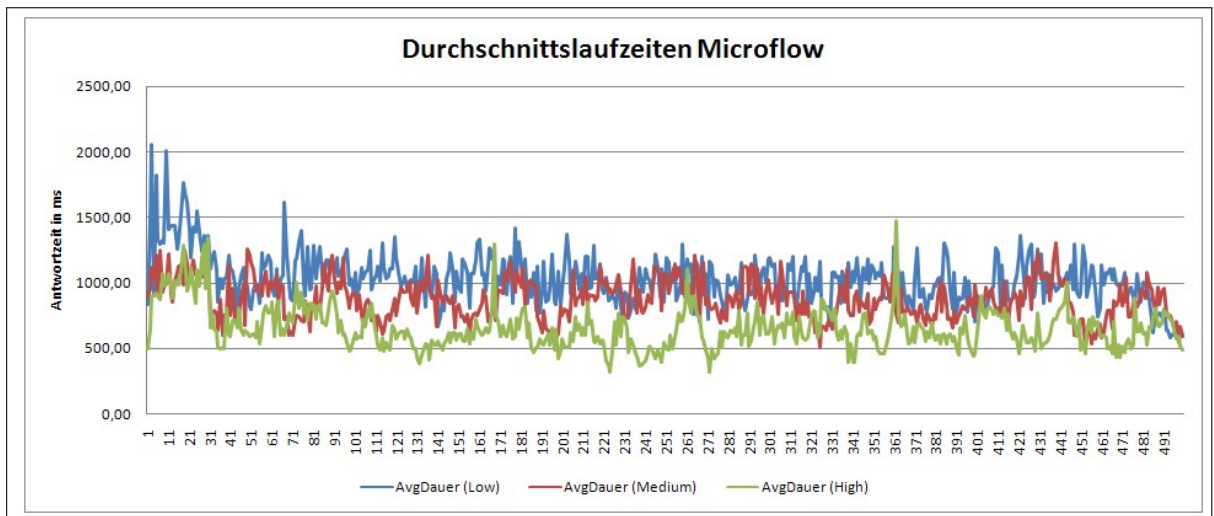


Abbildung 5.32.: Payloadtest - Antwortzeiten Microflow

ist bei allen drei Tests die Streuung recht hoch.

Im Liniendiagramm lässt sich gut erkennen, dass besonders die Anfangswerte des Lowtests sehr hoch liegen, und dann auf ein Niveau absinken, dass sehr nah am Mediumtest liegt. Die Werte des Hightest liegen bis auf wenige Ausnahmen klar unter den Werten der anderen beiden Tests.

Das Punktdiagramm (Abb. 5.33) zeigt hauptsächlich die große Nähe der Werte

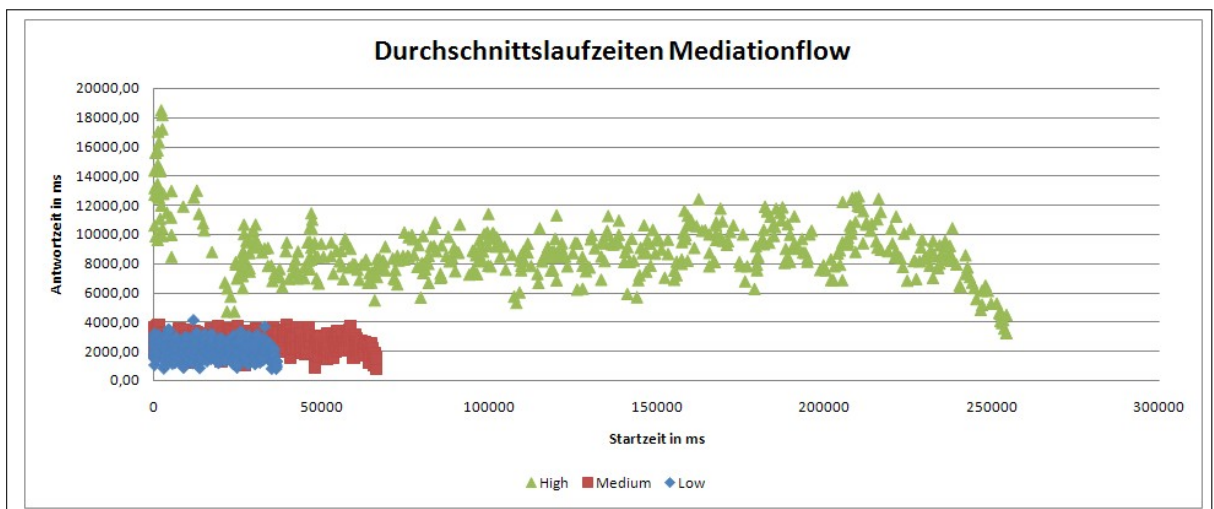


Abbildung 5.33.: Payloadtest - Antwortzeiten Mediationflow

von Low- und Mediumtest. Die Streuung ist so hoch, dass die überwiegende Anzahl der

