Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 2540

Analyse der "Pipes and Filter" Architektur gegenüber instanzbasierten Ansätzen bei Workflows

Jan Trautvetter

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Thorsten Scheibler
begonnen am:	01.03.2006
beendet am:	31.08.2006
CR-Klassifikation:	H.4.1

Erklärung

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

Unterschrift:

Stuttgart 31.08.2006

Inhaltsverzeichnis

1 Einleitung 1
2 Theoretische Analyse der beiden Lösungskonzepte 2
2.1 Einschränkung des betrachteten Einsatzgebiet 2
2.2 Workflow-Management-Systeme 3
2.2.1 Einführung 3
2.2.2 Architektur 4
2.2.3 Einschränkung des betrachteten Einsatzgebiets 8
2.2.4 Effizienz 9
2.3 Pipes and Filter Muster 10
2.3.1 Einführung 10
2.3.2 Architektur 12
2.3.3 Einschränkung des betrachteten Einsatzgebiets 14
2.3.4 Effizienz der Problemlösung 15
2.4 Gegenüberstellung der beiden Ansätze 19
2.4.1 Einsatzgebiete 19
2.4.2 Effizienz 20
3 Szenarien für die Effizienzanalyse 21
3.1 Ziel der Szenarien 21
3.2 Liste der Szenarien 22
4 Versuchsaufbau und Anwendungskonzeption 24
4.1 Hard- und Softwareumgebung 24
4.2 Anwendungskomponenten 24
5 Messwerkzeug und Messpunkte 32
5.1 Messziele 32
5.2 Auswahl der Messwerkzeuges 33
5.2.1 Überwachung mit Mitteln des Betriebssystems 33
5.2.2 Überwachung mit Mitteln der Java VM 34
5.2.3 Implementierung des Messwerkzeuges 36
5.3 Messpunkte 37
6 Praktische Analyse der Effizienz 42
6.1 Messergebnisse in Abhängigkeit zur Filteranzahl 42
6.2 Messergebnisse in Abhängigkeit zur Nachrichtengröße 47
6.3 Weitere Vergleiche 52
7 Abschließende Betrachtung 57
8 Glossar 59
A. Macroflow.bpel 64
B. Microflow.bpel 66
C. Aufbau der Message Driven Beans 68
C.1 Client MDB 68
C.2 Filter1 MDB 70
C.3 Filter2 MDB und Filter3 MDB 70
C.4 Filter4 MDB 71
D. Messdaten 72

Tabellenverzeichnis

1 - Auflistung der Szenarios 22
2 - Auflistung der Szenarien für die Analyse I 42
3 - CPU Zeit I der Analyse I 44
4 - CPU Zeit II der Analyse I 45
5 - Speicherallokation der Analyse I 46
6 - Beschriebener Festplattenspeicher der Analyse I 47
7 - Auflistung der Szenarien für die Analyse II 48
8 - CPU Zeit der Analyse II 49
9 - Speicherallokation der Analyse II 50
10 - Beschriebener Festplattenspeicher der Analyse II 51
11 - Messdaten der Analyse III 54
12 - Messdaten der Analyse IV 56
13 - Vergleichbare Messdaten 56
14 - Messdaten zur Anzahl der Filter 72
15 - Messdaten zur Nachrichtengröße 72
16 - Messdaten zu den Topics 72
17 - Messdaten zu dem Scatter and Gather Muster 73

Abbildungsverzeichnis

1 - Bestandteile eines Workflow Management System (Quelle [5]) 4
2 - Definition und Laufzeit eines Workflows (Quelle [5]) 5
3 - Zustandsübergangsdiagramm eines Macroflows (Quelle [5]) 7
4 - Zustandsübergangsdiagramm eines Microflows (Quelle [5]) 8
5 - Konzept des „Pipes and Filter“-Architekturmusters 11
6 - Anwendungsaufbau der Macroflow-Implementierung 26
7 - Anwendungsaufbau der Microflow-Implementierung 27
8 - Anwendungsaufbau der PaF-Implementierung 28
9 - Client zur PaF-Implementierung 29
10 - Client zur Workflow-Implementierung 30
11 - Anwendungsaufbau zum Scatter-Gather Musters 1 31
12 - Anwendungsaufbau zum Scatter-Gather Musters II 31
13 - Bestandteile einer JVMPI Implementierun (Quelle [13]) 35
14 - Messpunkte für die Antwortzeit der PaF-Implementierung 38
15 - Messpunkte für die Antwortzeit der Workflow-Implementierung 39
16 - Messpunkte für die Verarbeitungszeit der PaF-Implementierung 39
17 - Messpunkte für die Verarbeitungszeit der PaF-Implementierung 40
18 - CPU Zeit I der Analyse I 44
19 - CPU Zeit II der Analyse I 45
20 - Speicherallokation der Analyse I 46
21 - Beschriebener Festplattenspeicher der Analyse I 47
22 - CPU Zeit der Analyse II 49
23 - Speicherallokation der Analyse II 50
24 - Beschriebener Festplattenspeicher der Analyse II 51
25 - Messdaten der Analyse III 53
26 - Messdaten der Analyse IV 55

1 Einleitung

Hervorgerufen durch die große *Heterogenität* der bestehenden Anwendungslandschaften innerhalb der Unternehmen wurden in den letzten Jahren zahlreiche Integrationskonzepte und Implementierungen vorgestellt, die die Zusammenarbeit der bestehenden Systeme unterstützen.
In diesem Zusammenhang hat sich schließlich der Aufbau einer komplexen *Integrationsinfrastruktur*, die die Anwendungen den betrieblichen Anforderungen entsprechend miteinander verbindet, für die Unternehmen als wichtige Aufgabe herausgestellt.
Für den Aufbau einer solchen Integrationsinfrastruktur haben sich dazu in den letzten Jahren zwei Ansätze als besonders vielversprechend herauskristallisiert.
Der eine Ansatz basiert auf der asynchronen Übermittlung von Nachrichten und Empfängern, die diesen Nachrichten zugeordnet sind und hat unter anderem zu der Entwicklung der umfangreichen *Messaging-Systemen*, wie sie heute eingesetzt werden, geführt. Diese Systeme zeichnen sich durch eine erhöhte Dezentralisierung der Infrastruktur aus, da die Bestandteile der Messaging Systeme verteilt zusammenarbeiten.
Auf der anderen Seite werden inzwischen aber auch die *Workflowanwendungen* in dem Bereich der Anwendungsintegration mit Erfolg und steigender Verbreitung eingesetzt. Mit diesen Systemen wird die Integration, aufbauend auf einer interpretierten Prozessdefinition, zentral durchgeführt.
Nachdem sich herausgestellt hat, dass die beiden Lösungen an vielen Stellen synonym füreinander eingesetzt werden können und dabei zudem auch sehr komplexe Abläufe abgebildet werden können, klärt diese Arbeit nun, wie effizient diese Lösungen eine konkrete Problemstellung bearbeiten.
Die in dieser Arbeit vertieft betrachtete Problemstellung betrifft dabei den *sequentiellen Aufruf* von zu integrierenden Diensten und kann sowohl als Workflow modelliert, als auch als „Pipes and Filter“-Architektur im Zusammenhang mit Messaging Systemen implementiert werden.
Zu den dazu entwickelten Implementierungen werden in dieser Arbeit zunächst theoretische Überlegungen zu dem *Ressourcenverbrauch* angestellt. Danach werden zur Überprüfung dieser Überlegung Szenarien aufgestellt, mit denen der tatsächliche Ressourcenverbrauch der jeweiligen Implementierung festgestellt werden kann, so dass darauf aufbauend schließlich eine praktische .Analyse erfolgt.
Neben der Implementierung der Anwendungen, die die Integration leisten, wurde deshalb im Rahmen dieser Arbeit auch ein Messwerkzeug, inklusive der Konzeption der verschiedenen Messpunkte, entwickelt.

2 Theoretische Analyse der beiden Lösungskonzepte

In diesem Kapitel werden die beiden Lösungskonzepte vorgestellt und Ihre für diese Arbeit relevanten Eigenschaften näher betrachtet.
Dazu wird zuerst in dem Abschnitt 2.1 das Einsatzgebiet, in dem die beiden Architekturen miteinander verglichen werden, definiert.
In den nächsten beiden Abschnitten 2.2 und 2.3 werden die beiden Architekturen vorgestellt und auf ihre Effektivität und ihre Effizienz hin untersucht.
In dem letzten Abschnitt 2.4 erfolgt dann eine abschließende Gegenüberstellung.

2.1 Einschränkung des betrachteten Einsatzgebiet

Um einen aussagekräftigen Vergleich über die Effizienz unterschiedlicher Architekturen aufstellen zu können, ist es notwendig ein Einsatzgebiet zu finden, in dem die Architekturen sinnvoll und gegeneinander austauschbar eingesetzt werden können.
Neben den identischen Rahmenbedingungen für die Effizienzanalyse bedarf es daher eines Einsatzgebietes, das so gewählt wurde, dass durch diese Wahl nicht bereits ein Vorteil für eine Architektur entsteht.
Die Einführung in das Pipes and Filter Architekturmuster und die Einführung in die WfMS (Workflow Management Systeme) werden aber zeigen, dass sich die Einsatzgebiete der beiden Architekturen zu einem großen Teil unterscheiden.
So ist auf der einen Seite das Pipes and Filter Muster, als Konzept zur strukturierten Problemlösung, vielseitig einsetzbar und auf Problemklassen anwendbar, die über das Einsatzgebiet eines WfMS weit hinausreichen, da es beispielsweise im Bereich der Betriebssysteme oder auch der Grafikverarbeitung eingesetzt werden kann.
Auf der anderen Seite bietet aber auch die WfMS in Ihrem Einsatzgebiet, also der Abbildung von Prozessen und der damit einhergehenden Integration von Anwendungen, Funktionalitäten, die das Leistungsspektrum des Pipes and Filter Musters weit übersteigen wie zum Beipiel die Möglichkeit zur Inspektion der gerade laufenden Prozesse.
Bei der Integration heterogener Anwendungen, der so genannten EAI (Enterprise Application Integration) gibt es allerdings ein Einsatzgebiet in dem beide Architekturen synonym für eine Problemlösung eingesetzt werden können.
Dieses Einsatzgebiet ist weiterhin dadurch gekennzeichnet, dass die Integrationsschritte automatisiert, also ohne weitere manuelle Eingaben, abgearbeitet werden können. Sie werden entsprechend der Literatur [1] im Folgenden kurzlaufende Workflows genannt.

Aufbauend auf einem Message basierten System kann für solch einen kurzlaufenden Workflow eine Pipes and Filter Architektur aufgebaut werden, mit der die Nachrichten bearbeitet und an die zu integrierenden Dienste weitergeleitet werden.
Bei dem Einsatz eines WfMS kann solch ein kurzlaufender Workflow durch einen so genannten Macroflow oder aber auch durch einen Microflow abgebildet werden, der ebenfalls diese Nachrichtenbearbeitung und die anschließende Weiterleitung an die zu integrierenden Dienste durchführt.
Da das Ziel dieser Arbeit ein Effizienzvergleich der beiden Architekturen im Umfeld der Unternehmensintegration ist, findet die Durchführung der Effizienzanalyse der beiden Architekturen in diesem begrenzten Einsatzgebiet des kurzlaufenden Workflows statt.

2.2 Workflow-Management-Systeme

Um die Betrachtung der WfMS einzuleiten erfolgt zuerst eine kurze Klassifizierung dieser Systeme in Kapitel 2.2.1.
In Abschnitt 2.2.2 wird der konzeptionelle Aufbau und die Arbeitsweise der Systeme vorgestellt.
Da das betrachtete Einsatzgebiet eingeschränkt worden ist, erfolgt in Abschnitt 2.2.3 auch eine Einschränkung des Einsatzes der betrachteten Architektur.
In Abschnitt 2.2.4 wird dann die Effizienz der Problemlösung betrachtet.

2.2.1 Einführung

Das WfMS zählt zu den Groupware-Systemen und es gibt zahlreiche Produkte bekannter Hersteller, die der Kategorie der WfMS zugeordnet werden. Beispiele für solche Produkte sind Vitria BusinessWare[2], IBM Webspehere [3] oder auch Microsoft Biztalk[4].
Das Ziel des WfMS ist die Unterstützung von Arbeitsschritten über verschiedene Applikationen und Benutzer hinweg.
So werden in der Praxis mit diesen System die Wertschöpfungsketten eines Unternehmens abgebildet und dann (unter Einbeziehung aller beteiligten Applikationen) durch das WfMS für jeden Geschäftsvorfall wiederholt durchlaufen. Dies ermöglicht schließlich die koordinierte und integrierte Bearbeitung aller abgebildeten Geschäftsvorfälle.
Die Erfüllung dieser Aufgabe wird durch ein zweistufiges Vorgehen gewährleistet. Zuerst werden die Prozesse und die Infrastruktur analysiert und modelliert. Dabei werden die Abfolgen, die beteiligten Applikationen und die Benutzer festgestellt und in Beziehung zueinander gesetzt. Im nächsten Schritte werden dann die Prozessmodelle auf die Anwendungsserver aufgespielt, und bei Bedarf schließlich ausgeführt und überwacht.

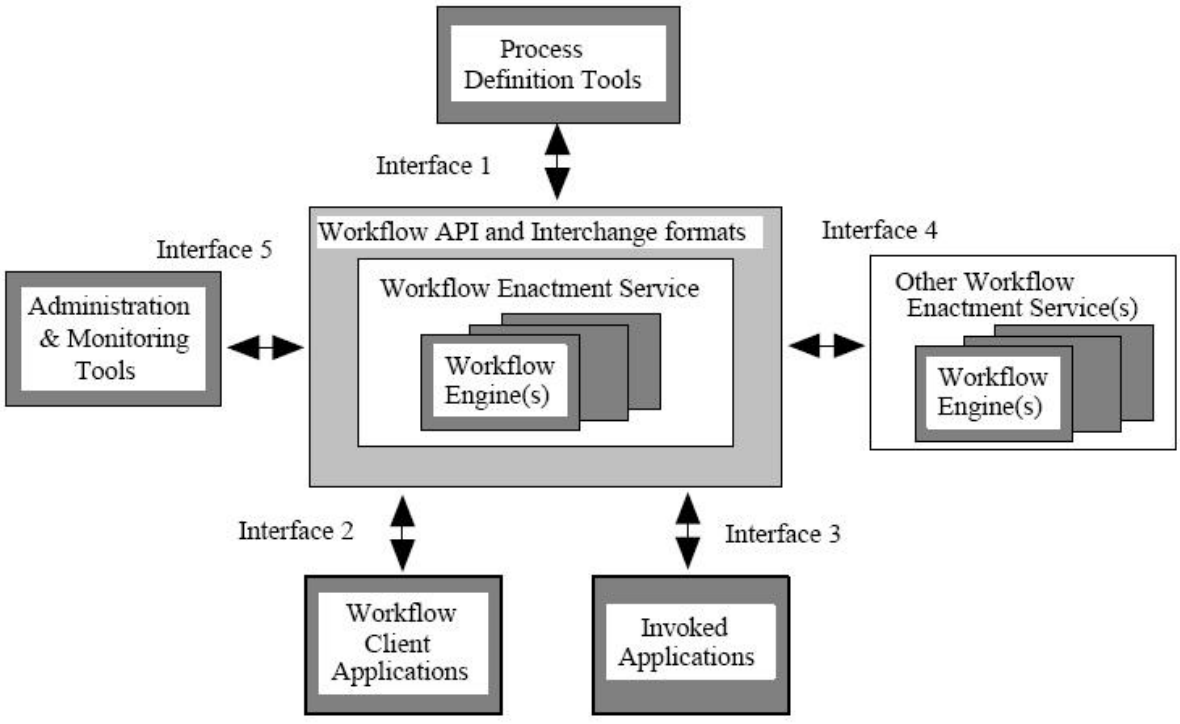
Die Möglichkeit zur Koordination von Arbeitsschritten und zur Vermeidung von Medienbrüchen hat schließlich zu einer weiten Verbreitung und schrittweisen Weiterentwicklung dieser Systeme geführt.

2.2.2 Architektur

Dem Referenzmodell der WfMC (Workflow Management Coalition) folgend sind für die Ausführung von Workflows durch eine WfMS bis zu sechs Komponenten notwendig (siehe hierzu auch [5]).

- Process Definition Tool - Modellierungswerkzeug
- Workflow Enactment Service - Container zur Ausführung von Workflows
- Administration und Monitoring Tools - Werkzeuge zur Verwaltung und Überwachung
- Workflow Client Applications - Grafische Schnittstellen für die Benutzer
- Invoked Applications - Extern aufgerufene Programme
- Other Workflow Enactment Services - Kooperierende Container zur Ausführung von Workflows

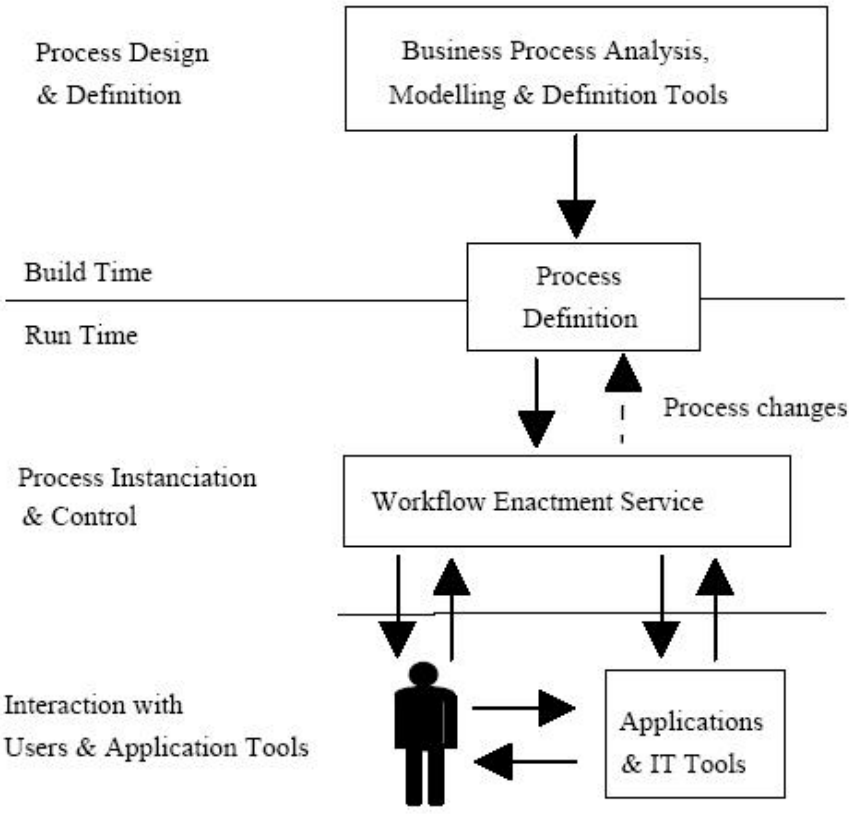
Diese Komponenten finden sich auch so oder so ähnlich im praktischen Einsatz wieder und werden zusammengenommen als WfMS vermarktet. Siehe hierzu 1 - Bestandteile eines Workflow Management System (Quelle [5]).



1 - Bestandteile eines Workflow Management System (Quelle [5])

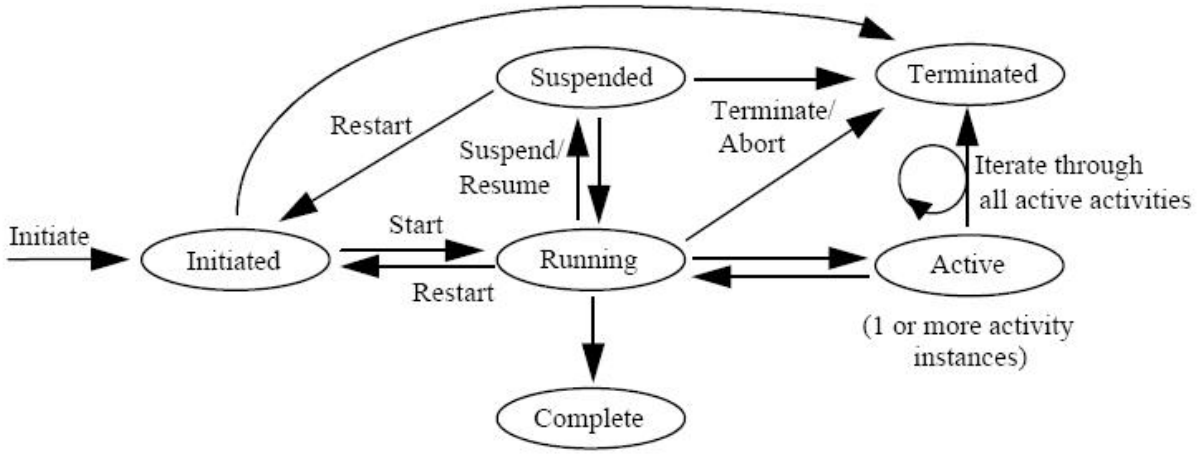
Diese Komponenten werden in verschiedenen Phasen eingesetzt.
Zuerst wird mit einem Modellierungswerkzeug in der **Definitionsphase** das Template für einen Prozess definiert.

Solch ein Template enthält neben allen relevanten Informationen zu den einzelnen Aktivitäten auch die Abfolge der Aktivitäten, inklusive der Ausführungsbedingungen, so dass ein vollständiger Ablaufgraf entsteht.
Als Sprache zur Definition solcher Templates hat sich BPEL (Business Process Execution Language) als Standard herauskristallisiert. Eine aktuelle Version der Spezifikation befindet sich bei [6].
Auf den Templates aufbauend werden von dem Workflow Enactment Service zur **Laufzeit** Instanzen für die aktuellen Geschäftsvorfälle gebildet.
Diese Instanzen werden nun von den Client Anwendungen abgefragt. Sie ermöglichen es den Benutzern der WfMS mit dieser zu interagieren, um beispielsweise die extern eingebundenen (und damit in einen größeren Ablauf integrierten) Anwendungen zu bedienen.
Zudem existieren im **Umfeld** der WfMS zur Laufzeit noch die Administrationsanwendungen, die die Auswertungsschnittstellen bedienen, sowie eventuell andere WfMS, die über eine Kollaborationsschnittstelle angebunden sind.
Die beschriebene Verwendung der einzelnen Komponenten wird noch einmal in Abbildung 2 - Definition und Laufzeit eines Workflows (Quelle [5]) verdeutlicht.