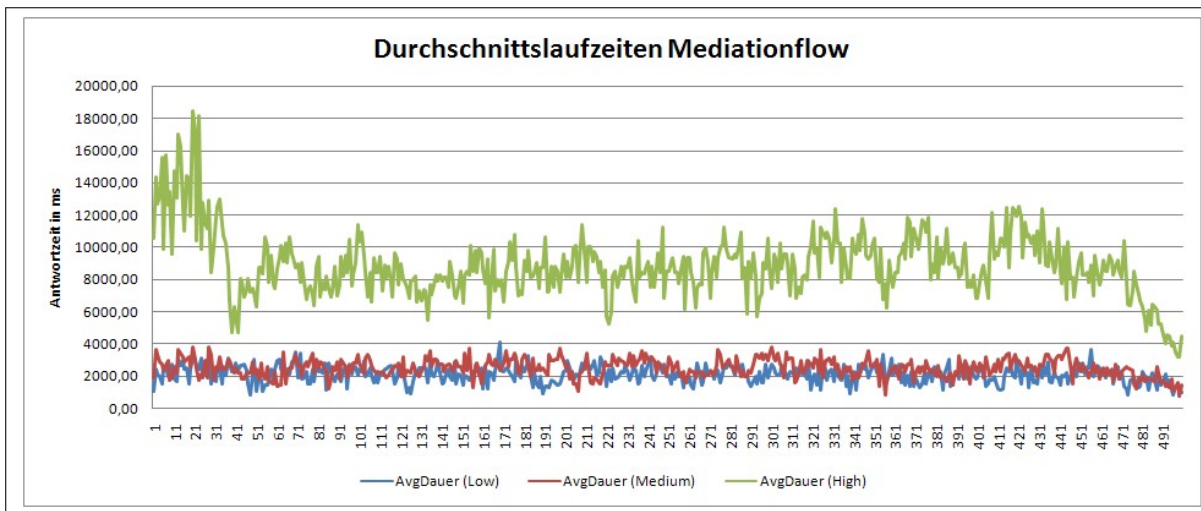


Abbildung 5.34.: Payloadtest - Antwortzeiten Mediationflow

Werte des Mediumtests im Wertebereich des Lowtests liegen. Hier erkennt man auch die deutlich höhere Gesamtdauer beim Mediumtest, die nicht allein auf die Durchschnittsdauer zurückzuführen ist. Beim Lowtest erkennt man die hohen Werte am Anfang sowie die starke Streuung sehr deutlich. Die Werte der Durchschnittsdauer scheinen sich, nachdem sie unmittelbar nach dem Start des Tests zurückgegangen sind, im weiteren Verlauf des Tests tendenziell zu erhöhen, fallen jedoch wie gewohnt am Ende des Tests schließlich klar ab. Dies zeigt sich auch noch einmal im Liniendiagramm (Abb. 5.34), in dem man diesen Verlauf der Werte des Hightests noch einmal besser erkennt. Die Kurven des Low- und des Mediumtests unterstreichen mit ihrem Verlauf noch einmal, wie gering der Unterschied zwischen den beiden Tests wirklich ist.

Auch beim Macroflow kann man sehen, wie gering die Unterschiede zwischen Low- und Mediumtest tatsächlich sind. Dadurch ist auch die Gruppenbildung weitestgehend gleich. Man erkennt aber, dass die Werte des Mediumtests im Schnitt deutlich über denen des Lowtests liegen, und im Abfallen der Werte am Ende des Tests verdeutlicht sich noch einmal die unterschiedliche Gesamtzeit.

Die Werte des Hightests setzen sich deutlicher ab, hier ist aber auch der Sprung im Parameterwert höher.

Zum Abschluss der Betrachtung ist in den Diagrammen 5.37 auf Seite 83, 5.38 auf Seite 83 und 5.39 auf Seite 84 noch einmal die Abweichung der Durchschnittswerte vom Gesamtdurchschnitt für die verschiedenen Tests der einzelnen Prozesstypen dargestellt.

Interessanterweise zeigt der Microflow (Abb. 5.37 auf Seite 83) die höchste Abweichung beim Lowtest und zwar unmittelbar zu Beginn. Dies kann auf die umgekehrte Entwicklung der

## 5. Testergebnisse

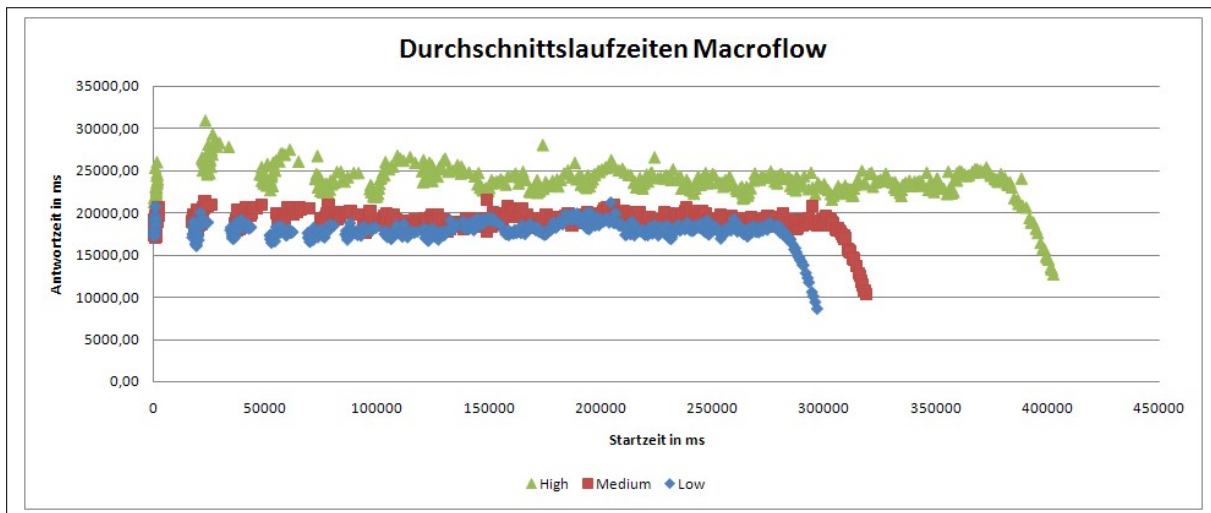


Abbildung 5.35.: Payloadtest - Antwortzeiten Macroflow

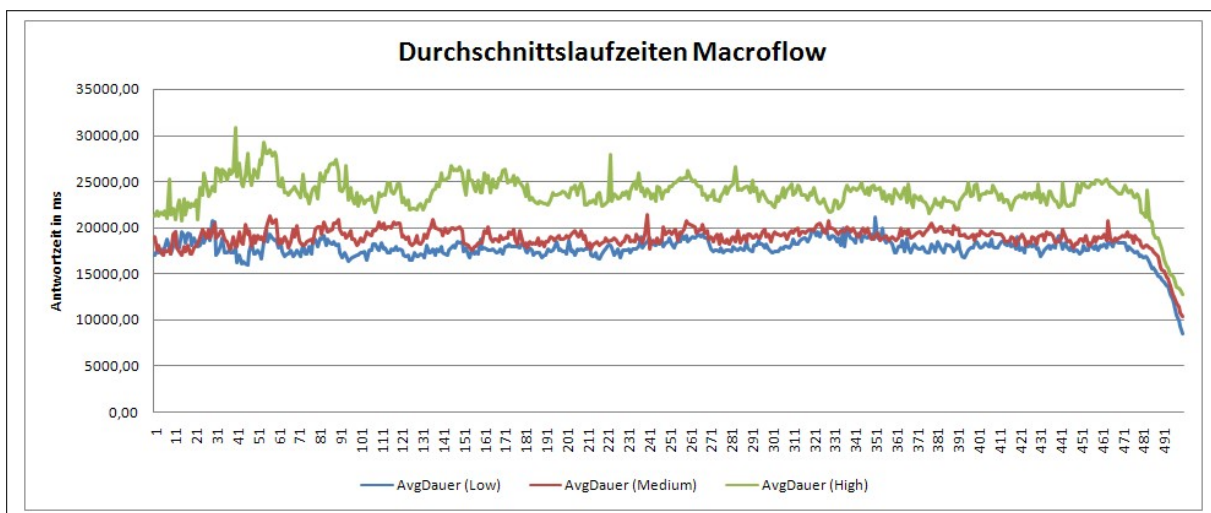


Abbildung 5.36.: Payloadtest - Antwortzeiten Macroflow

Testergebnisse beim Microflow zurückzuführen sein, bei denen der Lowtest den höchsten durchschnittlichen Wert hatte. Entsprechend sticht der Hightest in diesem Diagramm auch kaum besonders hervor.

Beim Mediationflow (Abb. 5.38 auf der nächsten Seite) hingegen dominieren die Abweichungen des Hightests klar das Bild. Analog zur Entwicklung der Durchschnittswerte sind die Unterschiede zwischen Low- und Mediumtest hingegen kaum auszumachen.

Beim Macroflow sind die stärksten Abweichungen vor allem in der ersten Hälfte des Tests auszumachen und dort besonders zu Beginn, sofern man von dem bekannten

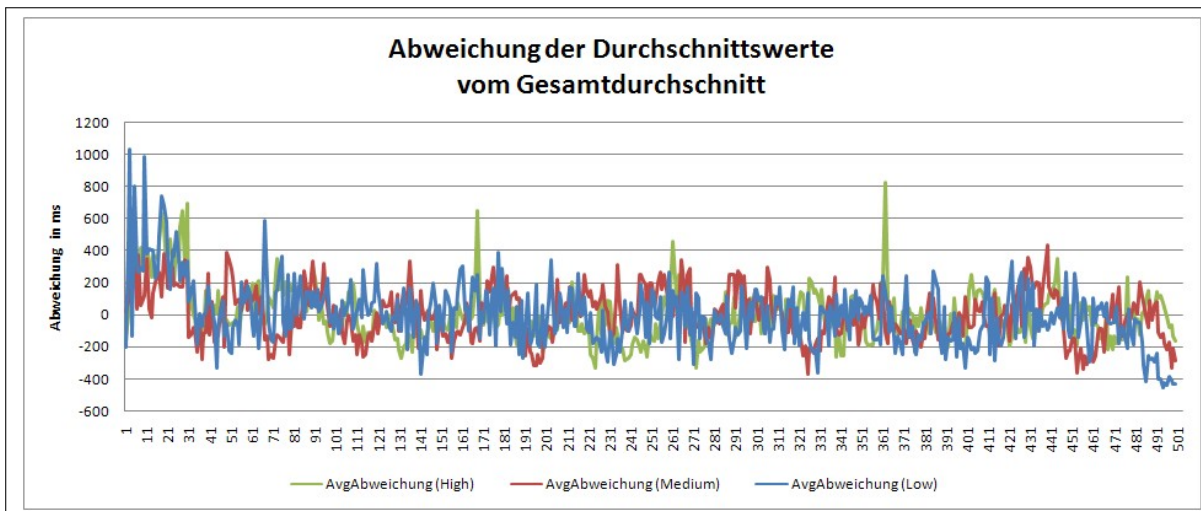


Abbildung 5.37.: Payloadtest - Abweichungen Microflow

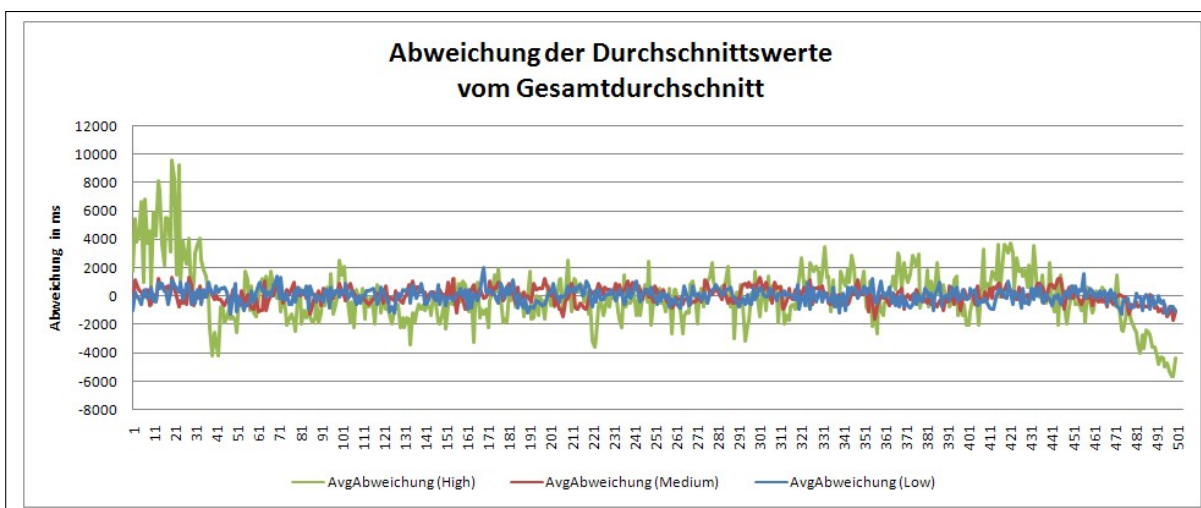


Abbildung 5.38.: Payloadtest - Abweichungen Mediationflow

Abfallen der Werte am Ende der Tests absieht. Insgesamt sind auch hier die Abweichungen vergleichsweise moderat, besonders im Vergleich zum Mediationflow, dessen Maxima beim Hightest deutlich über den Maxima des entsprechenden Wertes des Macroflows liegt.