2 - Definition und Laufzeit eines Workflows (Quelle [5])

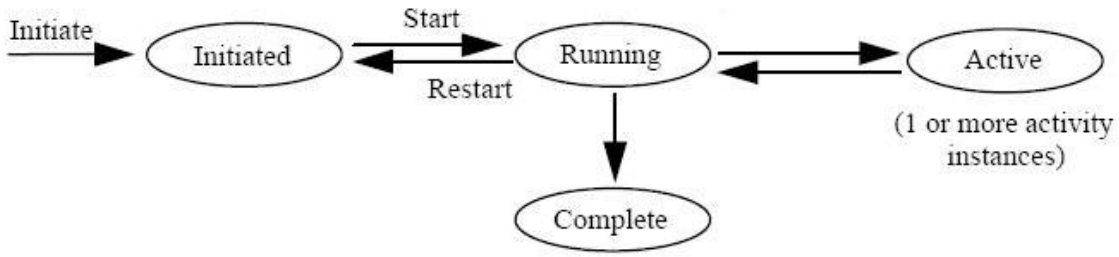
Da in dieser Arbeit die Laufzeiteigenschaften der WfMS näher betrachtet werden, konzentriert sich diese Einführung in die Architektur der WfMS hauptsächlich auf den Kern der WfMS, also die Komponente **Workflow Enactment Service**, da diese die Ausführung der Verarbeitungsschritte zur Laufzeit startet und ihren Ablauf überwacht.
Dem Workflow Enactment Service ist einerseits die Workflow Engine selber zugeordnet, der letztendlich die Kontrolle über die Prozessinstanzen unterliegt.
Andererseits gehören zur Funktion dieses Services aber auch die Verwaltung der Daten zu den Prozessinstanzen und deren Prozessaktivitäten.
Der **Workflow Engine** als Teil des Enactment Service unterliegt die Navigation durch die zuvor beschriebenen Ablaufgrafen und damit letztendlich die Durchführung und Bearbeitung der Prozessinstanzen.
Sie ist es, die bei einem Durchlauf durch den Ablaufgrafen anhand der vorliegenden Parameter entscheidet, welche Wege ausgewählt, beziehungsweise welche Aktivitäten aufgerufen werden.
Die Workflow Engine verfügt dabei prinzipiell über zwei Funktionsweisen.
Sie kann zum einen langlaufende Workflows, die so genannten Macroflows, starten und überwachen, bei denen die Zwischenschritte gesichert werden.
Zum anderen unterstützt sie aber auch die kurz oben zusammengeschrieben laufenden Workflows, die so genannten Microflows, bei denen keine längere Unterbrechung des Ablaufs geschieht, so dass auch keine Zwischenschritte festgehalten werden müssen. Zu den konkreten Einsatzmöglichkeiten und der Arbeitsweise solch einer Workflow Engine siehe auch [7].
Ein **Macroflow** ermöglicht die Durchführung von Prozessen, die aufgrund von angeforderten Benutzereingaben und asynchronen Dienstaufrufen über einen längeren Zeitraum aktiv sind.
Um dies zu gewährleisten werden vor und nach jedem Navigationsschritt alle Daten, die benötigt werden, um den Zustand vor dem Beginn der Durchführung in einem Fehlerfall wiederherstellen zu können, persistent gemacht.
Zudem geschehen die Aufrufe der beteiligten Dienste transaktional oder werden über speziell definierte Kompensationen geschützt, so dass die ACID-Anforderungen jederzeit eingehalten werden.
Die verschiedenen Zustände, die ein Macroflow während der Ausführung durchlaufen kann sind in Abbildung 3 - Zustandsübergangsdiagramm eines Macroflows (Quelle [5]) dargestellt.



3 - Zustandsübergangsdiagramm eines Macroflows (Quelle [5])

Der Macroflow bietet also, neben der Möglichkeit zur Unterbrechung vor allem eine sehr hohe Dienstqualität, da jeder zuvor erreichte Zustand bei einem nachfolgenden Fehler innerhalb der Aktivität wieder hergestellt wird. Um diese Dienstqualität zu erreichen, bedarf es aber auf der anderen Seite zahlreicher Maßnahmen, die sich zur Laufzeit negativ auf die Performanz auswirken.
Aus Überlegungen zur Steigerung eben dieser Performanz heraus wurde deshalb neben dem Konzept des Macroflow noch das Konzept des **Microflow** eingeführt, der eine schnellere, weil transiente, Informationsverarbeitung ermöglichen.
Ansatzpunkt der Überlegung war, dass es zahlreiche praktische Situationen gibt, in denen die Durchführung eines Workflows ohne Unterbrechung und damit ohne persistentes Festhalten der einzelnen Zwischenschritte geschehen kann. Solch ein Ablauf findet alleine im Hauptspeicher statt und folgt dem Alles-oder-Nichts-Prinzip. Das heißt, dass der Ablauf entweder terminiert oder keine Auswirkungen hat.
Um dies gewährleisten zu können, ist es notwendig, dass weder asynchrone Dienste aufgerufen werden noch Benutzeraktivitäten stattfinden, da diese beiden Aktivitäten das Festhalten von Zwischenschritten erfordern.
Werden aber andererseits nur synchrone Dienste aufgerufen, so kann die Verarbeitung wie beschrieben innerhalb einer Transaktion und innerhalb eines Threads ohne zwischenzeitliche Sicherung der Daten geschehen.
Diese Eigenschaft kann bei der Durchführung einer Sequenz von synchron angesprochenen Diensten mitunter sogar erwünscht sein, da nun entweder alle an der Sequenz beteiligten Dienste erfolgreich terminieren oder auf der anderen Seite keiner.
Im Fehlerfall bedeutet dies dann zwar, dass alle zwischenzeitlich vollzogenen Änderungen auch wieder rückgängig gemacht werden müssen. Dies ist aber

nicht problematisch, da davon keine Benutzereingaben betroffen sind und der Ablauf jederzeit neu gestartet werden kann.
Die praktische Umsetzung eines Mircroflows folgt dem Gedanken der Compile Sphere [1]. Eine Compile Sphere besteht aus einem Teil des Ablaufgraphen, der in ein aufrufbares Stück Software kompiliert wurde. Dieses Stück Ablauflogik kann dann von der Workflow Engine zur Laufzeit parametrisiert aufgerufen werden und arbeitet dann entweder alle geforderten Aktivitäten komplett ab oder es führt keine Änderung durch.
Der Graf zur Durchführung solch eines Ablauf benötigt dem folgend die Zustände ‚Suspended' und ‚Terminated' nicht mehr. Siehe hierzu Abbildung 4 - Zustandsübergangsdiagramm eines Microflows (Quelle [5]).



4 - Zustandsübergangsdiagramm eines Microflows (Quelle [5])

2.2.3 Einschränkung des betrachteten Einsatzgebiets

In dem Abschnitt 2.1 wurde bereits angesprochen, dass für die Effizienzanalyse Situationen gewählt werden müssen, in denen ein aussagekräftiger Vergleich zwischen dem Pipes and Filter Muster und der WfMS möglich ist.
Daher erfolgt eine Einschränkung bezüglich der genutzten Funktionalität der WfMS.
Die vielfältigen Möglichkeiten der WfMS die Workflows zu inspizieren und zu administrieren, werden nicht näher betrachtet beziehungsweise verwendet, da sie für die Effizienzanalyse nicht relevant sind und sie innerhalb der Pipeline auch nicht ohne Weiteres nachgebaut werden können.
Zudem erfolgt eine Einschränkung, die den Aufbau und die Art der Workflows betrifft.
Mit Hilfe der WfMS lassen sich komplexe Verarbeitungsgraphen zur Laufzeit aufbauen. Diese können im Folgenden auch dynamisch, also mit der Möglichkeit zur Pfadauswahl aufgrund von Eingabedaten, durchlaufen werden. Weiterhin bietet die WfMS auch noch die Möglichkeit diese Pfadauswahl von manuellen Eingabedaten abhängig zu machen, so dass letztendlich die unterschiedlichsten Prozessmodelle abgebildet werden können.
Für diese Arbeit werden im Folgenden dazu drei Ablaufgraphen unterschieden:

- Ablaufgraphen, die nur von den initialen Eingabedaten abhängen, ohne Verzweigungen
- Ablaufgraphen, die nur von den initialen Eingabedaten abhängen, mit Verzweigungen, die aufgrund dieser Daten ausgewählt werden
- Ablaufgraphen, die auch von Eingabedaten abhängen, die während des Prozessablablaufs aufgenommenen werden (Eingaben eines Benutzers zur Laufzeit) mit Verzweigungen, die aufgrund dieser Daten ausgewählt werden

Von Interesse für die Auswertungen sind dabei lediglich die Ablaufgraphen ohne Verzweigungen, die nur von den initialen Eingabedaten abhängen, da sie auch mit dem Pipes and Filter Muster ohne Weiteres abgebildet werden können.
Diese Art von Ablaufgraph bildet eine Sequenz ab, die mit einer WfMS sowohl über den Microflow als auch über den Macroflow abgebildet werden kann.
Zur Verwendung von Microflows und Macroflows in der Analyse ist zu sagen, dass der Microflow gut dazu geeignet ist, den beschriebenen Ablaufgrafen abzubilden, wenn er aus einer Sequenz von synchron aufgerufenen Diensten besteht.
Geschehen allerdings innerhalb des Ablaufs asynchrone Dienstaufrufe oder ist es erforderlich die Zwischenstände zu sichern, so kann nur ein Macroflow verwendet werden.
Da nun bei der PaF-Implementierung die Zustände (durch die verschiedenen Filterschritte und die Verwendung von Zwischenspeichern) aber implizit gegeben sind, wird der Fokus dieser Arbeit auf der Betrachtung des Macroflows liegen, um die Vergleichbarkeit sicherzustellen.
Auf der anderen Seite werden aber auch die Microflows als Konzept zur Beschleunigung des Ablaufs ebenfalls betrachtet werden, da die betrachteten Situationen einen Einsatz dieses Konzeptes nahe legen.

2.2.4 Effizienz

Der gesamte Ressourcenverbrauch zur Durchführung eines Workflows hängt im Wesentlichen von drei Dingen ab:

1. Ressourcenverbrauch des aufrufenden Clients
2. Ressourcenverbrauch des Workflows
3. Ressourcenverbrauch der externen Anwendungen inklusive der Nachrichtenübermittlung

Der Ressourcenverbrauch der externen Anwendungen sowie der Ressourcenverbrauch für die Kommunikation mit den externen Anwendungen sind dabei für die Untersuchung der Effizienz der WfMS nicht von näherem Interesse, da sie nicht direkt im Zusammenhang mit diesem System stehen und bei jeder Lösung anfallen, die die Anwendungsintegration durchführt, also auch bei einer Implementierung des Pipes and Filter Architekturmusters.

Gegenstand der Effizienzanalyse in dieser Arbeit ist daher der Ressourcenverbrauch durch den Workflow selber und der Ressourcenverbrauch durch den aufrufenden Client.
Der Ressourcenverbrauch für die externe Anwendung wird zwar ebenfalls in die Effizienzanalyse einfließen, da hier eine Anwendung aufgerufen wird, die in demselben Prozess läuft, aber auch die Implementierung des Pipes and Filter Architekturmusters benötigt diese Ressourcen, so dass die Vergleichbarkeit nicht gefährdet ist.
Bei der Implementierung der Workflows mit Hilfe eines Macroflows entsteht, hervorgerufen durch die Operationen, die für das Sicherstellen der Wiederherstellbarkeit einer Prozessinstanz, also der Umsetzung aller transaktionalen Gedanken ein Aufwand, der den Ressourcenverbrauch des Makroflows negativ belastet.
Auf der anderen Seite ist aber der Ressourcenverbrauch zur internen Weitergabe der Nachricht, also von dem abgeschlossenen Aufruf eines Dienstes bis zum Aufruf des nächsten Dienstes, gering, so dass hier ein Vorteil gegenüber verteilten Konzepten der Integration entsteht.
Eine wichtige Erkenntnis der Arbeit wird demnach sein, inwieweit sich der erhöhte Ressourcenverbrauch hervorgerufen durch Operationen, um die Dienstqualität sicherzustellen, im Vergleich zu dem verhältnismäßig geringen Ressourcenverbrauch hervorgerufen durch interne Nachrichtenübergaben, auswirkt.

2.3 Pipes and Filter Muster

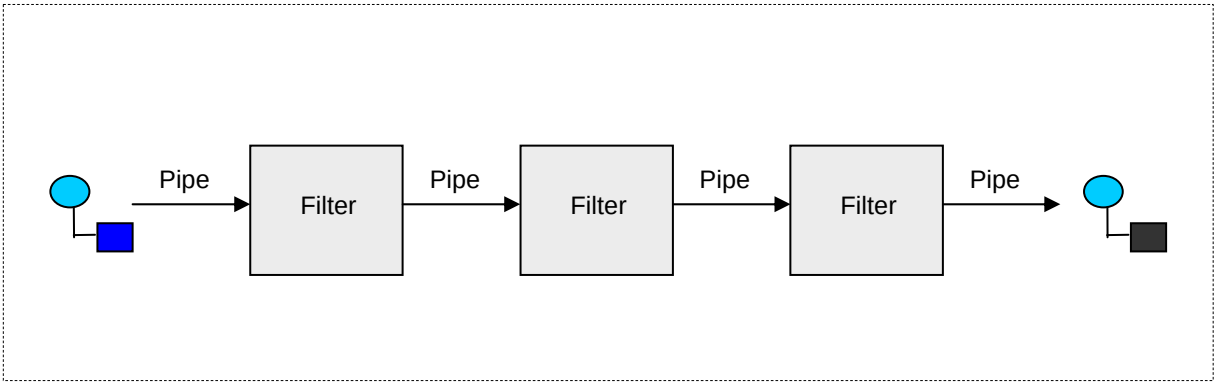
Wie zuvor bei der Einführung in das WfMS erfolgt auch bei der Betrachtung des Pipes and Filter Architekturmusters zunächst eine kurze Klassifizierung in Abschnitt2.3.1
In Abschnitt 2.3.2 wird dann der konzeptionelle Aufbau und die Arbeitsweise einer Implementierung vorgestellt.
Da das betrachtete Einsatzgebiet eingeschränkt worden ist, erfolgt in Abschnitt 2.3.3 auch eine Einschränkung des Einsatzes der betrachteten Architektur.
Schließlich wird in Abschnitt 2.3.4 die Effizienz der Problemlösung betrachtet.

2.3.1 Einführung

Unter dem Pipes und Filter Muster wird ein Architekturmuster zur Lösung einer bestimmten Problemklasse verstanden.
Mit dem Einsatz des Musters werden durch die Entkopplung von Verarbeitungsschritten im Kern zwei Dinge erreicht:

- Modularisierung der Funktionalität
- Beliebige (Re-)Kombination der modularisierten Funktionseinheiten

Durch die Modularisierung können die einzelnen Teile isoliert erweitert und getestet werden, während sie -bedingt durch die Möglichkeit zur Kombination- zudem in verschiedenen Sequenzen eingesetzt und je nach Bedarf zusammengestellt werden können.
Eine Implementierung dieses Musters eignet sich also dann besonders gut, wenn eine Sequenz von Verarbeitungsschritten flexibilisiert werden soll.
Die Erfüllung dieser Aufgabe wird durch zwei Komponenten ermöglicht. Zum einen gibt es dazu die Filter, die den Verarbeitungsschritten entsprechen. Zum anderen gibt es die Pipes, die die Übergabe von Daten zwischen den Filtern ermöglichen, so dass ein Filter die Daten für einen anderen Filter zur Verfügung stellen kann. Abbildung 5 – Konzept des „Pipes and Filter“-Architekturmusters verdeutlicht dieses Architekturmuster.



5 - Konzept des „Pipes and Filter“-Architekturmusters

Die Pipes und Filter inklusive der zwischen ihnen definierten Verbindungen ergeben zusammengenommen eine Pipeline.
Solch eine Pipeline kann dabei auf unterschiedliche Art implementiert werden.

Bei einem Einsatz innerhalb des Betriebssystems oder auch in der Grafikverarbeitung sind die Pipes als eine Sequenz von Speicherpuffern realisiert, die von den beteiligten Anwendungen, den Filtern, beschrieben, beziehungsweise ausgelesen werden.
Zur Initialisierung einer Pipeline dieser Art geschieht dabei einerseits die Instanziierung der Anwendungen (Filter), die einen bestimmten Speicherbereich als Quelle und einen anderen als Senke erhalten, sowie andererseits die Reservierung der Ihnen zugeordneten Speicherbereiche (Pipes).
Zur Laufzeit werden dann von jeder Anwendung (Filter) die Daten aus dem Ihr zugeordneten Eingabepuffer (Pipe) ausgelesen, verarbeitet und schließlich in den Ausgabepuffer (Filter) geschrieben, so dass ein sequentielles und effizientes Verarbeiten von Datenströmen ermöglicht wird.

Diese Form der Verarbeitung wird beispielsweise in dem Unix Betriebssytem häufig verwendet und kann unter anderem bei [8] genauer nachvollzogen werden.
Beim Einsatz dieses Architekturmusters in der Anwendungsintegration sind die Pipes als Nachrichtenspeicher realisiert, die über eine Identität verfügen und verteilt angesprochen werden können. Diese Realisierung kann beispielsweise mit Hilfe einer MOM (Message Oriented Middleware) erfolgen. Die Filter repräsentieren hierbei wiederum die Anwendungen, die die ihnen zugeordneten Nachrichtenspeicher kennen und dort ihre Eingaben abrufen bzw. ihre Ausgaben dorthin übermitteln.
Auch wenn in diesem Fall dasselbe Architekturmuster, wie in den Betriebssystemen beziehungsweise der Grafikverarbeitung verwendet wird, so ergeben sich doch zahlreiche Unterschiede in der Implementierung.
Neben der Realisierung der Pipes unterscheidet sich nämlich auch die Form der Daten grundlegend, da keine Datenströme, sondern vollständige Objekten sequentiell verarbeitet werden.
Das Pipes und Filter Architekturmuster ist also polymorph einsetzbar und da der Wunsch nach einer Strukturierung und Entkoppelung der Datenverarbeitung oft auftritt, wird es auch in verschiedenen Situationen implementiert.
Eine Taxonomie, die die Implementierungen des Musters klassifiziert, ist für diese Arbeit deshalb unabdingbar und wird in Abschnitt 2.3.2 aufgestellt. Auf dieser Taxonomie aufbauend kann dann in Abschnitt 2.3.3 die Gestalt der in dieser Arbeit genauer betrachteten Pipeline definiert werden.

2.3.2 Architektur

In diesem Kapitel erfolgt zunächst einmal die Klärung, welche Merkmale eine konkrete Pipeline kennzeichnet und welche Gestalt die Pipes und Filter als Bestandteile der Pipeline haben können. Dieser Teil schließt mit dem Aufbau einer Taxonomie, anhand derer eine Pipeline klassifiziert werden kann. Danach wird der Aspekt der Parallelität der Informationsverarbeitung noch einmal besonders beleuchtet.
Eine **Pipeline** verarbeitet sequentiell Informationen und ist deswegen zuerst einmal durch die Struktur der Daten, die sie verarbeitet, gekennzeichnet. Hier gibt es die Möglichkeit der Datenweitergabe und Datenverarbeitung als

- Datenstrom - Die Pipes enthalten Datenströme und die Filter verarbeiten diese
- Objekt - Die Pipes enthalten vollständige Objekte und die Filter verarbeiten diese

Im Fall der Verarbeitung von Datenströmen ist es möglich, dass ein Objekt über die Pipes und Filter verteilt vorliegt und die Filter jeweils einzelne Teile (Blöcke oder Zeilen) dieses Objektes verarbeiten. Dies hat natürlich erhebliche

Auswirkungen auf die Effizienz, da damit im Gegensatz zu einer Pipeline, in der vollständige Objekte übergeben werden, eine erweiterte Form der Parallelität in der Informationsverarbeitung ermöglicht wird.
Die **Filter** folgen in Verbindung mit den Pipes exakt dem Eingabe-Verarbeitungs-Ausgabe-Prinzip, wobei die Pipes hier Quelle und Senke darstellen und die Filter die Verarbeitung leisten. Aus der Art, wie die Filter die Informationen erhalten, ergibt sich folgende Klassifizierung

- Aktiv - Der Filter holt die Daten von der Pipe, beziehungsweise gibt sie an die Pipe weiter
- Passiv - Der Filter wird von der Pipe aufgerufen

Die Verarbeitungsmöglichkeiten der Filter sind dabei prinzipiell unbegrenzt und lassen sich grob in die folgenden Bereiche einteilen.