Im Payloadtestfall bietet der Microflow wohl das interessanteste Ergebnis. Die steigende Gesamtdauer bei abnehmender Durchschnittsdauer führt zu dem Schluss, dass weniger Prozesse parallel aktiv sind. Offenbar wirkt sich die Nachrichtengröße auf die Fähigkeit des Servers aus, parallele Prozesse ablaufen zu lassen. Diese Auswirkungen

## 5. Testergebnisse

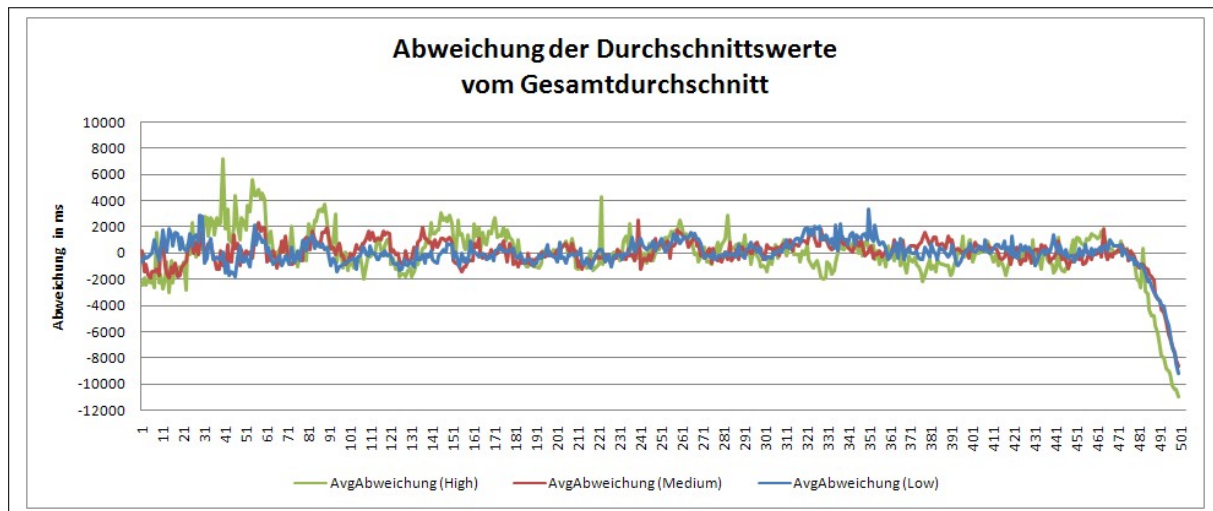


Abbildung 5.39.: Payloadtest - Abweichungen Macroflow

waren bereits bei den Ergebnissen des Maximumtests zu sehen, wo die Durchschnittsdauer unter der des Hightests des Serviceaufrufes lag, obwohl die Gesamtdauer größer war. Da beim Maximumtest die Payload bedeutend höher lag, war dort der selbe Effekt zu beobachten. Dadurch ist es schwierig die direkten Auswirkungen der Payload auf das Laufzeitverhalten abzuschätzen, da es sich nicht direkt von den Auswirkungen geringerer paralleler Prozesse trennen lässt. Die Auswirkungen der gewählten Parameterwerte auf den Durchsatz sind jedoch nur wenig stärker als dies bei den Serviceaufrufparametern der Fall war, obwohl die Steigerungen der Parameterwerte bei der Payload am größten waren. Damit sollte die Anzahl der Serviceaufrufe für den Microflow die bestimmende Größe bleiben.

Der Mediationflow scheint bis zu einem gewissen Grad mit zusätzlicher Payload gut umgehen zu können, wie der geringe Anstieg der Werte bei der ersten Erhöhung zeigt. Die deutliche Verschlechterung bei der zweiten Erhöhung zeigt jedoch, dass dies nicht in beliebigem Umfang gilt. Diese Probleme im Umgang mit hoher Payload und gleichzeitig eine hohe Anzahl an parallelen Prozessen sind wohl auch der Grund, weshalb der Mediationflow nicht mit den ursprünglichen Maximumwerten im Maximumtestfall aufgerufen werden konnte.

Der Macroflow zeigte ein sehr gutes Verhalten bei großen Nachrichten. Die Werte verschlechterten sich nur in geringem Umfang, so dass wohl auch größere Nachrichtengrößen für den Macroflow kein Problem darstellen dürften.

# Gesamtbewertung

---

## 6.1. Zusammenfassung der Testergebnisse

### Microflow

Der Microflow zeigt von den betrachteten Prozesstypen insgesamt die besten Ergebnisse. Schon beim einfachen, sequentiellen Aufruf hatte er sehr gute Resultate, wenn er auch hier nur unwesentlich unter dem Mediationflow lag. Deutlicher wurden seine Eigenschaften bei der Erhöhung der Parameter. Bei einer Erhöhung der Threadanzahl wirken sich die Beeinträchtigung durch parallel ablaufende Prozesse nur in geringem Maße negativ auf die Durchschnittsdauer aus. Daher verbessert sich der Durchsatz als Folge der parallelen Verarbeitung. Innerhalb der betrachteten Parameterwerte wirkte sich eine Erhöhung der aufrufenden Threads also stets positiv auf die Gesamtdauer aus (s. a. Tabelle 6.1 auf Seite 87). Den deutlichsten Einfluss auf die Prozesslaufzeit, zeigte die veränderte Anzahl der vom Prozess verwendeten Services. Die Veränderungen bei Durchschnittsdauer und Gesamtdauer waren bei einer Vergrößerung der Recipient List klar zu erkennen. Dieses Verhalten ist auch nachvollziehbar, da der Prozess keinen Einfluss auf die Dauer der externen Aufrufe hat. Durch die sequentielle Verarbeitung der Aufrufe durch den Prozess kann hier auch sonst keine Optimierung erfolgen. So stellt die zusätzliche Menge an Serviceaufrufen praktisch eine Verlängerung des Prozesses dar, was eine entsprechende Vergrößerung von Durchschnitts- und Gesamtdauer zur Folge haben muss.