- Anreicherung - Informationen werden durch externe Informationen erweitert
- Verfeinerung - Informationen werden verarbeitet
- Umwandlung - Informationen werden einem Schema gemäß transformiert

Da diese Schritte oft benötigt werden, werden die Filter unter anderem in den Betriebssystemen, der Grafikbearbeitung oder aber auch wie bei [9] aufgezeigt in der Anwendungsintegration eingesetzt.
Eine **Pipe** hat in dem Muster die Aufgabe, einen Filter mit Daten zu versorgen bzw. die bearbeiteten Daten eines Filters entgegenzunehmen, und ermöglicht so die sequentielle Informationsverarbeitung. Eine Pipe ist hierbei Quelle, wenn sie die erste Pipe ist, Senke, wenn sie die letzte Pipe ist und Quelle und Senke, wenn sie weder die letzte noch die erste Pipe ist. Die Pipe kann dabei als gemeinsam genutzter Speicherbereich, als Nachrichtenstapel oder jede andere Datenstruktur organisiert sein. Die Einschränkung hierbei ist jedoch, dass ein Filter, seine Eingabedaten von der Pipe abrufen bzw. empfangen und verarbeiten kann.
Eine Pipe kennzeichnet dabei die Unterscheidung nach

- Aktiv - Die Pipe verfügt selber über Funktionen zur Datenverwaltung
- Passiv - Die Pipe wird von den Filtern organisiert

beziehungsweise nach

- Synchron - Die Pipe gibt die Daten direkt weiter
- Asynchron - Die Pipe pollt und hält die Objekte bis zur Weitergabe

Es ist natürlich innerhalb einer Pipeline möglich,Pipes unterschiedlicher Ausprägung zu kombinieren.
Um die Frage nach dem Laufzeitverhalten einer Pipeline klären zu können, ist noch die **Instanziierung** der Pipelineelemente von Bedeutung. Hier gibt es folgende Möglichkeiten

- Filter

    - einfach - Filter werden exakt einmal instanziiert und laufen dann als eigenständiger Prozess (Beispiel: Dynamic Link Library oder auch Java-Servlet)
    - mehrfach - Filter werden für jede Pipeline neu instanziiert (Grafikverarbeitung und Betriebssystemprozesse)
- Pipes
    - einfach - Pipes werden exakt einmal instanziiert und laufen dann als eigenständiger Prozess (Beispiel: Message Queueing Systeme)
    - mehrfach - Pipes werden für jede Aufgabe neu instanziiert (Grafikverarbeitung und Betriebssystemprozesse)

Nachdem nun alle Bestandteile einer Implementierung, die dem Pipes and Filter Muster folgt, geklärt worden sind sowie deren Instanziierungsmöglichkeiten aufgezeigt wurden, kann folgende **Taxonomie** aufgestellt werden

- Eigenschaften der Pipeline
    - Pipelineinhalt - Datenstrom / Objekt
- Eigenschaften der Filter
    - Verhalten - Aktiv / Passiv
    - Aufgabe der Filter - Anreicherung / Verfeinerung / Umwandlung
    - Instanziierung - einfach / mehrfach
- Eigenschaften der Pipe
    - Verhalten - Aktiv / Passiv
    - Kommunikation - Synchron / Asynchron
    - Instanziierung - einfach / mehrfach

2.3.3 Einschränkung des betrachteten Einsatzgebiets

In dieser Arbeit werden Formen der Implementierung des Pipes and Filter Muster betrachtet, wie sie in der Anwendungsintegration eingesetzt werden [9].
Durch diese Einschränkung ergeben sich folgende Parameter für die Architektur der Pipeline

- Eigenschaften der Pipeline
    - Pipelineinhalt - Objekt
- Eigenschaften der Filter
    - Verhalten - Aktiv
    - Aufgabe der Filter - Anwendungsintegration
    - Instanziierung - einfach/mehrfach
- Eigenschaften der Pipe
    - Verhalten - Aktiv
    - Kommunikation - Asynchron
    - Instanziierung – einfach

Diese konkrete Ausgestaltung der Pipeline wird nun für die folgenden Betrachtungen und Analysen herangezogen.

Ihre Parameter ergeben sich dabei weitestgehend aus dem Einsatz einer MOM und dem damit einhergehenden Push-and-Pull-Mechanismus. So ruft eine Filterkomponente die eingegangenen Nachrichten von einem MOM Knoten ab und wird deshalb als aktiv klassifiziert. Die Pipekomponente ist ebenfalls aktiv, da sie die Nachrichtenverwaltung eigenständig durchführt und den Filtern Schnittstellen zum Abruf und zur Speicherung von Nachrichten zur Verfügung stellt.
Eine weitere wichtige Einschränkung in der Informationsverarbeitung durch eine Pipeline ist die nicht vorhandene dynamische Pfadauswahl zur Laufzeit, die das WfMS ermöglicht.
Sobald eine Verarbeitungssequenz für die Pipeline gestartet worden ist, werden die einzelnen Schritte sequenziell durchgeführt und es erfolgt kein weiteres Routing mehr.

2.3.4 Effizienz der Problemlösung

Die Organisation von Funktionalität als Pipeline führt zu besonderen Eigenschaften in der Informationsverarbeitung. So ist ein wichtiges Charakteristikum die **Parallelität** der Informationsverarbeitung, da die permanente Informationsweitergabe der Filter die gleichzeitige Verarbeitung mehrerer Eingaben ermöglicht. Je nach Ausprägung der Pipeline geschieht dabei jedoch die Parallelisierung der Bearbeitung auf unterschiedliche Art.

Auswirkungen des Pipelineinhalts auf die Möglichkeiten zur Parallelisierung

Besteht der Pipelineinhalt aus Datenströmen, so ist es möglich, dass mehrere Filter gleichzeitig an derselben Eingabe arbeiten und sie zudem auch noch stückweise aufgeteilt in den Pipes vorliegt. Somit findet eine parallele Verarbeitung eines Objekts innerhalb der Pipeline statt und es ist auf diese Weise möglich, dass ein Objekt die Pipeline auslastet.
Besteht der Pipelineinhalt hingegen aus vollständigen Objekten, so ist die gleichzeitige Bearbeitung desselben durch mehrere Filter nicht möglich. Ein Objekt befindet sich dann in (genau) einer Komponente, der Pipeline, so dass die Pipeline nur dann ausgelastet wird, wenn mehrere Objekte gleichzeitig in der Pipeline vorhanden sind.
Unabhängig davon, ob Objekte oder Datenströme durch die Pipeline fließen, kann also eine Pipeline ausgelastet werden - und dies ist auch die Situation, die in der praktischen Analyse herbeigeführt wird, da die oberen Grenzen der Auslastung für die Untersuchung relevant sind. In dem Fall der Auslastung bestimmt schließlich der langsamste Filter den gesamten Durchsatz der Pipeline.

Auswirkungen der Instanziierung auf die Möglichkeiten zur Parallelisierung

Bei einer einfachen Instanziierung der Pipes und der Filter ist eine Parallelisierung möglich, indem mehrere Eingaben die Pipeline gleichzeitig durchlaufen und die Filter jeweils an verschiedenen Eingaben arbeiten.
Im Falle einer mehrfachen Instanziierung der Pipes oder der Filter ergeben sich weitere Möglichkeiten zur Parallelisierung und zur Beseitigung von Flaschenhälsen innerhalb der Pipeline. Jetzt können nämlich für die langsamste Komponente - unabhängig davon, ob es eine überlastete Pipe, oder ein überlasteter Filter ist - neue Pipes und neue Filter instanziiert werden, die die zuvor überlastete Komponente entlasten und so den Durchsatz der Pipeline erhöhen. So kann durch Redundanz die Verarbeitung der Daten beschleunigt werden.

Einfluss der Parallelisierung auf die Effizienz

Mit den Möglichkeiten zur Parallelisierung lässt sich bei Bedarf der Durchsatz durch die Pipeline steigern, so dass die Pipeline anforderungsgerecht skaliert werden kann. Dies wird erreicht, indem der Pipeline mehr Ressourcen zugeordnet werden.
Der mit dieser Arbeit näher betrachtete Ressourcenverbrauch zur Informationsverarbeitung durch die Pipeline (exklusive der Dauer der Verarbeitung einer Eingabe) ist dadurch im besten Fall jedoch nicht beeinträchtigt. Das liegt daran, dass der Speicher- und Rechenzeitverbrauch der beteiligten Komponenten für die Bearbeitung eines Objektes innerhalb der Pipeline unverändert gegeben ist.
Die Erhöhung des Durchsatzes durch Redundanz hat auf den Ressourcenverbrauch für die Bearbeitung also zuerst einmal keine positive Auswirkung, kann aber auf der anderen Seite schnell zu einer Steigerung führen, wenn beispielsweise die Eingaben nun eindeutig gekennzeichnet und identifiziert werden müssen bzw. wenn die weiteren Instanzen einen Laufzeitoverhead erzeugen.
Ohne eine Beschränkung der Aussagekraft ist es dementsprechend möglich eine nicht extra auf Parallelisierung ausgelegte Pipeline zu untersuchen und ihren Ressourcenverbrauch festzustellen. Dies ist zulässig, da mit dieser Arbeit lediglich der optimale Ressourcenverbrauch des Konzepts untersucht wird.

Stärken und Schwächen der Pipeline:
Die Vorteile eines Einsatzes des Pipes and Filter Musters liegen bezüglich des Ressourcenverbrauchs zu einem großen Teil in der aus der Parallelisierung hervorgehenden Skalierbarkeit begründet.
Diese Skalierbarkeit ermöglicht eine Reduzierung der Durchlaufzeiten für die einzelnen Objekte, bedingt aber nicht eine effizientere Form der Informationsverarbeitung, so dass der Speicher und Rechenzeitverbrauch dadurch bestenfalls nicht negativ beeinflusst und die Parallelisierung in der Analyse daher auch nicht näher betrachtet wird.
Der Aufwand an benötigten Ressourcen durch den Einsatz einer Pipeline wird im Einzelnen durch folgende Punkte bestimmt

1. Ressourcenverbrauch der Pipes
2. Ressourcenverbrauch der Filter
3. Ressourcenverbrauch der Nachrichtenübermittlung zwischen den Pipes und den Filtern
4. Ressourcenverbrauch der externen Anwendungen inklusive der Nachrichtenübermittlung

Wie zuvor bei der Betrachtung zur Effizienz der WfMS in Abschnitt 2.2.4, ist der Ressourcenverbrauch der externen Anwendungen sowie der Ressourcenverbrauch für die Kommunikation mit den externen Anwendungen nicht von näherem Interesse.
Für den Fall, dass die Filter im praktischen Einsatz nicht auf demselben System wie die Pipes lokalisiert sind, sondern beispielsweise in der Nähe der aufgerufenen Anwendungen, würde sich eine Verschiebung bezüglich der Zeitdauern der Nachrichtenübermittlung (3. Ressourcenverbrauch und 4. Ressourcenverbrauch) ergeben. Diese wirkt sich jedoch im besten Fall nicht negativ auf die Gesamtdauer aus, da die Dauer der Übermittlung von der Pipe bis zur aufgerufenen Komponente durch einen Umweg nur verschlechtert werden kann. Zur Verdeutlichung dieses Punktes sei auf die sternförmige Kommunikation der WfMS verwiesen und die Tatsache, dass die kürzeste Verbindung zwischen zwei Punkten die Gerade ist.
Durch eine solche Verschiebung wäre es aber auf der anderen Seite notwendig, den 3. Ressourcenverbrauch sowohl für die WfMS als auch für die Pipeline zu betrachten, damit die Vergleichbarkeit gewährleistet bleibt.
Im Folgenden wird deshalb die optimale Pipeline betrachtet, bei der, dadurch dass alle Komponenten auf einem Rechner lokalisiert sind, durch die Filter keine zusätzlichen Umwege zwischen den Pipes und den aufgerufenen Anwendungen entstehen.
Eine Einschränkung bleibt bestehen, da es theoretisch möglich wäre, dass der Filter die Nachricht an die externe Anwendung erheblich vergrößert. In diesem Fall könnten Vorteile gegenüber dem WfMS entstehen, da hier die Nachricht an

die externe Anwendung nicht erst in der Nähe des Zielsystems zusammengebaut werden kann. Dieser Sonderfall wird allerdings nicht näher betrachtet.
Wie groß der Ressourcenverbrauch einer Pipeline nun tatsächlich ausfällt, hängt zum einen davon ab, wie gut die Pipes und die Filter implementiert sind, also wie effizient die Pipes den Speicher allokieren und Nachrichten empfangen und senden bzw. wie effizient die Filter die Nachrichten verarbeiten.
Zum anderen steht der Ressourcenverbrauch aber auch in einem unmittelbaren Zusammenhang mit der Konfiguration der Pipeline. Wenn der Nachrichtenabruf der Filter auf den Nachrichteneingang bei den Pipes nicht gut abgestimmt ist, können erhebliche Ineffizienzen entstehen.
Dadurch, dass in der Implementierung einer Pipeline zwei Komponenten arbeiten, nämlich die Pipes und Filter, die sich gegenseitig über asynchrone Nachrichten abstimmen müssen, werden Ressourcen zur Nachrichtenübergabe verbraucht, die nicht im direkten Zusammenhang mit der Lösung der Aufgabe stehen. Dieser Aufwand fällt beispielsweise bei den Workflows nicht an, da die beteiligten Anwendungen zentralisiert aus der WfMS heraus aufgerufen werden.
Der Aufwand zur Weitergabe der Informationen kann also als erster systemischer Nachteil der Pipes und Filter betrachtet werden. Der zweite ist sicherlich die Redundanz, da mehrere Komponenten parallel mit ihrem jeweils eigenen Overhead arbeiten.
Diese beiden Overheads (Informationsweitergabe und Redundanz) sind die zentrale Schwäche der Pipeline bei der Informationsverarbeitung. Je nachdem wie gut die Implementierung der Pipeline im Bezug auf das Abrufintervall konfiguriert ist, fällt er zudem unterschiedlich groß aus. Ungeeignet kurze, bzw. ungeeignet lange Intervalle im Bezug auf die Objektgröße und die Frequenz der Objekteingabe, werden die Pipeline als schlechte Wahl zur Lösung der Integrationsszenarien während der Analyse erscheinen lassen.
Mit einer für die objektbezogenen Parameter unpassend konfigurierten Pipeline bleibt nämlich dann entweder der Durchsatz (zu langes Abrufintervall) oder der Speicherverbrauch (zu kurzes Abrufintervall) hinter den theoretischen Möglichkeiten zurück.
Auch in dem Punkt der Gesamtlaufzeit sind dann Abweichungen zu erwarten, die sich aus der Asynchronität ergeben, da ein Nachrichtenabruf in ungeeignet langen Intervallen zu einer deutlich höheren Gesamtlaufzeit führen wird.

2.4 Gegenüberstellung der beiden Ansätze

2.4.1 Einsatzgebiete

Auch wenn die beiden Architekturen teilweise unterschiedliche Einsatzgebiete haben, so verfügen sie doch über eine Schnittmenge in der Problemlösungsfähigkeit.
Schließlich ist sowohl mit den kompilierten Mikroflows einer WfMS als auch mit der PaF-Implementierung eine sequentielle Abarbeitung der Eingangsinformation möglich.

Das **Pipes and Filter Muster** ist ein generisches Muster, um Probleme schrittweise zu lösen. Es wurde zu Beginn entwickelt, um den Umgang mit Datenströmen zu vereinfachen, weil es vorteilhaft war (unter anderem für die Wiederverwendung oder aber die flexible Kombination) die Schritte des Filterns zu entkoppeln.
Inzwischen hat sich das Muster aber auch in der Anwendungsintegration durchgesetzt, weil eben diese Entkopplung auch angestrebt wird, wenn eine Nachricht an eine bestimmte Menge von Anwendungen übergeben werden soll.
In diesem Fall wird eine Pipeline definiert, in der die Filter sequentiell aufgerufen werden.

Die **WfMS** aber auf der anderen Seite wurde speziell im Hinblick auf die Anwendungsintegration entwickelt.
Sie bietet deswegen im Bereich der Anwendungsintegration sehr viel weiter reichende Möglichkeiten der Informationsverarbeitung, als sie mit dem Pipes and Filter Muster möglich wären. So kann in einer WfMS beispielsweise die Verarbeitungsfolge mitunter erst zur Laufzeit dynamisch bestimmt und in Form eines Ablaufgraphen durchlaufen werden.
Trotzdem ist es natürlich auch mit einer WfMS möglich, eine vorher festgelegte Sequenz von Schritten zu durchlaufen.
Diese Sequenz ist aus der Sicht der WfMS ein kurz laufender Prozess und wird entweder als Macroflow oder als Microflow durchgeführt.
Eben diese strukturierte Form der Problembearbeitung ist nun der Ansatzpunkt für den Vergleich, so dass im folgenden statisch festgelegte Sequenzen von Problemlösungsschritten betrachtet werden, wie sie in der Anwendungs-integration häufig vorkommen.
Dies bedeutet zwar auf der einen Seite für beide Architekturen eine Reduktion ihrer Fähigkeiten auf einen bestimmtes Einsatzgebiet und einen bestimmten Einsatzkontext hin, ermöglicht aber auf der anderen Seite auch einen aussagekräftigen Vergleich.

2.4.2 Effizienz

Bei der Gegenüberstellung der beiden Lösungsansätze werden die folgenden Ressourcenverbräuche nicht näher betrachtet, da sie jeweils identisch sind und dementsprechend vernachlässigt werden können.

- Ressourcenverbrauch der externen Anwendungen

Folgende Ressourcenverbräuche sind für diese Arbeit von Interesse und werden deswegen erfasst werden.

- Ressourcenverbrauch des Microflows
- Ressourcenverbrauch des Microflows
- Ressourcenverbrauch der PaF-Implementierung (inklusive dem Abstimmungsaufwand zwischen den Pipes und den Filtern)

Die erwarteten Ergebnisse in diesem Zusammenhang ergeben sich bereits teilweise aus der durchgeführten theoretischen Analyse in Abschnitt 2.2.4 für die WfMS und Abschnitt 2.3.4 für die Implementierung des Pipes and Filter Architekturmusters.
Die Abweichungen zum theoretisch optimalen Ressourcenverbrauch ergeben sich bei der Lösung durch die WfMS vor allem durch die Operationen zur Sicherstellung der Dienstqualität.
Bei der PaF-gestützten Implementierung bedingt die Abweichung zum theoretisch optimalen Ressourcenverbrauch, dass zwei Komponenten in gegenseitiger Abstimmung arbeiten. Die Aufteilung der Nachrichtenspeicherung in Pipes und der Nachrichtenweitergabe in Filter bedingt zunächst einmal einen erhöhten Aufwand, hervorgerufen durch die Notwendigkeit zur Abstimmung und Nachrichtenweitergabe innerhalb der Pipeline. Durch eine ungeeignete Konfiguration bezüglich der Nachrichtenabrufe kann diese Zusammenarbeit extrem ineffizient und das Konzept nachteilig werden. In jedem Fall bleibt aber auch bei einer geeigneten Konfiguration die nachteilige Redundanz bestehen, da an der Bearbeitung einer Eingabe immer mehrere Komponenten beteiligt sind.
Inwieweit sich die genannten Charakteristika im praktischen Vergleich negativ auf den jeweiligen Ressourcenverbrauch auswirken, wird die szenariobezogene Analyse des Ressourcenverbrauchs zeigen.

3 Szenarien für die Effizienzanalyse

Wie in 2.1 festgestellt gibt es für die beiden Architekturen ein Einsatzgebiet, in denen eine sinnvolle und vergleichbare Analyse der jeweiligen Effizienz durchgeführt werden kann. Dieses Einsatzgebiet betrifft den sequentiellen Aufruf von Diensten mit dem Ziel der Anwendungsintegration.
Auch wenn sich dabei für die beiden Architekturen in diesem Gebiet die Entwicklungsgeschichte, das Systemumfeld sowie die Struktur der Lösung selber stark unterscheiden, ist es möglich Szenarien zu finden, in denen jeweils beide Architekturen zur Umsetzung der Anforderungen eingesetzt werden können.
Um diese Szenarien grob gliedern zu können, werden in Abschnitt 3.1 die Ziele der Szenarien festgehalten.
Die in Abschnitt 3.2 in der Liste aufgestellten Szenarien, sind dann zusammen mit den Implementierungen, die sie lösen, der Ausgangspunkt für die später durchgeführte Effizienzanalyse.

3.1 Ziel der Szenarien

Das Ziel, das mit dem Aufbau der Szenarien verfolgt wird, ist die Überprüfung des Ressourcenverbrauchs der beiden Integrationsarchitekturen unter unterschiedlichen Rahmenbedingungen.
Dazu werden Szenarien aufgestellt, die sich in ihrem grundlegenden Ablauf nicht unterscheiden, da jeweils die einzelnen Integrationsschritte, also die Aufrufe der zu integrierenden Dienste, im Mittelpunkt stehen. So ist das Ziel eines jeden Szenarios der sequentielle Aufruf von Diensten, denen eine Nachricht übergeben wird, die beispielsweise die Informationen eines Geschäftsvorgangs repräsentiert. Ein in diesem Zusammenhang angesprochener Dienst wird im folgenden Integrationsschnittstelle genannt.
Die Szenarien sind dabei darauf ausgelegt, dass sie nicht nur einmal von der jeweiligen Implementierung abgearbeitet werden, sondern in Massenläufen durchgeführt werden, so dass ein Szenario den wiederholten Aufruf der Implementierung und damit aller beteiligten Dienste mit einschließt.
Da die Szenarien insbesondere der Feststellung des Ressourcenverbrauchs einer Integrationsarchitektur dienen, fließen die folgenden Rahmenbedingungen in den Szenarioaufbau ein:

- Dienstqualität
- Größe der Nachrichten
- Anzahl der Filterschritte

Die Dienstqualität kann dabei zum einen die vollständige Wiederherstellbarkeit garantieren, indem alle Zwischenschritte persistent festgehalten werden. Dies ist

bei den Workflows im Fall der Macroflows der Fall und bei der PaF Implementierung, nicht jedoch bei den Microflows.
Mit der Größe der Nachrichten wird überprüft, inwieweit der Ressourcenverbrauch der Integrationsarchitektur von der Größe der eingehenden initialen Nachricht abhängt. Als kleine Nachricht wird eine Nachricht mit einer Größe kleiner als ein Kilobyte angesehen, während eine große Nachricht größer als 50 Kilobyte ist.
Mit der Anzahl der Filterschritte wird überprüft, inwieweit eine Erhöhung oder Verringerung der angesprochenen Integrationsschnittstellen den Ressourcenverbrauch beeinflusst.
Interessant bei der Betrachtung des Ressourcenverbrauchs unter diesen veränderten Rahmenbedingungen ist nicht nur die Frage, ob der Ressourcenverbrauch wächst, sondern auch ob dies überproportional geschieht.