Das Verhalten bei der Veränderung der Payloadgröße ist überraschend. War die höhere Gesamtdauer natürlich zu erwarten, ist das Zurückgehen der Durchschnittswerte praktisch nur durch eine verringerte parallele Belastung zu erklären. Trotz dieses Verhaltens wirken die Effekte, die eine höhere Payload auf den Microflow hat, nicht extrem. Obwohl die Erhöhung der Parameterwerte sehr deutlich ist, sinkt die Gesamtlaufzeit nur wenig stärker als bei den Servicecalls, so dass diese den bestimmenden Einfluss auf den Microflow darstellen.

### **Mediationflow**

Der Mediationflow offenbart einige gravierende Nachteile im Verlauf der Tests. Beim sequentiellen Aufruf sind seine Ergebnisse noch mit denen des Microflow vergleichbar, beide zeigen sogar die selbe Minimumdauer. Auch bei der Erhöhung der Serviceaufrufe zeigt er ein prinzipiell ähnliches Verhalten wie der Microflow. Ein großes Problem ist jedoch die parallele Ausführung der Prozesse. Von allen Prozesstypen zeigt er die stärkste Verschlechterung bei der Durchschnittsdauer. Bei der Gesamtdauer kann dies durch die parallele Ausführung nicht mehr kompensiert werden, was zu deutlichen Verschlechterungen des Durchsatzes führt (s. a. hier Tabelle 6.1 auf der nächsten Seite). Insgesamt ist für die Laufzeit des Mediationflow die Anzahl der laufende Prozesse die bestimmende Größe.

Bei der Payload bleibt ein gemischtes Bild. Während die erste Erhöhung sich nur gering auswirkt, zeigt er mit der maximalen Payloadbelastung deutliche Einbußen bei den Werten. Diese in Verbindung mit hoher paralleler Belastung stellt wohl die entscheidende Hürde für den Mediatonflow dar, wie die zwangsweise reduzierten Anforderungen des Maximumtests zeigten.

Ein ausschlaggebender Punkt ist für den Mediationflow der verfügbare Speicher. Bereits im Vorfeld der Tests musste die Größe des Java-Heaps erst mehrfach erhöht werden, bis überhaupt ein vernünftiger paralleler Ablauf des Prozesstyps zu erreichen war. Auch beim sequentiellen Aufruf lassen sich die wenigen, aber hohen Extremwerte auf die Garbage Collection zurückführen. Dass er als einziger der drei Prozesstypen nicht den Maximumtest mit den vorgegebenen Parameterwerten durchführen konnte, war ebenfalls auf ein Speicherproblem zurückzuführen. Insgesamt schneidet er im Vergleich zum Microflow also nicht nur deutlich schlechter ab, sondern kann bei gleichen Ressourcen dieselben Anforderungen teilweise gar nicht erfüllen.

### **Macroflow**

Der Macroflow hat insgesamt sehr gute Resultate geliefert. Die Werte liegen natürlich alle in einem sehr hohen Bereich. Dies muss man ihm zugestehen, da die Anforderungen bezüglich der Servicequalität auch ganz anders sind. Damit verbietet sich natürlich ein direkter Vergleich mit Micro- und Mediationflow. Betrachtet man aber die Entwicklung der Werte bei den verschiedenen Tests, fällt die starke Ähnlichkeit zum Microflow auf. Die Steigerungen der Werte liegen stets sehr nah an denen des Microflow und die Anzahl der Serviceaufrufe ist ebenfalls die dominierende Größe für die Laufzeit. Die Beeinträchtigung durch parallel ablaufende Prozesse hält sich in Grenzen, so dass der Durchsatz trotz höherer Durchschnittsdauer profitieren kann. Selbst bei der Payloaderhöhung zeigt er nur geringe Verschlechterungen. Insgesamt ist sein Verlauf bei allen Tests sehr gemäßigt, ohne große Streuungen oder starke Veränderungen bei anderen Parametern. Letztenendes sind dies auch die Anforderungen, die man an langlaufende Prozesse stellt. Während die Durchschnittsdauer eben nicht im Vordergrund steht, erwartet man, dass der Macroflow mit allen Veränderungen, die während seiner langen Laufzeit auftreten können, zurecht kommt. Extreme Reaktionen auf hohe Parameterwerte würden den erfolgreichen Abschluss des Gesamtprozesses gefährden, und müssen daher vermieden werden.

Zur Verdeutlichung der obigen Aussagen gibt die Tabelle 6.1 noch einmal einen Überblick über die Entwicklung des Durchsatzes bei Minimum-, Medium- und Maximumtestfall für jeden Prozesstyp. Die Werte sind wie sonst auch in Millisekunden angegeben.

Tabelle 6.1.: Vergleich des Durchsatzes

Testfall	tps Microflow	tps Mediationflow	tps Macroflow
Minimumtestfall	13,07	12,15	1,41
Mediumtestfall	18,61	7,40	1,52
Maximumtestfall	11,00	1,93	0,91

## 6.2. Schlußfolgerungen bezüglich der Architektureigenschaften

Die Ergebnisse des Microflow belegen zunächst einmal, dass die zu Grunde liegende Architektur auch bei hohen Belastungen keine kritischen Schwachpunkte aufweist. Besonders der gute Umgang mit parallelen Anfragen widerlegt eventuelle Befürchtungen, die durch das Verwenden eines Workflow-Management-Systems als zentraler Komponente bestehen könnten. Das Problem eines Single-Point-of-Failure bleibt natürlich erhalten, jedoch beweist der WSP, dass sich bei geeigneter Implementierung die gegenseitigen Beeinflussung der Prozesse in akzeptablen Grenzen halten kann.