3.2 Liste der Szenarien

Aus den in 3.1 definierten Rahmenbedingungen Dienstqualität, Anzahl der Filterschritte und Größe der Nachrichten, wird nun eine Liste von Szenarien erstellt (siehe 1 - Auflistung der Szenarios)
Für jeden Eintrag in dieser Liste wird dann sowohl in Form eines Workflows als auch in Form einer Pipeline eine Implementierung erstellt, um den jeweiligen Ressourcenverbrauch festzustellen.

Nr.	Dienstqualität	Anzahl der Filter	Nachrichtengröße (in Kb)
1	persistent / transaktional	1	1
2	persistent / transaktional	1	50
3	persistent / transaktional	4	1
4	persistent / transaktional	4	25
5	persistent / transaktional	4	50
6	persistent / transaktional	8	50
7	transient	1	1
8	transient	1	50
9	transient	4	1
10	transient	4	25
11	transient	4	50
12	transient	8	50

1 - Auflistung der Szenarios

Außerdem werden noch zwei weitere Implementierungen betrachtet.
Da die PaF-Implementierung, wie später noch beschrieben auf JMS basiert werden die Pipes als Queues auf einem Anwendungsserver realisiert.
In der erweiterten Analyse wird deshalb zum einen der Ressourcenverbrauch einer PaF Implementierung überprüft, die nicht auf Queues, sondern auf Topics basiert. Hierzu wird ansonsten das Szenario Nummer 5 verwendet.
Zum anderen werden die Auswirkungen der Parallelität sowohl innerhalb des Microflows als auch innerhalb der PaF Implementierung exemplarisch untersucht. Hierzu wird das Szenario 8 verwendet

4 Versuchsaufbau und Anwendungskonzeption

In diesem Kapitel werden die Implementierungen näher beschrieben, die die Szenarien aus 3.2 lösen.
Dazu wird in 4.1 die benutzte Hard- und Softwareumgebung näher beschrieben.
In 4.2 werden die einzelnen Anwendungskomponenten vorgestellt.

4.1 Hard- und Softwareumgebung

Für die Realisierung der Szenarien wird als Prozessor ein Intel Pentium 4 Prozessor mit Dual Core Technologie und 3,4GHz eingesetzt.
Die Tatsache, dass der Prozessor virtuell zwei Mal vorhanden ist, wird innerhalb des Messwerkzeuges kompensiert werden.
Der Rechner verfügt über 2GB Arbeitsspeicher und nutzt als Betriebssystem Microsoft Windows XP mit Service Pack 2.
Als Entwicklungsumgebung für die in den verschiedenen Szenarioimplementierungen verwendete Software wird die WebSphere Studio Application Developer Integration Edition (WSAD IE) in der Version 5.1 eingesetzt.
Die Anwendungen werden dabei mit dem JDK 1.4.2 von IBM kompiliert, auf der auch der integrierte WebSphere Testserver läuft, der die Anwendungen hostet.
Aus dieser Umgebung folgt, dass die entwickelten Lösungen als Enterprise Java Beans (EJB's) auf einem Anwendungsserver laufen.
Der Ressourcenverbrauch des Anwendungsserver verfälscht hierbei die Analyseergebnisse nicht, da der entstehende Overhead für beide Architekturen anfällt und auch keine besonderen Features des Servers zusätzlich benutzt werden.

4.2 Anwendungskomponenten

Allgemeine Dienste

In den Implementierungen existieren zwei gemeinsam genutzte Anwendungen, die sowohl von der PaF-Implementierung, als auch von der WfMS-Implementierung genutzt werden.

- ProfilingEAR
- SimpleServiceEAR

Diese beiden Dienste werden jeweils über die lokale Schnittstelle mit einer geringen Verzögerung angesprochen.

Dieser Aufruf ist deshalb so effizient, weil eine über die lokale Schnittstelle instanziierte EJB in demselben Kontext wie die aufrufende EJB läuft und die Objekte als Referenzen übergeben werden können.
Eine prozessübergreifende Kommunikation hingegen, beispielsweise unter Verwendung des SOAP-Protokolls, verlangsamt sowohl die Workflow-Implementierung als auch die PaF-Implementierung deutlich, so dass auf diese Form der Kommunikation verzichtet wird.
Die ProfilingEAR enthält das Messwerkzeug, wie es in 5.2.3 näher beschrieben wird und liefert die Messdaten zur Auslastung beziehungsweise zum Ressourcenverbrach der zu untersuchenden Implementierung.
Die Anwendung SimpleServiceEAR enthält selber keinerlei Funktionalität, wird jedoch in den einzelnen Aufrufen durch die zu untersuchende Implementierung angesprochen und fungiert somit als zu integrierender Dienst.
Die Funktionalität dieses Dienstes ist nicht von Interesse, da sie für den Vergleich der beiden Integrationsarchitekturen unerheblich ist und der in ihr entstehende Ressourcenverbrauch in beiden Fällen identisch ist.

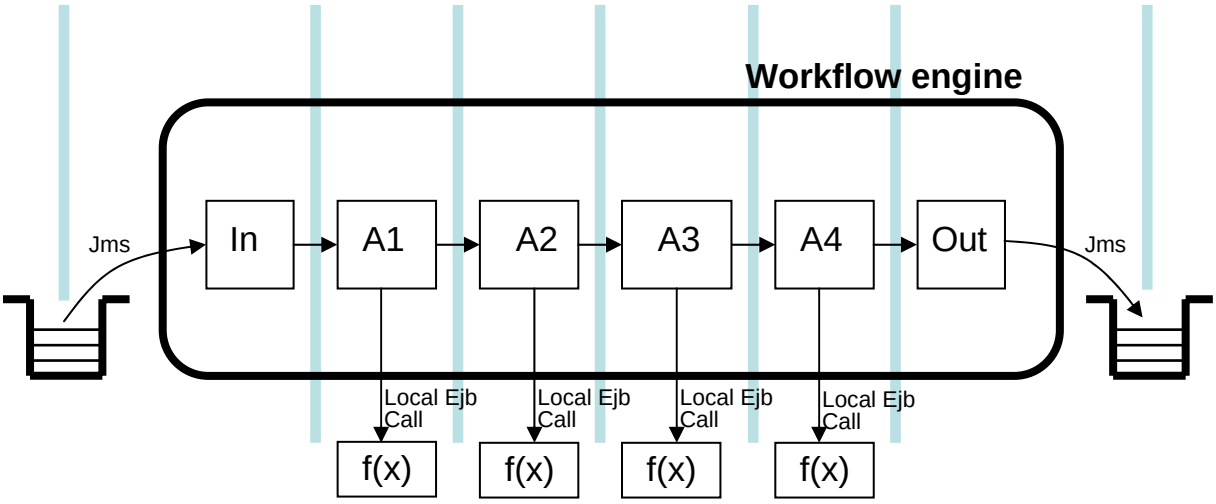
Integrationsdienste
Die Implementierungen des Macroflows und des Microflows sowie die Implementierung, die dem PaF Muster genügt, sind im Folgenden aufgeführt:

- MDB-Pipeline
- WorkflowImplementationEAR (Microflow)
- WorkflowImplementationEAR (Macroflow)

Diese Anwendungen führen sowohl den wiederholten Aufruf des zu integrierenden Dienstes als auch den Aufruf des Messwerkzeuges durch.
Alle drei Anwendungen verfügen dabei über eine Java Messaging Service (JMS) Schnittstelle, während die WorkflowImplementationEAR, die auf einem Microflow basiert zudem einen lokalen Aufruf unterstützt.

WorkflowImplementationEAR (Macroflow)
Diese Workflow-Implementierung basiert auf einer Prozessdefinition in BPEL [6].
In der BPEL Datei sind unter anderem die Prozessschnittstellen, der wiederholten Aufruf der Anwendung SimpleService sowie zwei Java-Snippets, definiert. Das Konzept der Java-Snippets ermöglicht den Einschub von Java Code in den Prozessablauf und wurde verwandt, um das Festhalten von Messdaten zur internen Verarbeitungsdauer anzuweisen. Der Wert „longRunning“ des Attributs executionMode zeigt die Durchführung des Workflows als Macroflow an.

Abbildung 6 - Anwendungsaufbau der Macroflow-Implementierung verdeutlicht den Aufbau dieser Implementierung. Die horizontalen Linien zeigen jeweils die Transaktionsgrenzen an.



6 - Anwendungsaufbau der Macroflow-Implementierung

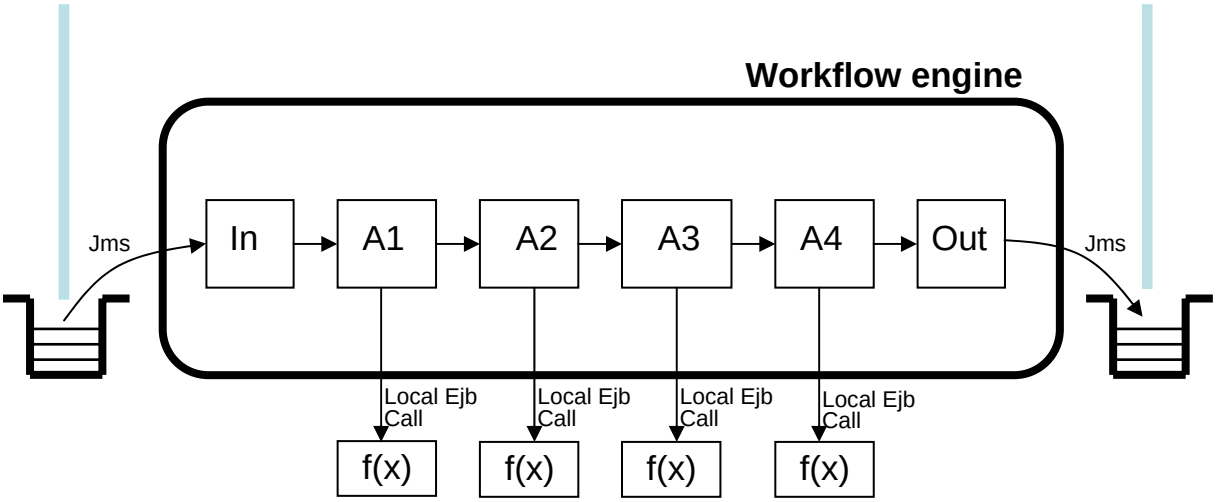
Ähnlich der PaF-Implementierung geschehen mit diesem Dienst die einzelnen Aktionen jeweils in einer Transaktion, so dass in Verbindung mit der stattfindenden Sicherung der ein Mal erreichten Zwischenstände im Fehlerfall ein zuvor erreichter Zustand wiederhergestellt werden kann.

WorkflowImplementationEAR (Microflow)

Die Workflow-Implementierung als Microflow basiert, wie die vorher beleuchtete Macroflow-Implementierung auch, auf einer Prozessdefinition in BPEL [6].

Der Wert „microflow“ des Attributs executionMode zeigt aber dieses Mal Durchführung des Workflows als Microflow an.

Abbildung 7 - Anwendungsaufbau der Microflow-Implementierung verdeutlicht den Aufbau dieser Implementierung, wobei die horizontalen Linien die Transaktionsgrenzen anzeigen.



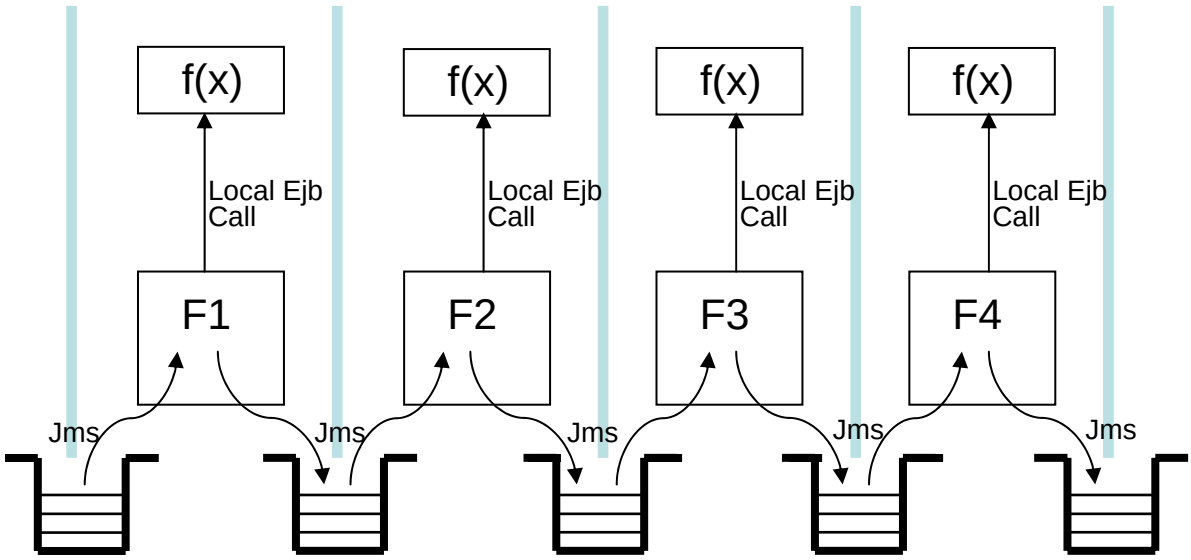
7 - Anwendungsaufbau der Microflow-Implementierung

Die Dienstqualität bei der Implementierung als Microflow ist, im Gegensatz zu den Macroflows, reduziert. So findet die gesamte Durchführung des Workflows hier innerhalb einer Transaktion und innerhalb eines Threads auf dem Anwendungsserver statt, so dass keine Zwischenschritte persistent festgehalten werden und die Ausführung dem Alles-oder-Nichts-Prinzip folgt.
Auf der anderen Seite geschehen aber in dem verwendeten Szenario nur synchrone Aufrufe, so dass dieser Dienst dazu geeignet ist, das bestehende Szenario zu lösen, weil dies eines der bevorzugten Einsatzgebiete für den Microflow ist.

MDB-Pipeline

Die Implementierung des PaF Musters basiert auf Message Driven Beans (MDB's), die an den Queues, die ebenfalls in dem Websphere Server angelegt worden sind, lauschen. Dadurch, dass eine MDB sowohl an einer Queue lauscht, als auch die von dort empfangene Nachricht an die Queue für die nächste MDB weitergibt, entsteht die in Abschnitt 2.2.2 angesprochene Pipeline.
Die Queues werden über eine JMS Verbindung angesprochen, so dass hier ein entfernter Aufruf geschieht.
Die zwischenzeitlich erreichten Zustände dieser Pipeline werden dabei in den Queues persistent festgeschrieben, so dass im Fehlerfall ein zuvor erreichter Zustand wiederhergestellt werden kann.
Jeder Filter arbeitet seinerseits vom Zeitpunkt des Empfangens einer Nachricht bis zum Zeitpunkt des Sendens einer Nachricht transaktional, so dass in den Filterkomponenten keine Nachricht verloren gehen kann und dementsprechend hier auch kein inkonsistenter Zustand entstehen kann.

Abbildung 8 - Anwendungsaufbau der PaF-Implementierung verdeutlicht den Aufbau dieser Implementierung, wobei die senkrechten Linien die Transaktionsgrenzen anzeigen.



8 - Anwendungsaufbau der PaF-Implementierung

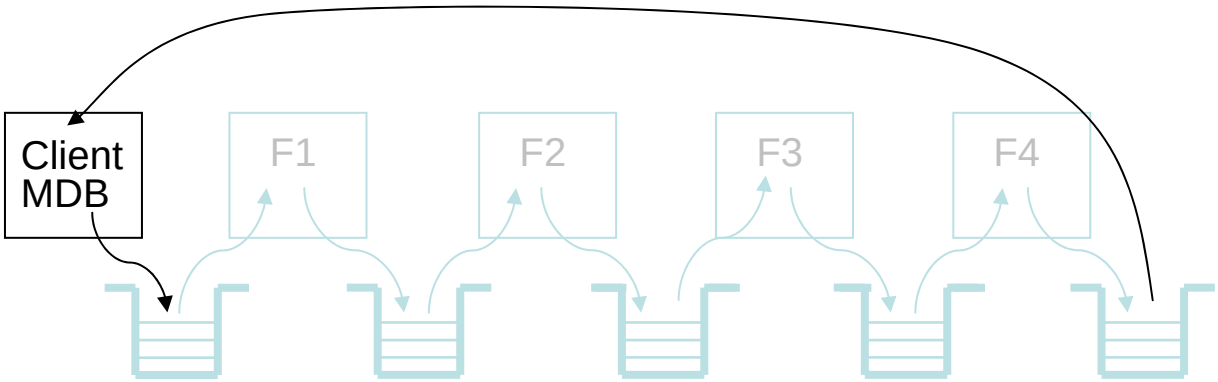
In diesem Zusammenhang sind mehrere Parameter, die die genaue Arbeitsweise der Pipeline bestimmen, zu klären.
Die im folgenden besprochenen Rahmenbedingungen ergeben sich aus den Administrationsmöglichkeiten, die der WebSphere Anwendungsserver für JMS zur Verfügung stellt.
Der erste ist *Maximum Messages*, der auf dem Server gesetzt wird und bestimmt, wie viele abgerufene Nachrichten gleichzeitig an einer Transaktion teilnehmen können (und dementsprechend an eine MDB übergeben werden). Um die Vergleichbarkeit mit den Workflow-Implementierungen zu gewährleisten, wird hier der Wert eins gewählt. Dies ergibt sich daraus, dass ein nicht abgeschlossener Microflow oder Macroflow nicht andere parallel in derselben Transaktion laufende Flows zu einem Abbruch bringen kann.
Der Parameter *Maximum Sessions* zeigt an, wie viele Instanzen von einer MDB zulässig sind. Dieser Wert ist dazu gedacht, den Server bezüglich verschiedener MDBs zu skalieren, also um beispielsweise einer MDB die mehrfache Instanziierung zu erlauben. Der Ressourcenverbrauch ist hiervon jedoch kaum beeinträchtigt (Messungen sind erfolgt), so dass ebenfalls der Wert eins gesetzt wird.
Die nächste Anforderung, die erfüllt sein muss, ist der *Transaktionsschutz* der MDBs, der garantiert, dass eine Nachricht auf der Queue erst dann gelöscht

wird, wenn die MDB, die sie abgerufen hat, sie erfolgreich bearbeitet hat. Dies ist notwendig, da es ansonsten möglich ist, dass Daten dauerhaft verloren gehen, was sowohl bei der Microflow, als auch bei der Macroflow-Implementierung nicht vorkommen kann.
Aus dieser Anforderung heraus ist es auch notwendig, dass die einzelnen Queues die Nachrichten persistent speichern.

Clients
Um die Integrationsarchitekturen selber instrumentiert aufzurufen, existieren zudem noch zwei Anwendungen, die als Client fungieren und die als SessionBean in die dazugehörenden Anwendungen integriert sind:

- Invocation_mdbService
- Invocation_workflowService

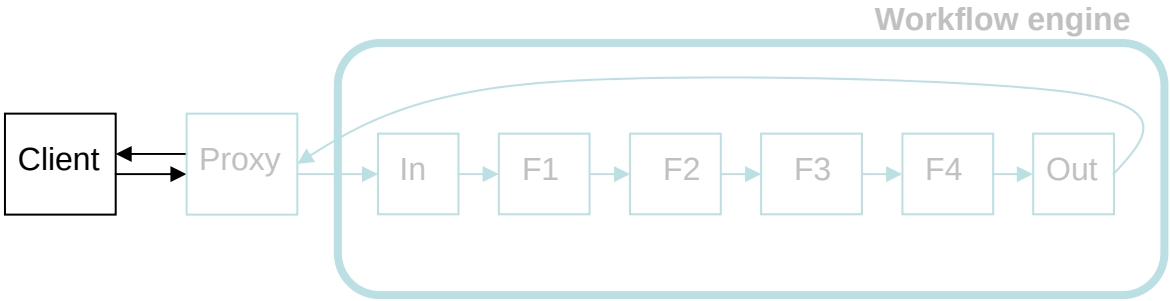
Die aufrufende Anwendung Invocation_mdbService stellt dabei eine Nachricht in die Startqueue. Diese Startqueue wird wiederum von einer MDB überwacht, die für die Pipeline als eigentlicher Client fungiert. Sie wird im folgenden Client MDB genannt und ist in der Lage Nachrichten an die Pipeline zu senden und Nachrichten von der Pipeline zu empfangen.
Abbildung 9 - Client zur PaF-Implementierung verdeutlicht den Aufbau, wobei die Anwendung Invocation_mdbService nicht dargestellt wird, da sie lediglich die initiale Nachricht in die letzte Queue stellt, so dass die Client MDB mit der Durchführung beginnen kann.
Wie auch in allen weiteren folgenden Grafiken sind hier die Aufrufe des zu integrierenden Dienstes ebenfalls nicht mehr explizit dargestellt.



9 - Client zur PaF-Implementierung

Der Client Invocation_workflowService ruft den Proxy Dienst des Workflows wiederholt auf und erhält die Ergebnisse der Ausführung.
Abbildung 10 - Client zur Workflow-Implementierung verdeutlicht die Lage des Clients.

Der Proxy stellt dabei eine Nachricht in die Eingangsqueue für den Prozess und empfängt eine Nachricht aus der Ausgangsqueue. Die Queues werden hier und in den folgenden Abbildungen nicht mehr explizit abgebildet.