Auch der Umgang mit großen Informationsmengen, wie er bei den Payloadtests gegeben war, spricht prinzipiell für die Architektur eines Workflow-Management-Systems. Die Auswirkungen der Payload sind angesichts der hohen Parameterwerte überschaubar, und besonders die gleichmäßige Entwicklung des Macroflow spricht für die Vorzüge einer zentralen Verwaltung der benötigten Informationen.

Die Serviceaufrufe als bestimmende Größe für die Dauer der Prozesse kann als ein optimales Ergebnis betrachtet werden. Es bedeutet, dass das Laufzeitverhalten des Microflow zuerst von einer der zentralen Eingabegrößen abhängt, die auch die Gesamtgröße des Prozesses entscheidend mitbestimmt. Die Laufzeit des Microflow wird also nicht primär durch große Eingangswerte (also durch die Payload) oder durch die Beeinträchtigung anderer Prozesse beeinflusst, sondern durch die Struktur des Prozesses an sich. Prinzipiell kann also das Verhalten eines Workflows weitgehend unabhängig von den weiteren Bedingungen anhand des reinen Modells abgeschätzt werden.. Hier zeigt sich, dass die Trennung von Modell und Ausführung praktikabel ist und einer der zentralen Vorteile des Workflowansatzes sich bewährt.

Die Testresultate des Mediationflow lassen nur begrenzt Rückschlüsse auf bestimmte Architektureigenschaften zu. Die hohe Belastung durch parallele Anfragen lässt sich auf eine sehr intensive Speicherbenutzung zurückführen. Dies erscheint logisch, muss beim Mediationflow doch vor jeder neuen Primitive die Nachricht transformiert werden. Anders als beim Workflow, wo stets nur die Daten verarbeitet werden müssen, die auch konkret benötigt werden,

muss beim Mediationflow stets die komplette Information, die an den verschiedenen Stellen des Prozesses benötigt wird, transformiert werden. Und gerade bei einer hohen Payload macht sich diese Belastung dann extrem bemerkbar, wie der Zusammenbruch des Servers bei den ursprünglichen Maximumwerten bewiesen hat. Die Ursache dieses Problems, dass anders als im Workflow, keine zentrale Informationsverwaltung vorhanden ist, wo stets nur die Daten abgerufen werden können, die gebraucht werden, stellt eine der zentralsten Eigenschaften einer Pipes-And-Filter-Architektur da. Dass diese mangelnde zentrale Information auch in der Praxis erhebliche Auswirkungen haben, lässt sich an Hand der Testergebnisse belegen. Dies lässt allerdings keine generellen Rückschlüsse auf die PaF-Architektur zu. Dafür ist zum einen der Mediationflow eine zu untypische Implementierung. Zum anderen kann eine Implementierung nur in seltenen Fällen negative Eigenschaften einer Architektur beweisen, da die Mankos stets aus einer schlechten Umsetzung resultieren können, die schwierige Punkte einer Architektur in der Praxis erst kritisch werden lässt. Anders herum ist es einfacher, da positive Resultate zeigen, dass eine Architektur erfolgreich umgesetzt werden kann. Allerdings sind die Testergebnisse durchaus ein Indiz, dass bei einer Umsetzung einer PaF-Architektur, die fehlende zentrale Information eines der Hauptprobleme in Hinblick auf die Performanz ist.

## Zusammenfassung und Fazit

---

Aufgabe dieser Studienarbeit war es, einen Performancevergleich zwischen Implementierungen einer Pipes-and-Filter-Architektur bzw. einem Workflow-basierten System durchzuführen, und eventuelle Rückschlüsse auf die Architektur darzulegen.

Zu Beginn wurden in Kapitel 2 beide Architekturen vorgestellt und ihre Unterschiede betrachtet. Als Implementierungen wurde der Websphere Process Server sowie der Websphere Enterprise Service Bus verwendet. Auf die Eignung dieser beiden Systeme als Implementierungen der jeweiligen Architektur wurde in Kapitel 2.4.4 auf Seite 20 näher eingegangen. Prinzipiell gilt, dass während der WPS ein typisches Workflow-Management-System darstellt, der WESB nur begrenzt als Implementierung einer PaF-Architektur gelten kann. Allerdings machen die gemeinsamen Grundlagen der grundsätzlich unterschiedlichen Produkte einen sinnvollen Vergleich überhaupt erst möglich.

Basis der vorgenommenen Performancetests war das für diese Arbeit entworfene und in Kapitel 3 vorgestellte Erweiterte Loan Broker Szenario. Hier wurde ein hinreichend großes Szenario entwickelt, das zum einen eine gewisse Komplexität der Verarbeitungslogik aufweist, und zum anderen eine größere Anzahl an externen Services benötigt. Dieses wurde einmal implementierungsunabhängig in Form von Patterns dargestellt, und schließlich die konkrete Umsetzung als Micro- bzw. Macro-Flow und als Mediatonflow gezeigt.

Kapitel 4 umfasste die Rahmenbedingungen des Tests. Es beinhaltet zunächst einmal eine Beschreibung der Testumgebung. Anschließend werden ausführlich die verschiedenen Alternativen der Testwerkzeuge vorgestellt und die Wahl der CEI begründet. Schließlich wird ein Überblick über die entworfenen Testfälle gegeben und die jeweilige Vorgehensweise für die Durchführung der Tests erläutert.

Die Ergebnisse der Tests werden in Kapitel 5 vorgestellt. Zunächst werden die Testfälle präsentiert, bei denen alle Parameter nach und nach erhöht wurden, um das Verhalten der Prozesstypen unter zunehmender Belastung zu prüfen. In den Abschnitten 5.4 bis 5.6 werden schließlich die einzelnen Parameter auf ihren individuellen Einfluss auf die Prozesstypen hin untersucht.