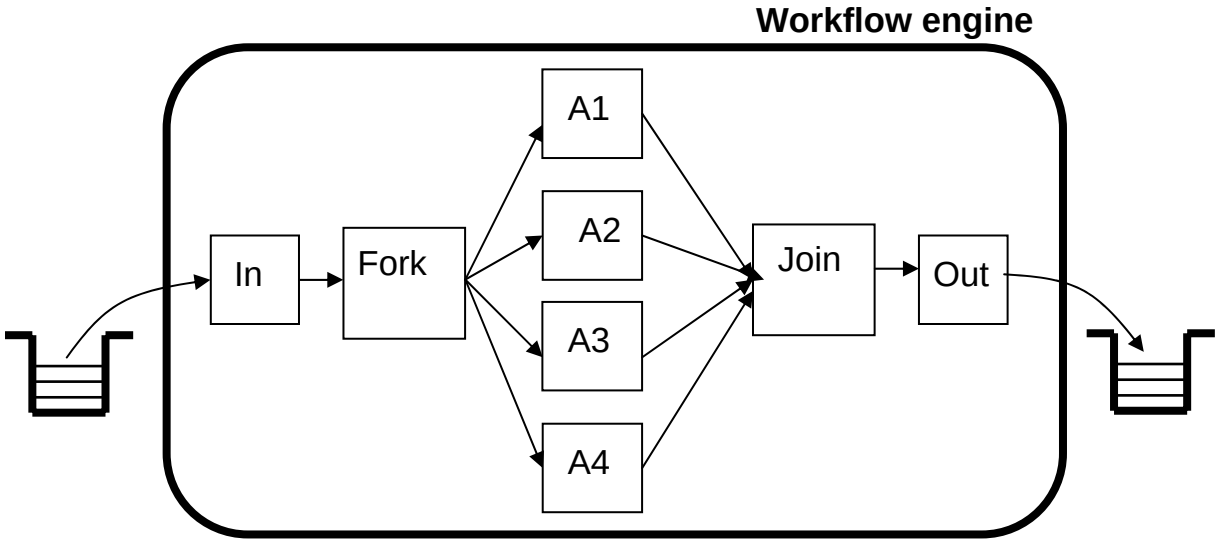


10 - Client zur Workflow-Implementierung

Zu den Implementierungen mit der Verwendung lediglich einer Aktivität beziehungsweise eines Filters existieren in diesem und dem nächsten Kapitel keine Abbildungen. Sie sind aber wie die Implementierungen mit vier Filtern aufgebaut, außer dass drei Aktivitäten bzw. Filter fehlen.

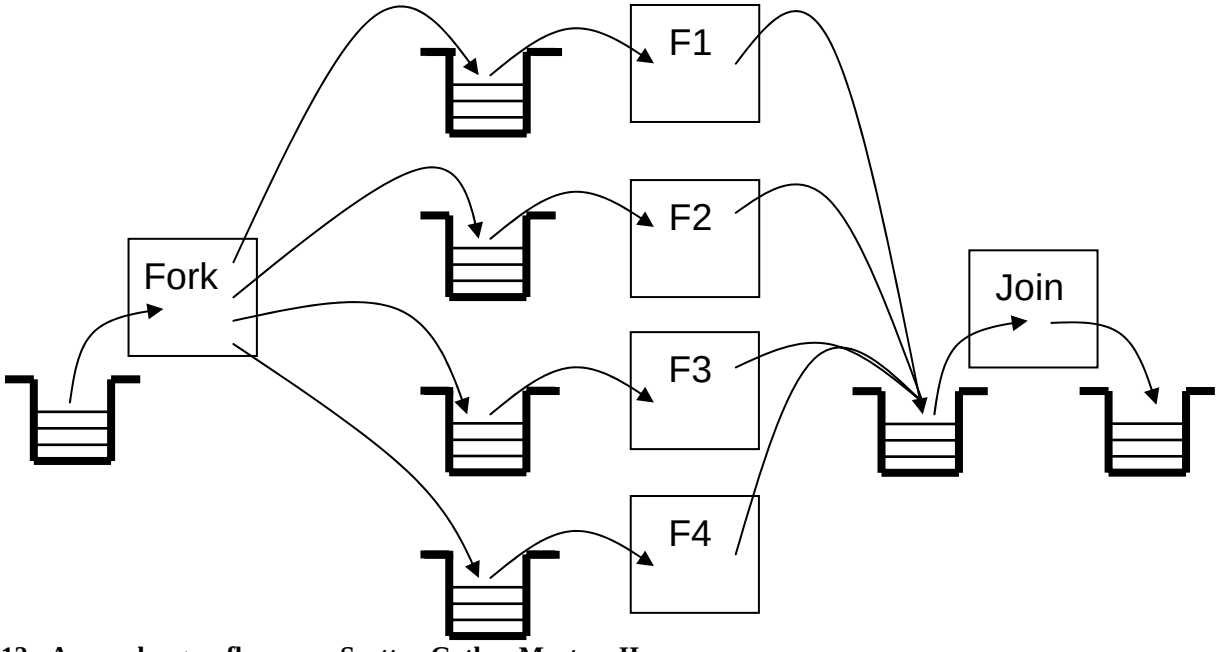
Scatter-Gather Muster

Da in einem der Szenarien zudem noch der Ressourcenverbrauch für die Durchführung eines Ablaufs überprüft wird, der dem Scatter-Gather-Muster folgt, ergeben sich zwei weitere Implementierungen.
In der Microflow-Implementierung wird dieses Muster über parallele Aktivitäten innerhalb eines Flows abgebildet. Die Ergebnisse der einzelnen Aktivitäten werden in einem Vereinigungsschritt zusammengefasst und an den Client zurückgesendet. Siehe hierzu auch Abbildung 11 - Anwendungsaufbau zum Scatter-Gather Musters 1.
Das Muster selber kann unter [9] nachgelesen werden.



11 - Anwendungsaufbau zum Scatter-Gather Musters 1

In der PaF-Implementierung wird dieses Muster abgebildet, indem in dem Filter, der die Verzweigungslogik enthält, gleichzeitig eine Nachricht an vier Filter geschickt wird. Die angesprochenen Filter wiederum senden jeweils eine Nachricht an einen Filter, der daraufhin alle Nachrichten zu einer Nachricht zusammenführt und sie an den Client zurücksendet. Siehe hierzu auch Abbildung 12 - Anwendungsaufbau zum Scatter-Gather Musters II.



12 - Anwendungsaufbau zum Scatter-Gather Musters II

5 Messwerkzeug und Messpunkte

Mit dem Messwerkzeug soll der Ressourcenverbrauch der untersuchten Implementierungen zwischen definierten Messpunkten festgestellt werden.
In diesem Kapitel wird daher zunächst in Abschnitt 5.1 unter anderem die Frage beantwortet, welche Messdaten für die Analyse benötigt werden.
In Abschnitt 5.2 werden dann die verschiedenen Ansätze zum Feststellen der Messdaten und die bereits verfügbaren Werkzeugimplementierungen vorgestellt und diskutiert. Dieser Abschnitt schließt mit der Beschreibung, wie das Messwerkzeug für diese Arbeit aufgebaut ist.
Danach werden in 5.3 die Messpunkte definiert, an denen die Messungen erhoben werden.

5.1 Messziele

Um den Ressourcenverbrauch von zwei Softwaresystemen miteinander zu vergleichen muss die Auslastung der Betriebsmittel des Rechners untersucht werden.
Zu diesen Betriebsmitteln eines Rechners zählt natürlich insbesondere der Prozessor und im nächsten Schritt der flüchtige und der persistente Speicher.
Der in dieser Arbeit untersuchte Ressourcenverbrauch gliedert sich daher in die folgenden Punkte

- **Time** – Zeitdauer der Ausführung (in Sekunden)
- **CPU-Time** – Zeitdauer, die der Prozessor verwendet und gesperrt wurde (in Sekunden)
- **Average CPU Usage** – relative Prozessorauslastung (in Prozent)
- **Working Set** – Hauptspeicherallokation (in Megabyte)
- **Read** – lesende Festplattenzugriffe (in Megabyte)
- **Write** – schreibende Festplattenzugriffe (in Megabyte)

Um alle entstehenden Ressourcenverbräuche durch die Workflow-Implementierung und die PaF-Implementierung zu protokollieren wird das Messwerkzeug Messdaten für die folgenden Prozesse erheben:

- **JVM-Prozess** – Java-Prozess, der die PaF-Implementierung bzw. die Workflow-Implementierung enthält
- **MQ-Prozesse** – native Prozesse, die die Funktionalität des Message Queueings zur Verfügung stellen

Nach der Klärung der Frage, welche Messdaten erhoben werden sollen, ist noch ein weiteres Ziel von Interesse.
Da die Messpunkte teilweise innerhalb der Anwendungen liegen werden, ist es erforderlich, dass dynamische Aufrufe durch das Messwerkzeug unterstützt werden.

5.2 Auswahl der Messwerkzeuges

In diesem Abschnitt werden die möglichen Messwerkzeuge aufgeführt und ihre Stärken und Schwächen diskutiert, so dass schließlich eine Auswahl des für diese Arbeit am besten geeigneten Werkzeuges stattfinden kann.

5.2.1 Überwachung mit Mitteln des Betriebssystems

In das Betriebssystem Microsoft Windows ist einerseits Funktionalität integriert die der Überwachung von elementaren Ereignissen dient. Andererseits gibt es auch die Möglichkeit über zahlreiche Schnittstellen Informationen zu den laufenden Prozessen auszulesen.
Im Folgenden erfolgt eine Diskussion der Möglichkeiten und der Praktikabilität dieses Ansatzes.

Windows Performance Monitor
Dieser Dienst dient der Überwachung und dem Monitoring des Betriebssystems und bietet die Funktionalität, detailliert Daten zur Auslastung der Ressourcen des Rechners inkl. Prozessor, Hauptspeicher, Festplattenspeicher, Cache, Threads und Prozesse zu erheben. Dazu werden sogenannte Counter definiert, die die benötigten Informationen protokollieren.
Um diesen Dienst zu benutzen, gibt es unterschiedliche Möglichkeiten. Die beiden wichtigsten Anwendungen sind

- perfmon.msc - Ein Werkzeug, welches zur Programmgruppe "Administration des Betriebssystems" gehört und in [10] detailliert eingeführt wird. Mit diesem Programm können die Counter für die Resourcen definiert und im nächsten Schritt grafisch aufbereitet in Diagrammen eingesehen werden. Ist es allerdings notwendig, dass dieser Dienst Verlaufsdaten speichert, so bedarf es einer weitergehenden Konfiguration inklusive der Definition und Anbindung einer ODBC-fähigen Schnittstelle. Zudem gibt es bei einer Verwendung dieses Ansatzes keine Möglichkeit, das Protokollieren von Verlaufsdaten durch einen externen Aufruf zu starten, so dass dieses Vorgehen nicht in Frage kommt.
- Windows Management Interface (WMI) - Diese Schnittstelle bietet die Möglichkeit skriptbasiert auf die Ressourcen des Betriebssystems zuzugreifen, die für die Administration benötigt werden. Eine genaue Beschreibung dieser Schnittstelle befindet sich bei [11]. Zu den ansprechbaren Ressourcen zählen dabei auch die Dienste, die durch den Performance Monitor zur Verfügung gestellt werden, so dass mit Hilfe dieser Schnittstelle die externen Aufrufe in die für die Messungen benötigten Dienste des Betriebssystems durchgeführt werden können. Eine Implementierung eines WMI Providers und eines WMI Agenten

erscheint allerdings zu aufwändig, da unter anderem das Problem der Prozessidentifizierung zur Laufzeit gelöst werden muss, so dass dieses Vorgehen nicht in Frage kommt.

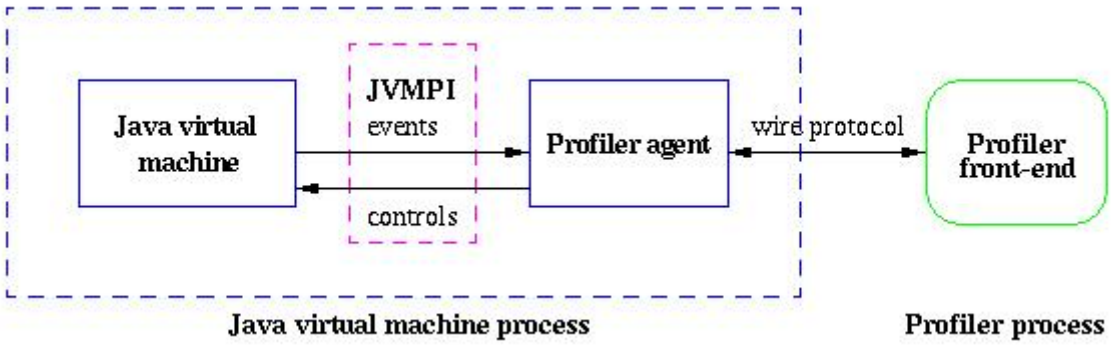
NSClient4j
Das Logging-Frameworks NSClient4j basiert auf dem Programm NSClient, welches unter Windows zur Resourcenüberwachung installiert werden kann [12]. Auch wenn hier, ähnlich dem Performancemonitor, Counter definiert werden können und über eine Java-Api auch externe Aufrufe möglich sind, so bleibt das Problem der Identifizierung der zu überwachenden Java-Prozesse zur Laufzeit noch ungelöst. Zudem verwendet die Anwendung NSClient Dynamic Link Libraries (DLLs) des Betriebssystems, die durch eine eigene native Implementierung auch direkt eingebunden und verwendet werden können, so dass ein direkter Aufruf der zugrunde liegenden Funktionen vorteilhaft erscheint.

Direkte Nutzung der Ressourcen des Betriebssystems
Zu dem Kern des Microsoft Betriebssystems gehören zahlreiche DLLs, die die Hard- und Software Ressourcen, die dem Rechner zugeordnet sind, verwalten. Sie überwachen die Prozessausführung, die Speicherverwaltung, Festplattenzugriffe und viele andere Aktionen, die mit den Messungen erfasst werden sollen. Da mit diesen Bibliotheken alle relevanten Ausführungen protokolliert werden können, ist es möglich, über ihre verfügbaren externen Schnittstellen die benötigten Informationen zu erhalten.
Die Benutzung dieser Funktionalität geschieht von Java aus über einen Aufruf von nativen Windows Funktionen und lässt sich vergleichsweise leicht implementieren.

5.2.2 Überwachung mit Mitteln der Java VM

Für die Java Virtual Mashine (JVM) existiert eine Schnittstellen, die implementiert werden kann, um Daten zum Ressourcenverbrauch des laufenden Java Prozesses zu erheben. Diese Schnittstelle heißt Java Virtual Mashine Profiling Interface (JVMPI) und bietet die Implementierung zahlreicher Ereignis-Handler an, die in der Schnittstelle definiert sind. Eine Spezifikation dieser Schnittstelle befindet sich bei [13].
Im Gegensatz zur Überwachung mit den Mitteln des Betriebssystems können durch Verwendung dieses Interfaces die Messergebnisse feingranular erhoben werden, da es möglich ist, die interne Verarbeitung des Java-Prozesses zu überwachen.

Der Aufbau einer JVMPI Implementierung ist prinzipell zweistufig ausgelegt. Siehe hierzu auch Abbildung 13 - Bestandteile einer JVMPI Implementierun (Quelle [13]). Die sogenannten Profiler agents sind die Komponenten, die das JVMPI implementieren und damit die Ereignisse der JVM behandeln, während die Profiler front-ends, die Daten eines Agents aufbereiten.



13 - Bestandteile einer JVMPI Implementierun (Quelle [13])

Ein Dienst, der diese Schnittstelle implementiert wird der JVM zum Startzeitpunkt über einen Parameteraufruf bekannt gegeben, so dass die JVM nach erfolgreichem Start die registrierten Ereignisse an den Agent weiterleitet.
Zur Benutzung dieses Dienstes gibt es mehrere Möglichkeiten, wobei die beiden wichtigsten kurz vorgestellt werden.

Java Profiling Werkzeuge

Es existieren zahlreiche kommerzielle und nicht kommerzielle Werkzeuge, die sich jedoch in ihrer Funktionialität sehr stark unterscheiden. Folgende Anwendungen wurden auf einen möglichen Einsatz hin untersucht:

- HProf - Dieser Profiler Agent von Sun ist im JDK seit der Version 1.2 standardmäßig enthalten. Er bietet keine Möglichkeit zur dynamischen Einbindung, erstellt aber umfangreiche Protokolle für die es frei verfügbare Profiler Frontends gibt, die die Daten interpretieren und grafisch aufbereiten. Auch wenn die Datenaufbereitung durch die Werkzeuge sehr gut gelungen ist, kommt dieser Profiler nicht in Frage, da bei einer Einbindung des Agenten der Laufzeitverlust für die zu überwachenden Prozesse zu hoch ist.
- Jmp - Dieser Open Source Profiler fungiert sowohl als Profiler Agent als auch als Profiler Frontend, indem eine GTK+ Oberfläche angebunden ist, an die die Daten weitergeleitet werden. Der Profiler arbeitet allerdings nicht mit der JVM des Websphere 5.1 zusammen, so dass er nicht in Frage kommt.

- Your Kit Java Profiler - Dieser kommerzielle Profiler ist JMP in seiner Funktionalität ähnlich und arbeitet ebenfalls nicht mit der JVM von Websphere zusammen

Die verfügbaren Werkzeuge haben sich in der konkreten Anwendung also als untauglich herausgestellt.
Zu der JVM Implementierung von IBM und der beobachteten Inkompatiblität ist anzumerken, dass durch die vorgenommenen Optimierungen die Schnittstelle JVMPI sich in ihrem Verhalten zur Referenz JVMPI von Sun unterscheidet und nicht mehr wie spezifiziert arbeitet.
Beispiel: Mit Hilfe des hinzugefügten Code-Inlinings werden Methoden direkt in die aufrufenden Methoden integriert, um den Laufzeit Stapel zu verkleinern. Dadurch werden aber im folgenden die Method Entries nicht mehr ausgelöst und an die JVMPI übergeben [14].
Die Unterstützung der Profiling Funktionialität wird aber in der Version 1.5 und dem damit zusammenhängenden Wechsel zum Interface Java Virtual Machine Tool Interface (JVMTI) behoben.

Eigene Entwicklung
Das Schreiben eines eigenen Profiler Agents ist prinzipiell möglich, da ähnlich der Einbindung nativer Windows Betriebssystem Funktionen für die JVM eine native Anwendung geschrieben werden kann, die das JVMPI implementiert. Mit Hilfe dieser Schnittstelle können die elementaren Ereignisse des JVM, wie bereits beschrieben, abgefangen und behandelt werden.
Der Nachteil einer eigenen Entwicklung an dieser Stelle ist jedoch, dass für komplexere Messoperationen eine größere Anwendung geschrieben werden muss, so dass dieses Vorgehen nicht bevorzugt wird.

5.2.3 Implementierung des Messwerkzeuges

Nachdem in den letzten beiden Abschnitten die Stärken und Schwächen der beiden Ansätze festgestellt worden sind und auch zahlreiche praktische Tests dazu stattgefunden haben, ergeben sich nun einige Implikationen für die Verwendung beziehungsweise den Aufbau des Messwerkzeuges.

Die Feststellung des Ressourcenverbrauchs mit den Mitteln des Betriebssystems ist ein vielversprechender Ansatz, aber nicht aussreichend.

- Die Schnittstellen des Betriebssystems bieten die Möglichkeit zahlreiche Daten zu den laufenden Prozessen anzufordern. Dies betrifft die CPU-Time, und die Festplattenzugriffe
- Die Hauptspeicherallokation kann nicht zufrieden stellend ermittelt werden, da hervorgerufen durch den Einsatz der Garbage Collection

innerhalb der JVM die Auslastung des Speichers nicht genau ermittelt werden kann

Für die Feststellung des Ressourcenverbrauchs durch Werkzeuge die das JVMPI benutzen haben sich folgende Punkte herausgestellt:

- Die verfügbaren JVMPI Profiling Werkzeuge sind ungeeignet, da sie nicht mit der JDK 1.4.2 von IBM zusammenarbeiten
- Die Verwendung eines JVMPI Profiling Werkzeuges ist aber trotzdem notwendig, um die Speicherallokation überwachen zu können.

Ausgehend von diesen Tatsachen bietet sich ein zweigeteilter Aufbau des Messwerkzeuges an.

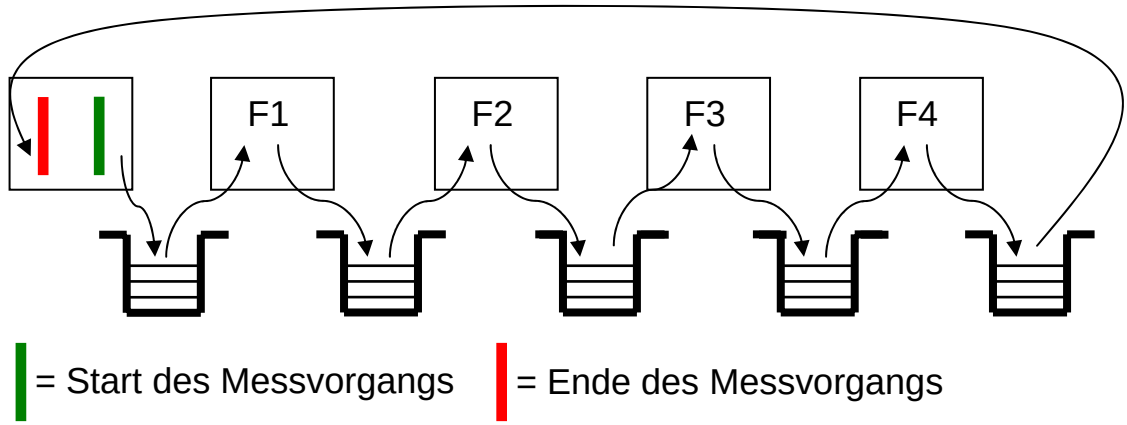
Der erste Teil betrifft die Ansprache von Funktionen, die in das Betriebssystem integriert sind.
Hierzu werden Funktionen innerhalb von DLLs verwendet, die die Prozesse und ihre Resourcenzuweisung verwalten und überwachen. Neben den gebotenen Möglichkeiten zur Protokollierung ist es zudem möglich, diese Funktionen durch einen externen Aufruf einzubinden, so dass alle Voraussetzungen an die Ziele des Messwerkzeugs, erfüllt sind.
Die Implementierung des Messwerkzeugs, das diese Funktionen verwendet, erfolgt in C++ und der Aufruf aus Java heraus geschieht über eine ebenfalls implementierte native Schnittstelle.
Der zweite Teil betrifft die Implementierung des JVMPI. Hierbei werden die Ereignisse zur Objektallokation der JVM abgefangen und nachverfolgt, so dass die Summe aller allokierten Objekte nachvollzogen werden kann. Dieser Profiler wird zusammen mit der JVM gestartet.
Eine Einschränkung des Messwerkzeuges muss an dieser Stelle noch genannt werden. Der Hauptspeicherbedarf wird durch das Messwerkzeug nicht errechnet. Das Messwerkzeug addiert aus Gründen der Performanz lediglich die Größe der allokierten Objekte.