Kapitel 6 fasst abschließend die Testresultate noch einmal zusammen und legt die Rückschlüsse dar, die sich daraus für die Eigenschaften der Architekturen ergeben.

### **Fazit**

Durch die zunehmende Bedeutung flexibler Arbeitsabläufe in Unternehmen wird die Bedeutung von Softwaresystemen, die diese Abläufe unterstützen und steuern können immer wichtiger. Während die grundlegende Funktionalität mittlerweile seit Jahren zur Verfügung steht, gewinnen heute spezielle Funktionalitäten an Bedeutung. Dazu gehört zum Beispiel die Unterstützung manueller Tätigkeiten. Trotz allem ist die Gesamtperformance eines solchen Systems nach wie vor eine der bedeutensten Kriterien für einen erfolgreichen Einsatz solcher Systeme. Innerhalb dieser Arbeit wurde gezeigt, dass die Workflows und damit der Workflowbasierte Ansatz allen gestellten Anforderungen mit einer akzeptablen Performance sowohl als Micro- wie auch als Macroflow genügen können. Für den Mediationflow kann dies nicht in gleichem Umfang gesagt werden, wie besonders seine Probleme mit parallelen Anfragen zeigen. Daraus lassen sich jedoch keine eindeutigen Schlüsse auf das generelle Verhalten einer PaF-Architektur machen, wie in Kapitel 6.2 dargelegt wurde. Die prinzipielle Eignung beider Architekturen für die Aufgaben, in denen sie vorrangig eingesetzt werden, zeigt ihre hohe Verbreitung in der Praxis. Weitere Untersuchungen mit anderen PaF-Implementierungen könnten hier eindeutigeres Resultate liefern. Es dürfte jedoch schwierig sein, Produkte beider Architekturen zu finden, die von ihrem technischen Entwicklungsstand her ähnlich vergleichbar sind, was die Interpretation entsprechender Testresultate wiederum schwierig macht.

# Anhang

---

## A.1. Konfigurationsänderungen bei den Wepshere-Produkten

Die folgende Auflistung gibt eine Übersicht über die in der Admin-Console des WPS geänderten Einstellungen, und wo diese zu finden sind.

- **Deaktivierung der Überwachungsfunktionen**
  - Fehlerbehebung > Protokollierung und Tracing > server1 > Detailstufe für Protokoll ändern:  
Configuration und Runtime auf: \*=all=disabled
  - Überwachung und Optimierung > Performance Monitoring Infrastructure (PMI) > server1:  
Select none und Performance Monitoring Infrastructure (PMI) aktivieren ausschalten
  - Anwendungsserver > server1 > Business Process Choreographer Container > Statusüberwachungsfunktionen:  
alles ausschalten (außer CEI)
- **Java Heap Size anpassen**
  - Anwendungsserver > server1 > Prozessdefinition > Java Virtual Machine:  
Anfangsgröße des Heap-Speichers und Maximale Größe des Heap-Speichers setzen:  
Minimum auf 256  
Maximum auf 1280
- **Entwicklungsmodus deaktivieren**
  - Anwendungsserver > server1:  
Im Entwicklungsmodus ausführen deakt.
- **Entwicklungsmodus deaktivieren**

- Anwendungsserver > server1:  
Im Entwicklungsmodus ausführen deaktivieren.

- **Work Manager optimieren**

- Ressourcen > Asynchronous Bean > Work Manager:  
AppSchedulerWorkManager: Default 10 - Neu 100  
BPENavigationWorkManager: Default 12 - Neu 100  
BPESchedulerWorkManager: Default 10 - Neu 100  
DefaultWorkManager: Default 10 - Neu 100  
es-workmanager: Default 20 - Neu 200

- **Thread Pool konfigurieren**

- Anwendungsserver > server1 > Thread-Pools > NAME : Max. Größe:  
ORB.thread.pool: Default 50 - Neu 80  
Default: Default 20 - Neu 80  
WebContainer: Default 50 - Neu 80  
SIBFAPThreadPool: Default 50 - Neu 50  
SIBFAPInboundThreadPool: Default 50 - Neu 50  
TCPChannel.DCS: Default 20 - Neu 80

- **Data Source Connection Pool konfigurieren**

- Ressourcen > JDBC > Datenquellen > NAME > Merkmale für Verbindungspools :  
Maximale Anzahl  
BPEDataSourceDb2: Default 30 - Neu 100  
Business Process Choreographer ME data source: Default 10 - Neu 100  
CEI ME datasource: Default 10 - Neu 100  
Default Datasource: Default 10 - Neu 100  
ESBLoggerMediationDataSource: Default 10 - Neu 100  
SCA Application Bus ME data source: Default 10 - Neu 100  
SCA System Bus ME data source: Default 10 - Neu 100  
WBI\_DataSource: Default 10 - Neu 100  
event: Default 10 - Neu 100  
event\_catalog: Default 10 - Neu 100

- **Validierung der CEI-Events deaktivieren**

- Serviceintegration > CEI > Ereignis-Emitter-Factorys > Default Common Event  
Infrastructure emitter > Benutzerdefinierte Merkmale:  
Neue Property: name=validateEvent, value=false

- **Activation specification konfigurieren**

- Ressourcen > JMS > Aktivierungsspezifikationen > NAME: Maximale Anzahl  
paralleler Endpoints:  
BPEInternalActivationSpec: Default 10 - Neu 100  
CommonEventInfrastructure\_ActivationSpec: Default 10 - Neu 100

BFMJMSAS: Default 10 - Neu 100

HTMInternalActivationSpec: Default 10 - Neu 100

- **Persistente Http-Verbindungen nutzen**

- Anwendungsserver > server1 > Endpoint-Listener > Ports > SOAPPORT - Zugeordnete Transporte anzeigen > HTTP-Channel für eingehende Anforderungen: Persistente Verbindungen aktivieren

- **CEI-Eventverteilung deaktivieren**

- Serviceintegration > CEI > Ereignisservice > Ereignisservices > Default Common Event Infrastructure event server: Ereignisverteilung deaktivieren