5.3 Messpunkte

Die Erhebung der Messdaten geschieht auf zwei verschiedene Arten.
Zum einen wird der Ressourcenverbrauch erhoben, der zwischen dem Aufruf durch einen Client und der erfolgreichen Bearbeitung des Aufrufs also bis zur erfolgreichen Rückmeldung an den Client geschieht, so dass der Ressourcen-verbrauch für einen kompletten Durchlauf inklusive des Aufrufs erfasst wird.
Zum anderen wird der Ressourcenverbrauch, der für die interne Verarbeitung anfällt, erhoben. Dies betrifft den Zeitraum vom Eingang der Nachricht in PaF-

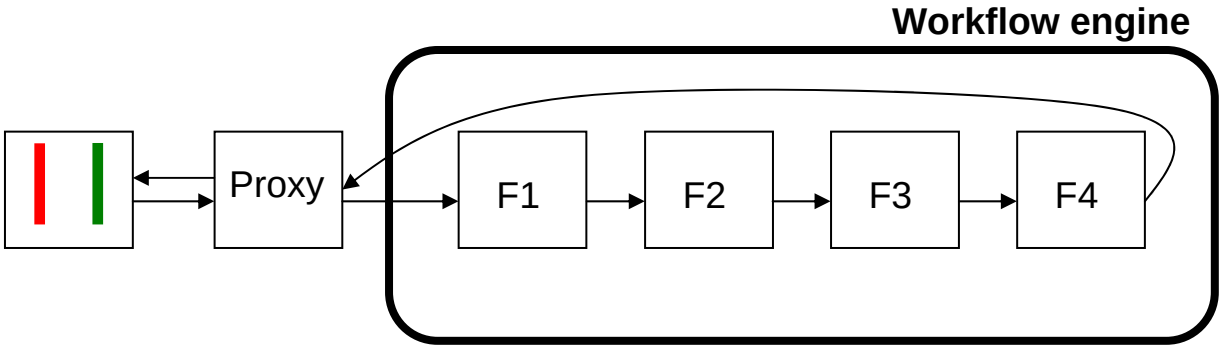
Implementierung beziehungsweise die Workflow-Implementierung bis zum Austritt der Nachricht aus derselben.
Die in diesem Abschnitt dargestellten Szenariolösungen basieren jeweils auf vier Dienstaufrufen. Für den Fall, das nur ein Dienstaufruf geschieht, fallen lediglich drei Aktionen beziehungsweise drei Filterschritte weg.
Die Tatsache, dass das Messwerkzeug ebenfalls Ressourcen verbraucht und somit die Messdaten verfälscht, stellt bei der Wahl aller genannten Messpunkte kein Problem dar, da dieser Aufwand bei beiden Implementierungen gleichermaßen anfällt.

Messpunkte – Client
Hier interessiert der Ressourcenverbrauch der vom Absenden der Anfrage durch einen Client, bis zum Eintreffen der Antwort, anfällt.
Abbildung 14 - Messpunkte für die Antwortzeit der PaF-Implementierung verdeutlicht diesen Ansatz für die PaF-Implementierung. Die Pipeline in der Abbildung besteht hier aus vier Filtern, während der fünfte Filter die Rolle der Client MDB einnimmt, und die Messdaten anfordert.



14 - Messpunkte für die Antwortzeit der PaF-Implementierung

Für die Umsetzung derselben Messpunkte im Zusammenhang mit der Workflow-Implementierung siehe Abbildung 15 - Messpunkte für die Antwortzeit der Workflow-Implementierung.

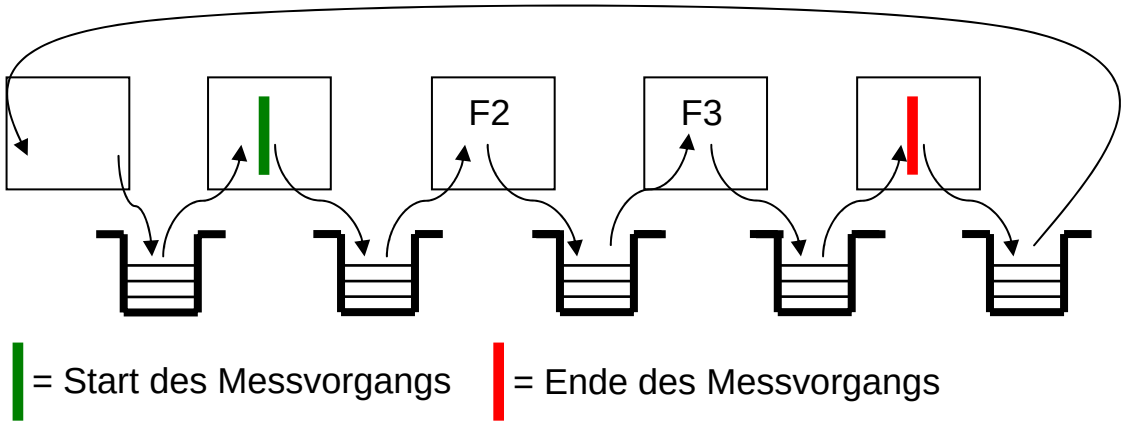


15 - Messpunkte für die Antwortzeit der Workflow-Implementierung

In diesem Fall fordert der Client des Proxy Dienstes für die Workflow-Implementierung alle Messdaten an.
Da der korrekte Ansatzpunkt für den Client und damit für das Erheben der Messergebnisse der Proxy-Dienst wäre, wurde im Rahmen dieser Arbeit auch ein eigener Client, der die Queues direkt bedient, geschrieben. Es hat sich jedoch im Folgenden herausgestellt, dass der generierte Proxy weniger Ressourcen verbraucht.

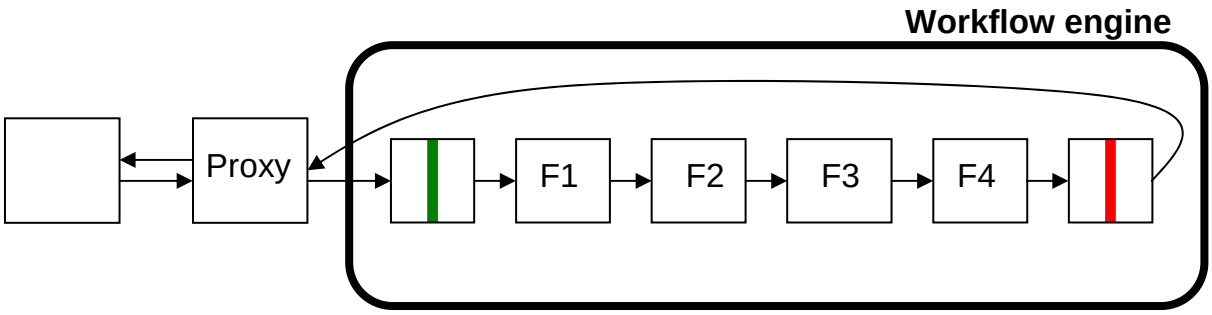
Messpunkte – interne Verarbeitung

Hier interessiert der Ressourcenverbrauch, der für die interne Verarbeitung der Nachricht durch den Workflow beziehungsweise durch die Pipeline anfällt.
Bei der PaF-Implementierung geschieht diese in dem Zeitraum nach dem Eintritt der Eingangsachricht in den ersten Filter bis zum Ausstritt der Ausgangs-nachricht aus dem letzten Filter, so dass an dieser Stelle auch die Messdaten erhoben werden. Siehe hierzu Abbildung 16 - Messpunkte für die Verarbeitungszeit der PaF-Implementierung.



16 - Messpunkte für die Verarbeitungszeit der PaF-Implementierung

Da in diesem Fall sowohl der erste, als auch der letzte Filter das Messwerkzeug ansprechen müssen, werden die Nachrichten von der Client MDB speziell markiert, da sie die Anzahl der versendeten Nachrichten überwacht.
So kann den beiden Filtern mit dem Messaufbau das Erreichen eines Zwischenschritts oder das Erreichen des Endes des Messvorgangs mitgeteilt werden.
Bei der Workflow-Implementierung beginnt der Zeitraum der internen Verarbeitung synonym nach dem Eintritt in den Workflow bis zum Austritt aus demselben. Hier ist wie bei der PaF-Implementierung auch das Empfangen der Nachricht aus der Queue bereits geschehen, beziehungsweise das Senden noch nicht durchgeführt. Siehe hierzu auch Abbildung 17 - Messpunkte für die Verarbeitungszeit der PaF-Implementierung.



17 - Messpunkte für die Verarbeitungszeit der PaF-Implementierung

Die Messschritte sind hierbei übrigens durch Java-Snippets realisiert, die in den Workflow eingefügt sind. In Ihnen geschieht, wie bei der PaF-Implementierung auch, der Aufruf des Messwerkzeugs.

Messvorgang

Ein *Messvorgang* besteht jeweils aus jeweils 100 Aufrufen des Integrationsdienstes. Dabei wird sowohl für die Workflow-Implementierungen, als auch für die PaF-Implementierung erst dann eine neue Anfrage gestellt, sobald die vorherige Anfrage beantwortet worden ist. Dies hat sich im Bezug auf die PaF-Implementierung, obwohl hier auch die Möglichkeit gegeben wäre die erste Queue zu Beginn mit 100 Nachrichten aufzufüllen und dann auf Bearbeitung des Bursts zu warten, als nicht nachteilhaft herausgestellt (Messungen sind erfolgt).
Zu jedem dieser einzelnen Aufrufe der Workflow-Implementierung und der PaF-Implementierung werden die *Messdaten*, an den zuvor beschriebenen Messpunkten, erhoben.

Nach jeweils zehn Aufrufen werden die Messedaten addiert und in eine Textdatei geschrieben, so dass nach einem abgeschlossenen Messvorgang insgesamt zehn festgehaltene *Messergebnisse* existieren.
Für jedes Szenario und jede Architektur werden insgesamt fünf dieser Messvorgänge durchgeführt, so dass die festgestellten *Messergebnisse* schließlich den Ressourcenverbrauch von 500 Aufrufen widerspiegeln.

6 Praktische Analyse der Effizienz

In diesem Kapitel wird, unter Verwendung der in 3.2 aufgestellten Szenarien, die Effizienz der beiden Lösungskonzepte im praktischen Einsatz festgestellt. Dazu werden die in 4.2 definierten Implementierungen, unter Verwendung der in 5.3 vorgestellten Messpunkte, auf ihren Ressourcenverbrauch hin überprüft.
Die Messergebnisse, die für den Vergleich Implementierungen im Bezug auf die Anzahl der Filter festgestellt worden sind, sind in Abschnitt 6.1 enthalten. Die Messergebnisse, die für den Vergleich der Implementierungen im Bezug auf die Nachrichtengröße festgestellt worden sind, sind in Abschnitt 6.2 enthalten.
Alle weiteren Vergleiche der Implementierungen sind Abschnitt 6.3 aufgeführt.
Die Darstellung der Messdaten und die Verdichtung der Ergebnisse geschehen dabei in den folgenden drei Dimensionen

- benötigte CPU Zeit
- allokierter Hauptspeicher
- beschriebener Festplattenspeicher

Alle übrigen Dimensionen werden nicht dargestellt, können jedoch im Anhang nachgelesen werden. Dies betrifft die folgenden Messwerte

- benötigte Zeit – nicht hinreichend stichhaltig, da eine sehr starke Abhängigkeit zur CPU-Auslastung besteht
- CPU Auslastung – stark schwankend, da das Betriebssystem die CPU allen laufenden Prozessen situationsbedingt zuweist
- Lesende Festplattenzugriffe – durch Caching der Objekte sind hier große Verzerrungen möglich

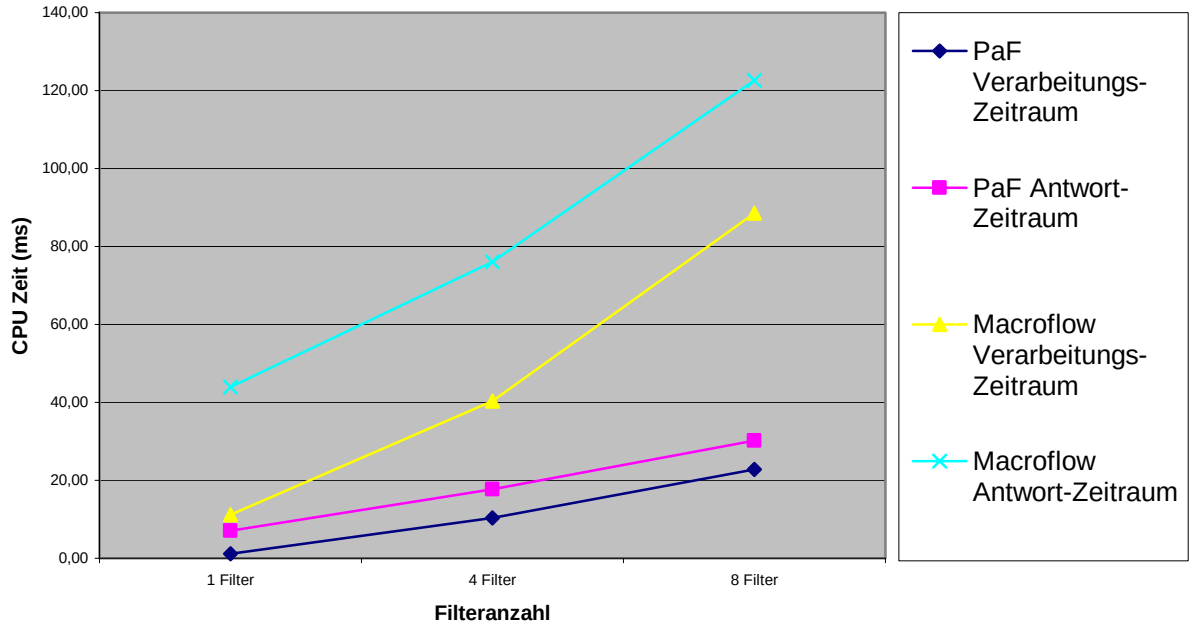
6.1 Messergebnisse in Abhängigkeit zur Filteranzahl

Für den Vergleich der drei Implementierungen werden die zuvor aufgestellten Szenarien 2, 5, 6, 8, 11 und 12 verwendet. Siehe hierzu auch Tabelle 2 - Auflistung der Szenarien für die Analyse I.

Nr.	**Dienstqualität**	**Anzahl der Filter**	**Nachrichtengröße (in Kb)**
2	persistent / transaktional	1	50
5	persistent / transaktional	4	50
6	persistent / transaktional	8	50
8	transient	1	50
11	transient	4	50
12	transient	8	50

2 - Auflistung der Szenarien für die Analyse I

Die ersten sechs Messreihen werden für die Szenarien 2, 5 und 6 erhoben, die jeweils durch eine Macroflow Implementierung und eine PaF-Implementierung gelöst werden. Die nächsten sechs Messreihen zeigen die Ressourcenverbräuche dieser Implementierungen, die während der internen Verarbeitung anfallen.
Darauf folgen sechs Messreihen für die Szenarien 8, 11 und 12, die jeweils durch eine Microflow-Implementierung mit einem JMS-Client und eine Microflow-Implementierung mit einem EJB-Client gelöst werden.
Schießlich werden noch drei Messreihen für die interne Verarbeitungsdauer und den dabei entstehenden Ressourcenverbrauch eines Microflows erhoben, so dass sich insgesamt 21 Messreihen ergeben.
In den folgenden Abbildungen spiegeln dann die zwei Messpunkte zum Antwortzeitraum und zum Verarbeitungszeitraum den Ressourcenverbrauch einer Implementierung für ein konkretes Szenario wieder.
Abbildung 18 - CPU Zeit I der Analyse I zeigt die Beanspruchung der CPU durch den Macroflow und die PaF-Implementierung. Dabei fällt die starke Auslastung der CPU durch den Macroflow auf, die auf den großen Aufwand der Operationen zur Sicherstellung der Dienstqualität hindeutet. So werden neben allen Variablen des Macroflows auch die Navigationsinformationen und eventuell weitere Metainformationen zur Dienstausführung gespeichert, die bei der zustandslosen Message Driven Bean nicht festgehalten und überprüft werden. Siehe hierzu auch die Diskussion der Effizienz der Implementierungen Abschnitt 2.4.2.
Eine Einschränkung muss an dieser Stelle allerdings noch gemacht werden. Der verhältnismäßig große Anstieg der bei den Macroflows in allen Abbildungen zu sehen ist, bedingt sich zumindest teilweise auch aus der Tatsache, dass hervorgerufen durch den hohen Ressourcenverbrauch entsprechend häufig die Garbage Collection aufgerufen werden muss, um den Speicher wieder aufzuräumen. Auf diese Weise erhöht sich der Aufwand abermals.



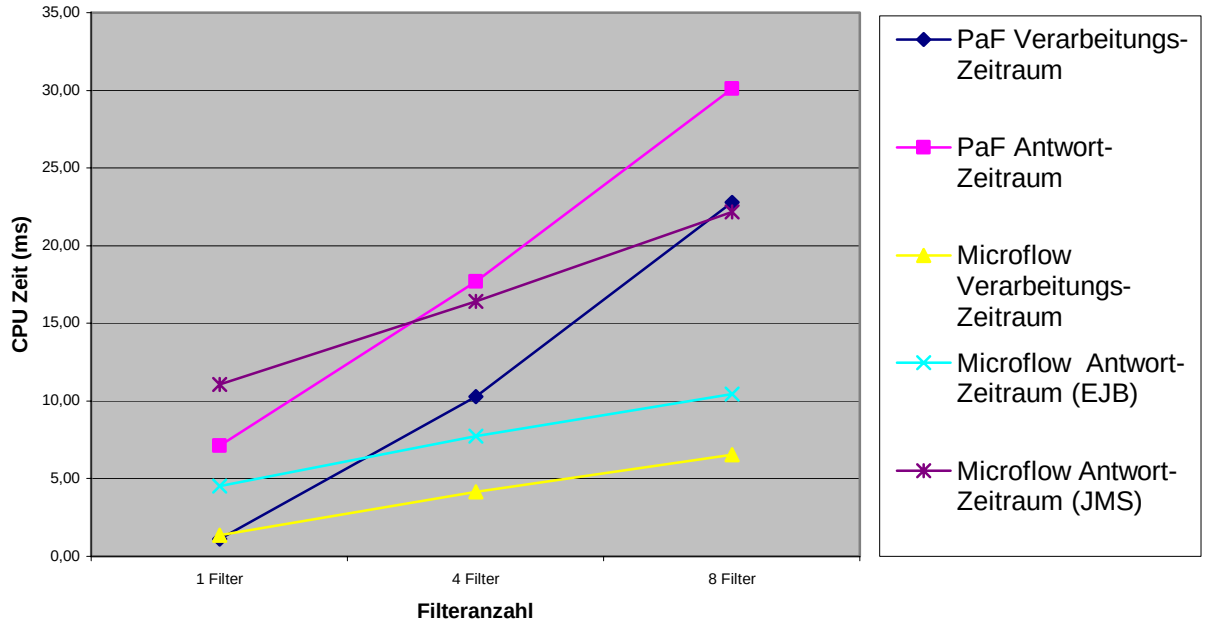
18 - CPU Zeit I der Analyse I

Tabelle 3 - CPU Zeit I der Analyse I Enthält die Messergebnisse zu dem Diagramm.

	1 Filter	**4 Filter**	**8 Filter**
PaF Verarbeitungs-Zeitraum	1,11	10,29	22,77
PaF Antwort-Zeitraum	7,13	17,67	30,12
Macroflow Verarbeitungs-Zeitraum	11,13	40,35	88,52
Macroflow Antwort-Zeitraum	43,95	76,08	122,58

3 - CPU Zeit I der Analyse I

In der Abbildung 19 - CPU Zeit II der Analyse I wird der Vergleich der CPU Beanspruchung der PaF-Implementierung mit der Microflow-Implementierung dargestellt. Hier fällt besonders die Effizienz der Microflows auf, deren Bearbeitungsdauer in Zusammenhang mit einem lokalen EJB-Aufruf, bereits bei einem Filter, um ein Drittel niedriger liegt als die Bearbeitungsdauer der vergleichbaren PaF-Implementierung.
Hinsichtlich der Steigerungsraten verhalten sich alle Betrachtungen des Microflows gleich, in dem ein leichter Anstieg der Verarbeitungszeit hinsichtlich gestiegener Filteranzahl zu beobachten ist. Die Steigerung bei der PaF-Implementierung fällt deutlich größer aus.



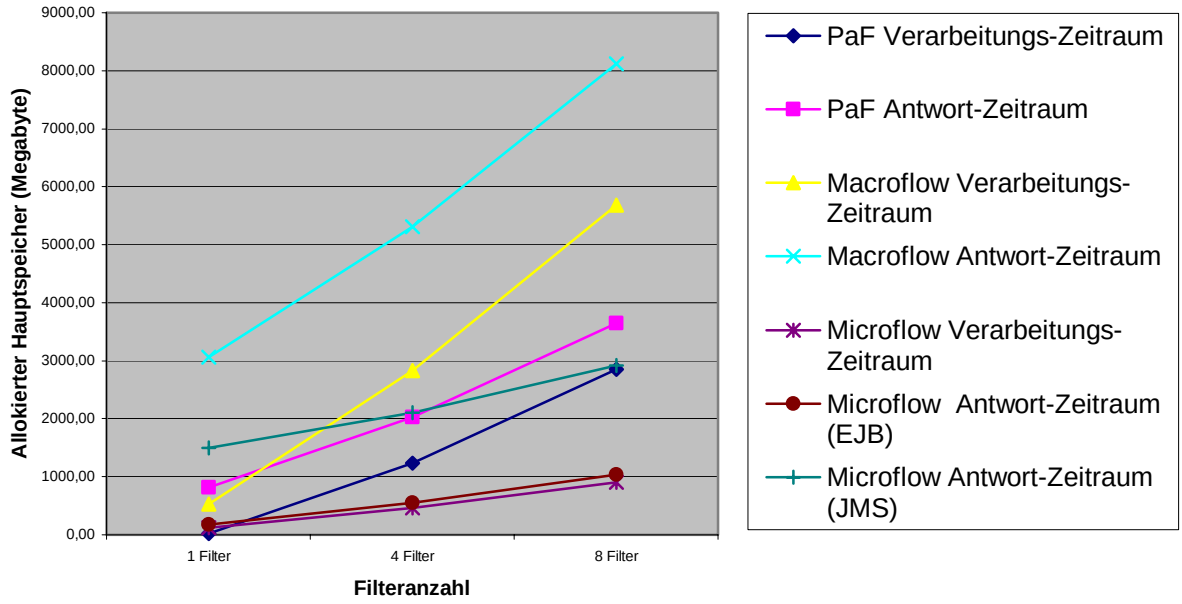
19 - CPU Zeit II der Analyse I

In Tabelle 4 - CPU Zeit II der Analyse I sind die Messergebnisse zu diesem Diagramm detailliert festgehalten.

	1 Filter	**4 Filter**	**8 Filter**
PaF Verarbeitungs-Zeitraum	1,11	10,29	22,77
PaF Antwort-Zeitraum	7,13	17,67	30,12
Microflow Verarbeitungs-Zeitraum	1,34	4,17	6,55
Microflow Antwort-Zeitraum (EJB)	4,51	7,73	10,43
Microflow Antwort-Zeitraum (JMS)	11,06	16,43	22,15

4 - CPU Zeit II der Analyse I

Bei den Messdaten ist zudem die hohe Sockelbearbeitungsdauer auffällig, die den Aufruf des Microflows mit einem JMS Proxy Dienst auszeichnet. Dieser Proxy Dienst ist generiert worden und wird auch in der Macroflow-Implementierung eingesetzt. Vergleichende Messungen mit einem selbst geschriebenen JMS Proxy Dienst haben jedoch gezeigt, dass es hier durch eine Eigenentwicklung wenig Optimierungspotential gibt, so dass für die Messungen dieser generierte Proxy Dienst verwendet wurde.
Abbildung 20 - Speicherallokation der Analyse I zeigt die Speicherallokation der überprüften Implementierungen. Auch hier sind wieder sehr deutlich die unterschiedlichen Steigerungsraten bezüglich der drei Implementierungsarten zu erkennen.



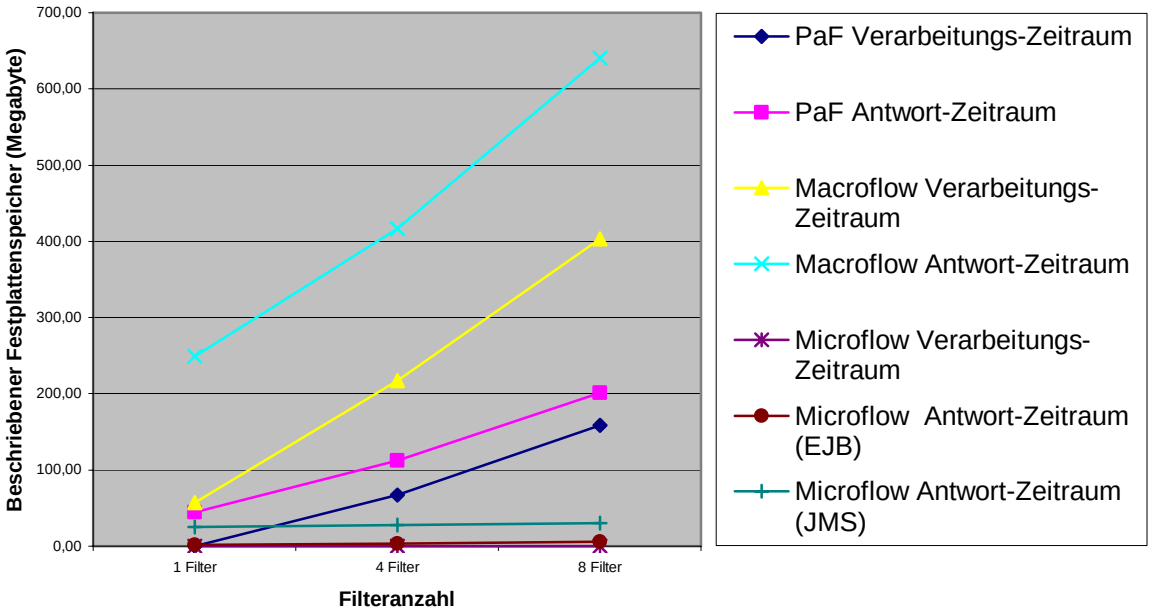
20 - Speicherallokation der Analyse I

Tabelle 5 - Speicherallokation der Analyse I enthält die Messergebnisse.

	1 Filter	**4 Filter**	**8 Filter**
PaF Verarbeitungs-Zeitraum	19,48	1234,01	2853,08
PaF Antwort-Zeitraum	816,29	2030,39	3649,74
Macroflow Verarbeitungs-Zeitraum	533,59	2830,38	5678,72
Macroflow Antwort-Zeitraum	3057,93	5315,01	8118,90
Microflow Verarbeitungs-Zeitraum	126,65	462,82	908,68
Microflow Antwort-Zeitraum (EJB)	176,76	545,53	1035,43
Microflow Antwort-Zeitraum (JMS)	1493,82	2105,88	2921,91

5 - Speicherallokation der Analyse I

Abbildung 21 - Beschriebener Festplattenspeicher der Analyse I verdeutlicht die Menge der Daten die von den jeweiligen Implementierungen persistent festgehalten werden. Auch diese Abbildung bestätigt im Bezug auf die Steigerungsraten im Ressourcenverbrauch die bisherigen Erkenntnisse.



21 - Beschriebener Festplattenspeicher der Analyse I

Die Messergebnisse sind in Tabelle 6 - Beschriebener Festplattenspeicher der Analyse I abgebildet.

	1 Filter	**4 Filter**	**8 Filter**
PaF Verarbeitungs-Zeitraum	0,02	67,16	158,11
PaF Antwort-Zeitraum	44,77	112,09	201,51
Macroflow Verarbeitungs-Zeitraum	57,08	216,71	403,28
Macroflow Antwort-Zeitraum	249,28	416,97	640,39
Microflow Verarbeitungs-Zeitraum	0,00	0,00	0,00
Microflow Antwort-Zeitraum (EJB)	1,67	3,17	5,84
Microflow Antwort-Zeitraum (JMS)	24,80	27,70	30,56

6 - Beschriebener Festplattenspeicher der Analyse I

6.2 Messergebnisse in Abhängigkeit zur Nachrichtengröße

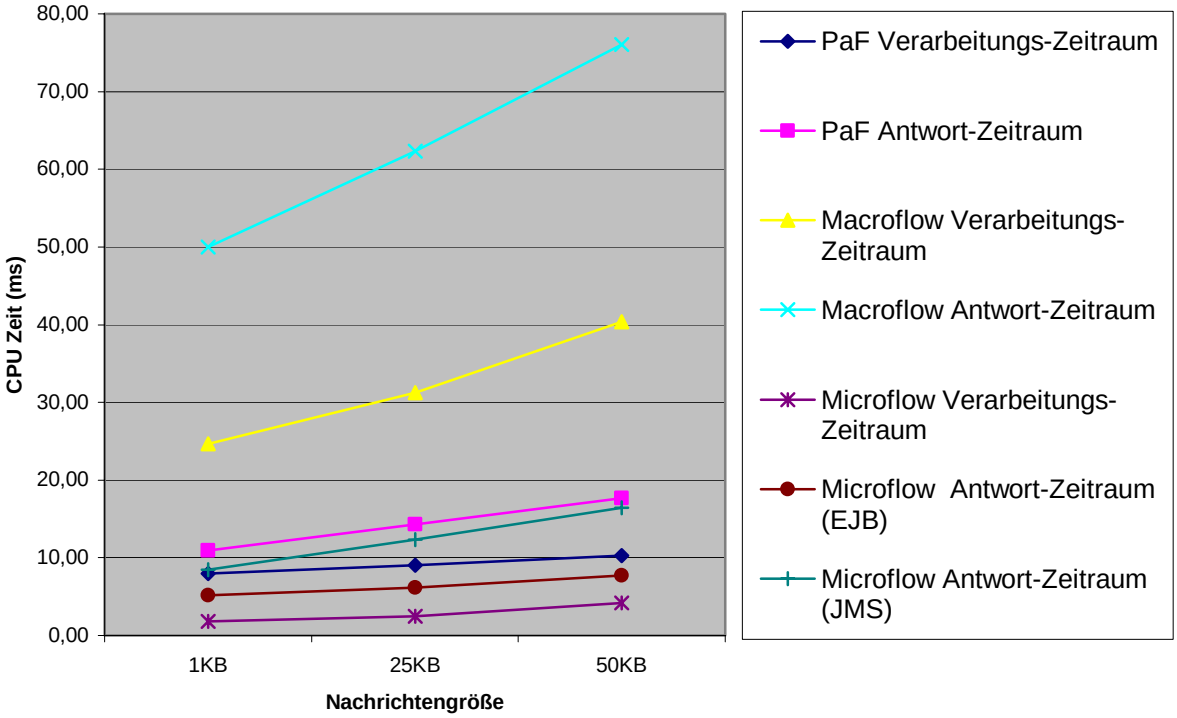
Für den Vergleich der beiden Implementierungen werden die zuvor aufgestellten Szenarien 3, 4, 5, 9, 10 und 11 verwendet. Siehe hierzu auch Tabelle 7 - Auflistung der Szenarien für die Analyse II.

Nr.	Dienstqualität	Anzahl der Filter	Nachrichtengröße (in Kb)
3	persistent / transaktional	4	1
4	persistent / transaktional	4	25
5	persistent / transaktional	4	50
9	transient	4	1
10	transient	4	25
11	transient	4	50

7 - Auflistung der Szenarien für die Analyse II

Die ersten sechs Messreihen werden für die Szenarien 3, 4 und 5 erhoben, die jeweils durch eine Macroflow Implementierung und eine PaF-Implementierung gelöst werden. Die nächsten sechs Messreihen zeigen die Ressourcenverbräuche dieser Implementierungen, die während der internen Verarbeitung anfallen.
Darauf folgen sechs Messreihen für die Szenarien 9, 10 und 11, die jeweils durch eine Microflow-Implementierung mit einem JMS-Client und eine Microflow-Implementierung mit einem EJB-Client gelöst werden.
Schießlich werden noch drei Messreihen für die interne Verarbeitungsdauer und den dabei entstehenden Ressourcenverbrauch eines Microflows erhoben, so dass sich insgesamt 21 Messreihen ergeben.

Die Ergebnisse dieser Messreihen werden im folgenden aufgeführt und interpretiert.
Abbildung 22 - CPU Zeit der Analyse II verdeutlicht die Beanspruchung der CPU durch die verschiedenen Implementierungen. Dabei zeigt sich, dass die Macroflow-Implementierung hervorgerufen durch die Operationen zur Sicherstellung der Dienstqualität die CPU am stärksten beansprucht, während die Microflow-Implementierung diese am wenigsten beansprucht.
Die Steigerungsrate der Microflow–Implementierung und der PaF-Impelmentierung in der CPU Verarbeitungszeit sind sich ähnlich bezüglich größer werdender Nachrichten.



22 - CPU Zeit der Analyse II

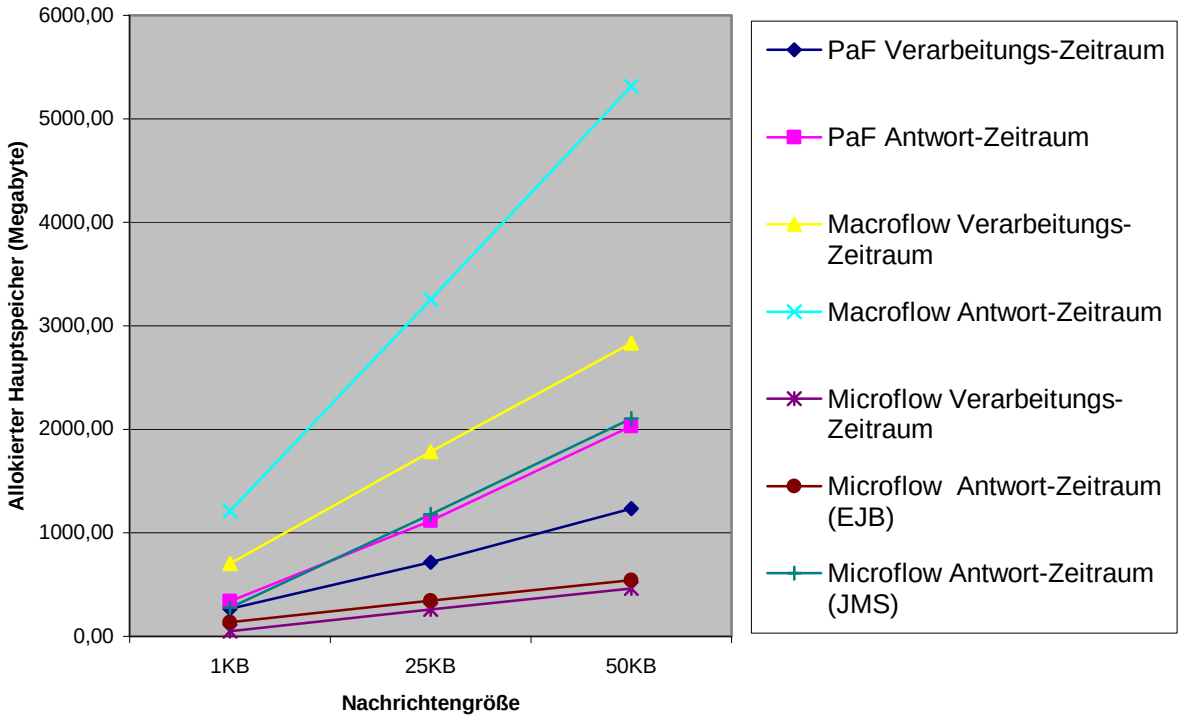
Die Messdaten zu der Grafik sind in Tabelle 8 - CPU Zeit der Analyse II aufgeführt.

	1KB	**25KB**	**50KB**
PaF Verarbeitungs-Zeitraum	7,98	9,05	10,29
PaF Antwort-Zeitraum	10,93	14,32	17,67
Macroflow Verarbeitungs-Zeitraum	24,68	31,28	40,35
Macroflow Antwort-Zeitraum	49,95	62,34	76,08
Microflow Verarbeitungs-Zeitraum	1,84	2,44	4,17
Microflow Antwort-Zeitraum (EJB)	5,15	6,19	7,73
Microflow Antwort-Zeitraum (JMS)	8,45	12,34	16,43

8 - CPU Zeit der Analyse II

Bei der Betrachtung der Steigerungsraten zeigt sich, dass die Microflow-Implementierung und die PaF-Implementierung sich hier und in den weiteren Ressourcenverbräuchen ähnlich sind, wobei der Microflow aber zugleich etwas effizienter arbeitet und auch die Steigerungen zumeist etwas niedriger ausfallen. Diese belegt auch die Betrachtung des in Abbildung 23 - Speicherallokation der Analyse II gezeigten Verlaufs zur Speicherallokation. Der unterschiedlich starke Anstieg der beiden Messungen zur Macroflow-Implementierung ergibt sich aus

der Tatsache, dass zu Beginn der internen Verarbeitung bereits zahlreiche Objekte zur Prozessausführung instanziiert worden sind und während der internen Verarbeitung zur Verfügung stehen.



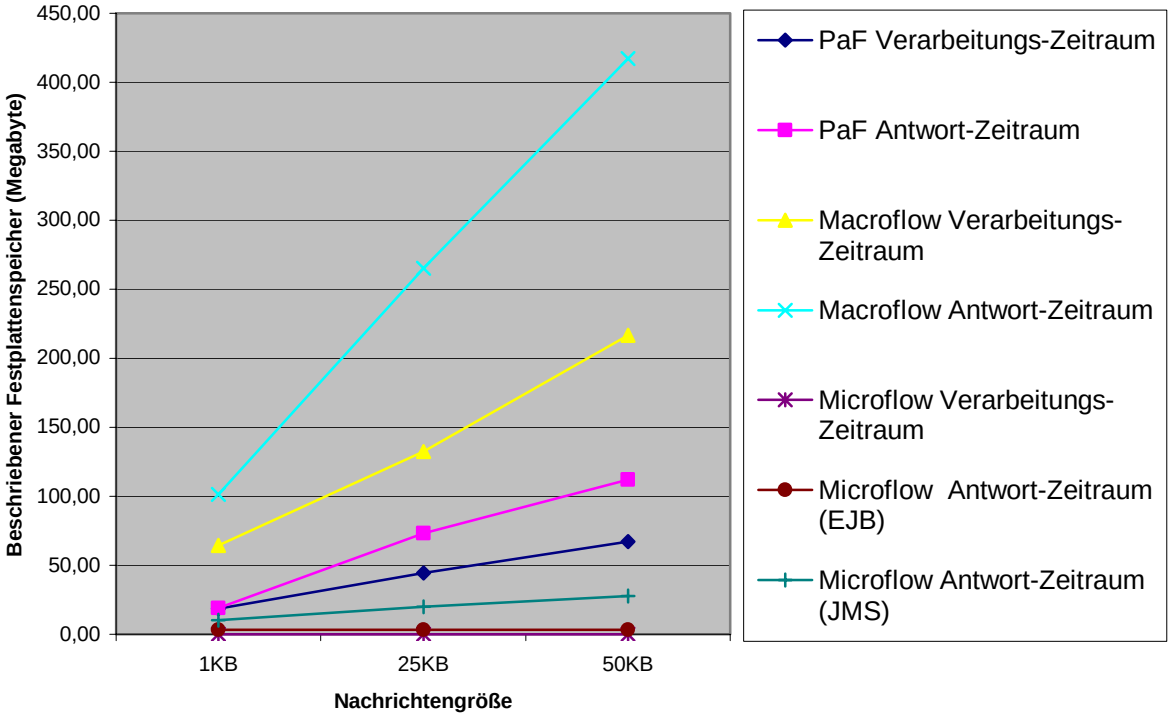
23 - Speicherallokation der Analyse II

9 - Speicherallokation der Analyse II enthält die Messdaten zu der vorangehenden Abbildung.

	1KB	**25KB**	**50KB**
PaF Verarbeitungs-Zeitraum	262,63	715,32	1234,01
PaF Antwort-Zeitraum	338,69	1115,13	2030,39
Macroflow Verarbeitungs-Zeitraum	704,56	1783,11	2830,38
Macroflow Antwort-Zeitraum	1208,20	3253,18	5315,01
Microflow Verarbeitungs-Zeitraum	49,64	260,63	462,82
Microflow Antwort-Zeitraum (EJB)	133,61	342,72	545,53
Microflow Antwort-Zeitraum (JMS)	278,47	1179,29	2105,88

9 - Speicherallokation der Analyse II

In Abbildung 24 - Beschriebener Festplattenspeicher der Analyse II ist schließlich der Speicherbedarf an persistentem Speicher aufgeführt. Auch hier zeigt sich wieder die Effizienz der Microflows und der große Bedarf an Ressourcen durch den Macroflow, während die PaF-Implementierung, auch im Bezug auf die Steigerungsrate, dazwischen liegt.



24 - Beschriebener Festplattenspeicher der Analyse II

In Tabelle 10 - Beschriebener Festplattenspeicher der Analyse II sind die Messdaten zu dieser Abbildung aufgeführt.