- **Mediation-Thread-Pool konfigurieren**

- Busse > CommonEventInfrastructure\_Bus > Messaging-Steuerkomponenten > BUSNAME > Mediation-Thread-Pool:
  - CEI-Bus: Default 5 - Neu 100
  - BPC-Bus: Default 5 - Neu 100
  - SCA-Applicatin-Bus: Default 5 - Neu 100
  - SCA-System-Bus: Default 5 - Neu 100

## Literaturverzeichnis

---

- [AHO4] W. van der Aalst, K. van Hee. *Workflow management : models, methods, and systems*. Cambridge, Mass. ; London : MIT Press, 1. mit press paperback ed. edition, 2004. (Zitiert auf Seite 7)
- [BPE] WSBPEL. URL [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel). (Zitiert auf Seite 16)
- [Bus00] F. Buschmann, editor. *Pattern-orientierte Softwarearchitektur : ein Pattern-System*. München [u.a.] : Addison-Wesley, 1. korr. nachdruck edition, 2000. (Zitiert auf Seite 12)
- [CBE] Common Base Events. URL <http://www.ibm.com/developerworks/library/specification/ws-cbe/>. (Zitiert auf Seite 43)
- [Coa] W. M. Coalition. Workflow Reference Modell. URL <http://www.wfmc.org/reference-model.html>. (Zitiert auf den Seiten 5 und 9)
- [Coa99] W. M. Coalition. Terminology & Glossary Document Number WFMC-TC-1011, 1999. URL [http://www.wfmc.org/index.php?option=com\\_docman&task=doc\\_download&gid=93&Itemid=74](http://www.wfmc.org/index.php?option=com_docman&task=doc_download&gid=93&Itemid=74). (Zitiert auf Seite 8)
- [Cop95] J. O. Coplien, editor. *Pattern languages of program design*, volume Band 1. Reading, Mass. [u.a.] : Addison-Wesley, 1995. (Zitiert auf Seite 12)
- [EAI] Enterprise Integration Patterns. URL <http://www.eaipatterns.com/>. (Zitiert auf Seite 24)
- [Faso8] M. Fasbinder. BPEL or ESB: Which should you use?, 2008. URL [http://www.ibm.com/developerworks/websphere/library/techarticles/0803\\_fasbinder2/0803\\_fasbinder2.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0803_fasbinder2/0803_fasbinder2.html). (Zitiert auf Seite 19)
- [Hol95] D. Hollingsworth. The Workflow Reference Model Document Number TCoo-1003, 1995. URL [http://www.wfmc.org/index.php?option=com\\_docman&task=doc\\_download&gid=92&Itemid=72](http://www.wfmc.org/index.php?option=com_docman&task=doc_download&gid=92&Itemid=72). (Zitiert auf Seite 10)
- [HW03] G. Hohpe, B. Woolf. *Enterprise integration patterns : designing, building, and deploying messaging solutions*. Boston, Mass. [u.a.] : Addison-Wesley, 2003. (Zitiert auf den Seiten 23 und 24)

- [KBC<sup>+</sup>07] M. Keen, N. Betteridge, D. Cardalliaguet, A. C. Ferrari, A. Groeschl, E. Kuehlthau, M. Maia, S. Mamindla, K. Senior, F. Williams. *z/OS Getting Started: WebSphere Process Server and WebSphere Enterprise Service Bus V6*, 2007. URL <http://www.redbooks.ibm.com/redbooks/pdfs/sg247378.pdf>. (Zitiert auf den Seiten 5, 16, 18, 19 und 20)
- [KGG<sup>+</sup>07] M. Keen, M. Gandhe, S. Gibney, A. Lis, D. Veronese, Y. Z. Zhang. *Production Topologies for WebSphere Process Server and WebSphere ESB V6*, 2007. URL <http://www.redbooks.ibm.com/redbooks/pdfs/sg247413.pdf>. (Zitiert auf Seite 16)
- [KMC<sup>+</sup>06] M. Keen, B. Moore, A. Carvalho, M. Hamann, P. Imandi, R. Lotter, P. Norton, C. Ringler, G. Telerman. *Getting Started with WebSphere Enterprise Service Bus V6*, 2006. URL <http://www.redbooks.ibm.com/redbooks/pdfs/sg247212.pdf>. (Zitiert auf den Seiten 5, 16, 17 und 19)
- [LR00] F. Leymann, D. Roller. *Production workflow : concepts and techniques*. Upper Saddle River, NJ : Prentice Hall PTR, 2000. (Zitiert auf den Seiten 7 und 8)
- [Mülo5] J. Müller. *Workflow based integration : Grundlagen, Technologien, Management*. Berlin ; Heidelberg [u.a.] : Springer, 2005. (Zitiert auf den Seiten 9 und 12)
- [MPGMo8] T. Muehlfriedel, G. Pfau, J. Grundler, D. Meyer. *WebSphere Process Server V6.1 – Business Process Choreographer: Performance Tuning Automatic Business Processes for Production Scenarios with DB2*, 2008. URL <http://www-01.ibm.com/support/docview.wss?uid=swg27012639&aid=1>. (Zitiert auf Seite 41)
- [PT008] IBM WebSphere Business Process Management V6.1 Performance Tuning, 2008. URL <http://www.redbooks.ibm.com/redpapers/pdfs/redp4431.pdf>. (Zitiert auf Seite 41)
- [SOU] SOAP UI. URL <http://www.soapui.org/>. (Zitiert auf Seite 41)
- [TM003] IBM WebSphere Application Server, Version 5.1 Performance Tuning and Monitoring, 2003. URL [ftp://ftp.software.ibm.com/software/webserver/appserv/library/wasv51base\\_perf.pdf](ftp://ftp.software.ibm.com/software/webserver/appserv/library/wasv51base_perf.pdf). (Zitiert auf Seite 42)
- [WSD01] Web Services Description Language, 2001. URL <http://www.w3.org/TR/wsd1.html>. (Zitiert auf Seite 16)

Alle URLs wurden zuletzt am 25.10.2008 geprüft.

## **Erklärung**

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

---

(Oliver Eckhardt)