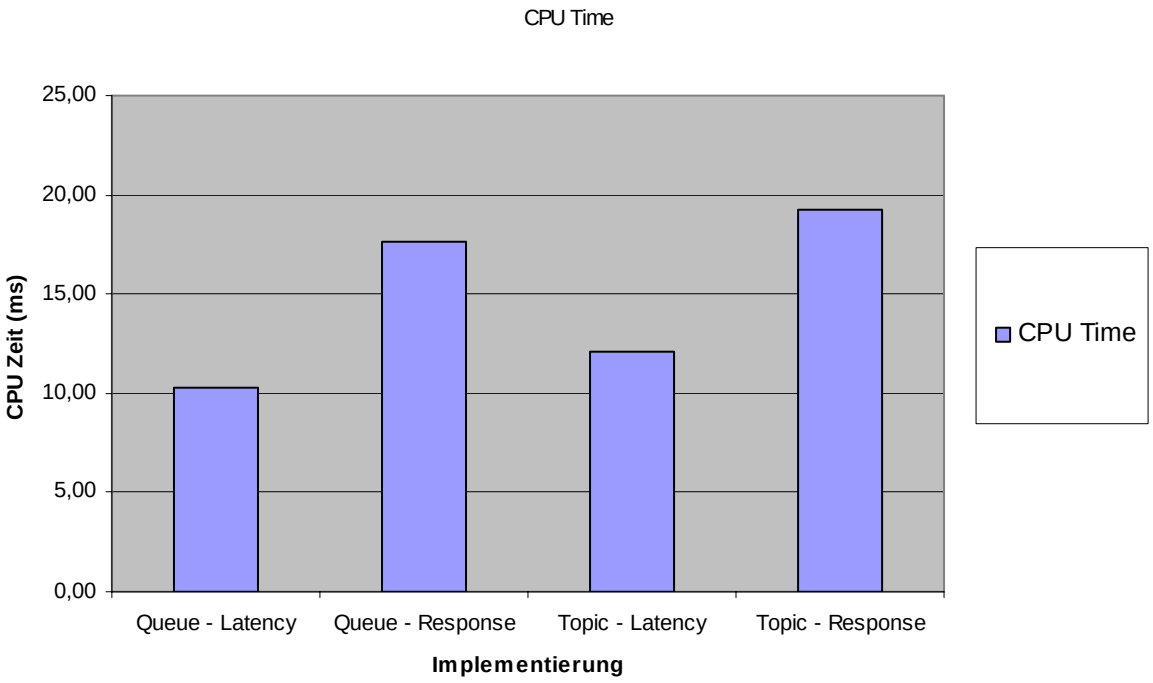
	1KB	**25KB**	**50KB**
PaF Verarbeitungs-Zeitraum	18,41	44,25	67,16
PaF Antwort-Zeitraum	18,76	73,08	112,09
Macroflow Verarbeitungs-Zeitraum	64,35	132,42	236,57
Macroflow Antwort-Zeitraum	101,26	265,32	416,97
Microflow Verarbeitungs-Zeitraum	0,11	0,00	0,00
Microflow Antwort-Zeitraum (EJB)	3,14	3,17	3,17
Microflow Antwort-Zeitraum (JMS)	10,24	19,99	27,70

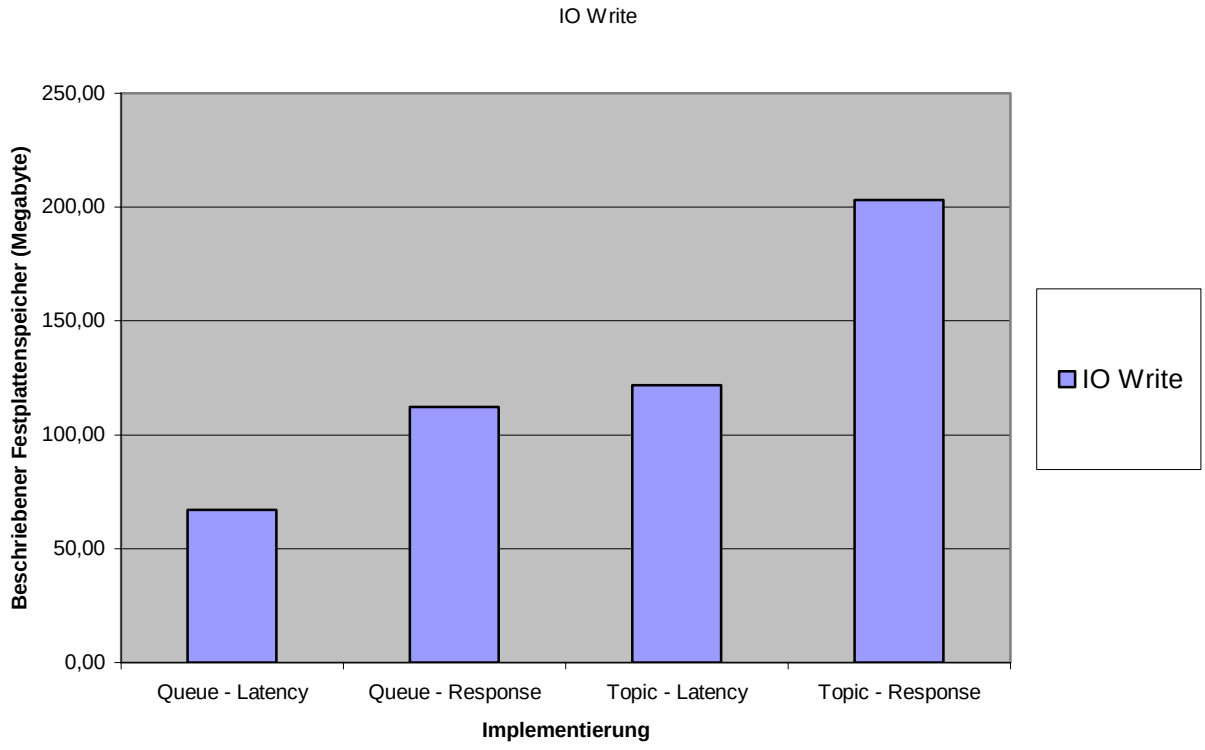
10 - Beschriebener Festplattenspeicher der Analyse II

6.3 Weitere Vergleiche

Speziell die Vergleiche in 6.2 haben gezeigt, dass die Microflow-Implementierung und die PaF-Implementierung vereinzelt ähnliche Ressourcenverbräuche und Steigerungsraten bezüglich der Ressourcenverbräuche aufweisen. Deswegen erfolgt nun eine vertiefte Analyse dieser beiden Implementierungen.
Deswegen wird nun zum einen überprüft, wie sich eine Implementierung des PaF-Musters basierend auf Topics verhält, indem diese Implementierung mit der bereits analysierten PaF-Implementierung, die auf Queues basiert, verglichen wird.
Zum anderen wird aber auch exemplarisch ein Ablauf, der sich durch eine höhere Komplexität auszeichnet, untersucht. Dazu wird jeweils das Scatter-Gather Muster mit einer Microflow-Implementierung und einer PaF-Implementierung gelöst und auf den Ressourcenverbrauch hin untersucht. Mit dieser Analyse kann zugleich auch festgestellt werden, inwieweit sich eine parallele Bearbeitung auf den Ressourcenverbrauch auswirkt.
Für beide genannten Vergleiche wird jeweils das Szenario 11 herangezogen, in dem vier Filterschritte von einer Nachricht mit der Größe 50Kilobyte durchlaufen werden.

Die Ergebnisse bezüglich der auf Topics basierenden Implementierung sind im Folgenden aufgeführt, wobei auf eine Abbildung zur Hauptspeicherallokation verzichtet wurde, da diese bei beiden Implementierungen nahezu identisch ist.
Abbildung 25 - Messdaten der Analyse III zeigt den leichten Anstieg in den Messdaten zur benötigten CPU Zeit und den deutlichen Anstieg bezüglich des verwendeten persistenten Speichers. Dieser erklärt sich dadurch, dass im Gegensatz zur Queue mehrere Empfänger existieren können und die Nachricht nicht gelöscht werden kann, sobald sie erfolgreich abgerufen wurde.





25 - Messdaten der Analyse III

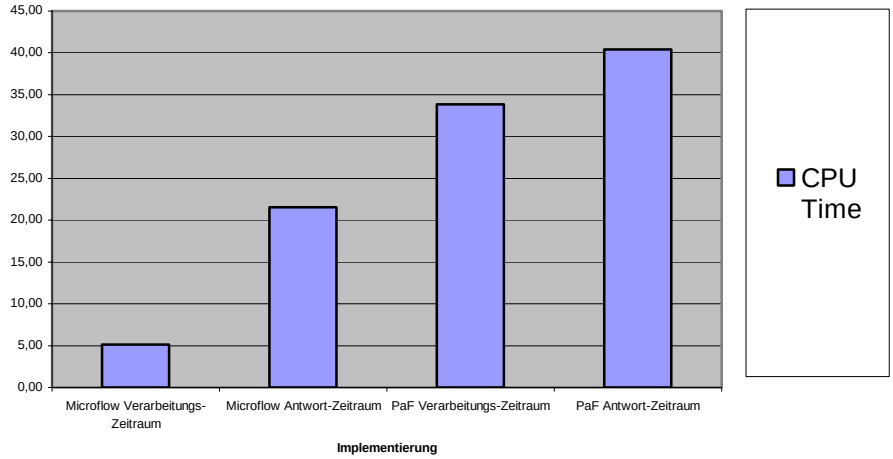
Das Ergebnis dieser ersten vertieften Analyse ist, dass durch Verwendung der Topics der Ressourcenverbrauch einer PaF-Implementierung ansteigt.

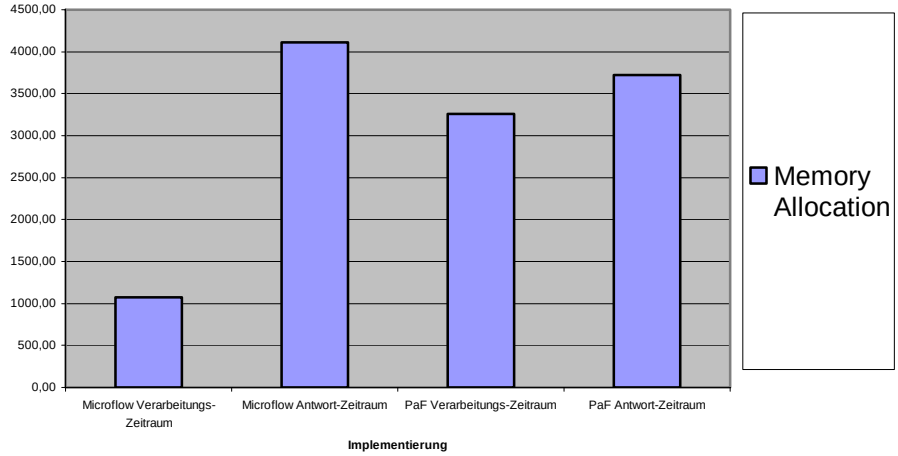
Die Daten zu den beiden Abbildungen sind in Abbildung 11 - Messdaten der Analyse III aufgeführt.

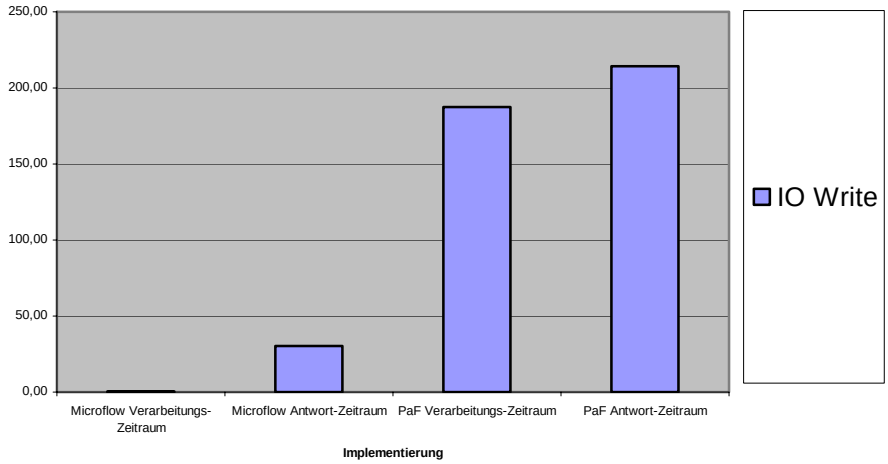
	CPU Time	IO Write
Queue - Latency	10,29	67,16
Queue - Response	17,67	112,09
Topic - Latency	12,08	121,91
Topic - Response	19,28	203,22

11 - Messdaten der Analyse III

Die nun folgenden Messdaten, die zum Scatter-Gather-Muster erhoben worden sind, unterstreichen die bisher gesammelte Erkenntnis, dass ein Microflow, speziell dann, wenn längere Verarbeitungsketten betrachtet werden, überaus effizient gegenüber einer PaF-Implementierung arbeitet.
Zu den verbrauchten Ressourcen siehe Abbildung 26 - Messdaten der Analyse IV.







26 - Messdaten der Analyse IV

Die Messdaten zu den Abbildungen sind in Tabelle 12 - Messdaten der Analyse IV abgebildet.

	CPU Time	Memory Allocation	IO Write
Microflow Verarbeitungs-Zeitraum	5,12	1071,84	0,21
Microflow Antwort-Zeitraum	21,52	4108,92	30,09
PaF Verarbeitungs-Zeitraum	33,87	3260,84	187,46
PaF Antwort-Zeitraum	40,40	3726,33	214,44

12 - Messdaten der Analyse IV

Bei diesen Ergebnissen zeigt sich also noch einmal deutlich, dass die Microflow-Implementierung während der Bearbeitung zwischenzeitlich keine Daten persistent speichert und so nicht nur wenig Festplattenspeicherplatz in Anspruch nimmt, sondern auch in der Dimension der benötigten CPU Zeit gegenüber der PaF-Implementierung im Vorteil ist.
Dies führt, auch wenn der Sockelverbrauch einer Instanziierung relativ hoch ist (siehe unter anderem die Diskrepanz in der CPU Zeit zwischen dem Antwortzeitraum und dem Verarbeitungszeitraum in der letzten Abbildung), über die gesamte Verarbeitungsdauer zu einem geringeren Ressourcenverbrauch und zeigt die Effizienz dieses Konzepts.
Im Vergleich mit dem sequentiellen Aufruf der Dienste verbraucht die Microflow-Implementierung dabei geringfügig mehr Ressourcen, während der Anstieg bei der PaF-Implementierung sehr viel größer ausfällt. Dies kann anhand der Tabelle 13 - Vergleichbare Messdaten nachvollzogen werden, in der die in den letzten beiden Abschnitten erhobenen Daten aufbereitet wurden.

	CPU Time	Memory Allocation	IO Write
Microflow Verarbeitungs-Zeitraum	4,17	462,82	0,00
Microflow Antwort-Zeitraum (JMS)	16,43	2105,88	27,70
PaF Verarbeitungs-Zeitraum	10,29	1234,01	67,16
PaF Antwort-Zeitraum	17,67	2030,39	112,09

13 - Vergleichbare Messdaten

7 Abschließende Betrachtung

Nachdem sowohl die theoretische, als auch die praktische Analyse stattgefunden hat, lassen sich nun die eingangs gesuchten Aussagen zum Ressourcenverbrauch treffen.

Während bei der Feststellung des Ressourcenverbrauchs für das „Pipes and Filter“ Muster eine Implementierung zur Analyse herangezogen wurde, erfolgte die Analyse des Ressourcenverbrauchs für den Workflow anhand von zwei Implementierungen. Hier wurdewurden zum einen ein Macroflow und zum anderen ein Microflow für die Implementierung verwendet, die das Szenario abbildet.

Obwohl der Microflow sich dabei durch eine niedrigere Dienstqualität als der Macroflow und die in dieser Arbeit betrachtete „Pipes and Filter“ Architektur auszeichnet, weil er ohne Unterbrechung arbeitet und keinerlei Sicherung der Zwischenergebnisse durchführt, kann die ebenfalls in dieser Arbeit betrachtete Sequenz von Dienstaufrufen mit diesem Konzept abgebildet werden. Dies liegt daran, dass nur synchrone Dienstaufrufe ohne Benutzereingaben geschehen und deshalb das Alles-Oder-Nichts-Prinzip einer Microflow-Transaktion kein Problem darstellt.

Auf der anderen Seite hat sich aber während der theoretischen Analyse herausgestellt, dass der Vergleich einer Implementierung des Pipes and Filter Musters eher mit dem Macroflow geschehen sollte, da, bedingt durch die Sicherung der Zwischenergebnisse, sich diese beiden Konzepte näher liegen.

Da das Szenario also eigentlich eine typische Anwendung des Microflows darstellt, die Dienstqualität der in dieser Arbeit betrachteten „Pipes and Filter“ Architektur aber eher den Macroflows folgt wurden letztendlich beide Konzepte zur Analyse herangezogen.

Die Messergebnisse folgen dabei schließlich auch zu einem großen Teil der gerade angesprochenen Dienstqualität.

So hat es sich gezeigt, dass der Microflow sehr effizient mit den Ressourcen des Systems umgeht, da er, einmal gestartet, in einem Thread und ohne Zugriffe auf den persistenten Speicher für einen Durchlauf lediglich den Hauptspeicher und die CPU verwendet. Hier stellt es auch kein Problem dar, dass sowohl die Instanziierung eines Microflows als auch die Verwendung eines JMS Proxy Dienstes, um den Microflow aufzurufen, einen hohen Sockelverbrauch an Ressourcen beanspruchen.

Die PaF-Architektur als alternatives Konzept liegt bezüglich ihres Ressourcenverbrauch prinzipiell zwischen dem Konzept des Microflows und dem des Macroflows, da hier, durch die persistente Sicherung der Zwischenergebnisse, mehr Aufwand betrieben wird als bei einem Microflow,.

aber andererseits keine Instanziierungen von komplexen Objekten und zusätzliche Zwischensicherungen der Navigationsstati geschehen, wie dies bei dem Macroflow der Fall ist.
Neben dem Niederschlag der Architektur-Charakteristika in absoluten Ressourcenverbräuchen zeigte sich dieses Verhalten zudem aber auch noch in den Steigerungsraten bezüglich der in den Szenarien veränderten Parametern ‚Anzahl der Nachrichten' und ‚Anzahl der Filterschritte' beziehungsweise ‚Anzahl der Aktivitäten'.
Da sich jedoch in der Analyse zur Auswirkung des Nachrichtenwachstums auch gezeigt hatte, dass sich die Steigerungsraten der Microflow-Implementierung und der PaF-Implementierung ähnlich sind wurde schließlich noch ein komplexerer Ablauf betrachtet und das Scatter-Gather Muster implementiert. Bei diesem Muster geschehen die Dienstaufrufe parallel und werden wieder zusammengeführt. Die Implementierungen hierzu haben die Effizienz und den, im Vergleich zur PaF-Implementierung, geringen Ressourcenverbrauchs der Microflow-Implementierung deutlich bestätigt, so dass die Aussagen bezüglich der Reihenfolge bekräftigt wird.

8 Glossar

Compile Sphere:
Die Compile Sphere ist ein Abschnitt, in dem ein kurzlaufender Prozessbestandteil stattfindet. Daher kann dieser Abschnitt mit all seinen Aktivitäten in ein ausführbares Programm kompiliert werden und verhält sich dann entsprechend eines -> Microflows.

kurzlaufende Prozesse:
Prozesse oder Prozessabschnitte, die von keinen Benutzereingaben oder asynchronen Diensten abhängen, werden kurzlaufende Prozesse genannt, da es möglich ist sie ohne Unterbrechung in einem Schritt abzuarbeiten. Die Implementierung eines solchen Prozesses kann durch einen -> Microflow erfolgen.

langlaufende Prozesse:
Prozesse oder Prozessabschnitte, die zumindest teilweise noch von Eingaben der Benutzern oder aber asynchronen Diensten abhängen, werden langlaufende Prozesse genannt, da ihre Bearbeitung zeitlich nicht genau bestimmt werden kann und mitunter lange dauert.

Macroflow: Konzept zur Ablaufsteuerung eines Workflows, bei dem die Zwischenergebnisse gesichert werde und deshalb Unterbrechung erfolgen können. Um dies zu gewährleisten erfolgen nach jeder Aktivität die notwendigen Sicherungen, während die Aktivitäten selber innerhalb einer Transaktion aublaufen.

Microflow: Konzept zur Ablaufsteuerung eines Workflows, bei dem keine Unterbrechungen erfolgen. Es dürfen daher weder asynchrone Dienstaufrufe noch Benutzereingaben geschehen. Der Microflow läuft innerhalb einer Transaktion und terminiert nur nachdem er alle seine Aktivitäten erfolgreich durchlaufen wurde.

„Pipes and Filter“-Architektur:
Architektur die dem Gestaltungsgrundsatz folgt die Funktionalität in den Filterkomponenten zu isolieren, und gleichzeitig durch die Verwendung der Pipes eine Verbindung zwischen diesen Komponenten herzustellen. So kann die Funktionalität beliebig kombiniert und verwendet werden. Das Muster wird an vielen Stellen der Informationsverarbeitung eingesetzt.

Resourcen:
Zeit-, Hauptspeicher-, und Rechenzeitbedarf eines Ablaufs, der durch den Start einer Anwendung bis zu deren Terminierung hervorgerufen wird.

Transaktion:
Die Durchführung einer Abfolge von Tätigkeiten, die als Ganzes dem ACID-Prinzip genügt, und deshalb auch im Fehlerfall auf einen konsistenten Zustand zurückgesetzt werden kann, wird Transaktion genannt.

Transaktionsgrenze:
Im Zusammenhang mit den betrachteten Architekturen, stellt sich die Frage, wo die Grenzen einer ->Transaktionen liegen. Geschehen Sicherungen der Zwischenschritte, so laufen die einzelnen Aktivitäten die den Übergang der Zwischenschritte darstellen, sinnvollerweise in einer Transaktion. Dies ist bei der Macroflow-Implementierung und der PaF-Implementierung auch der Fall. Bei der Microflow-Implementierung hingegen geschieht der gesamte Ablauf innerhalb einer Transaktion, da keine Sicherung der Zwischenschritte geschieht.

Web Service: Ein Web Service ist eine Software-Anwendung, die mit einem Uniform Resource Identifier (URI) eindeutig identifizierbar ist und deren Schnittstellen als XML-Artefakte definiert, beschrieben und gefunden werden können. Ein Web Service unterstützt die direkte Interaktion mit anderen Software-Agenten unter Verwendung XML-basierter Nachrichten durch den Austausch über internetbasierte Protokolle. Web Services spielen als Middleware im Bereich E-Business eine zunehmend bedeutende Rolle.

WfMS:
Abkürzung für Workflow Management System. Ein WfMS ist ein System zur regelbasierten Steuerung von Aktivitäten entlang einer Wertschöpfungskette, mit dem Ziel Geschäftsfunktionen, die in verschiedenen Applikationen oder Komponenten implementiert sind, in einer vordefinierten automatisierten Abfolge auszuführen und so applikationsübergreifende Services zu realisieren.

Workflow Enactment Service:
Zentrale Komponente det WfMS. In ihr werden alle Daten für die einzelnen Prozessinstanzen und deren Aktivitäten vorgehalten. Zudem enthält dieses Komponente die Workflow engine (oder mehrere), die die Kontrolle über die Bearbeitung der Prozessinstanzen hat.

Workflow Engine:
Zentrale Kompomenente des Workflow Enactment Service. In ihr geschieht die Instanziierung, Aktivierung, Unterbrechung und Terminierung von Prozessinstanzen, sowie die Navigation und die Aktionsgenerierung und -überwachung.

Literaturverzeichnis:

[1] Leymann, Frank; Roller, Dieter: Production Workflow - Concepts and Techniques. New Jersey: PTR Prentice Hall 2000

[2] Vitria Integration Platform Businessware. URL: http://www.vitria.com/products/platform [Stand: 19.04.2006]

[3] WebSphere Process Server. URL: http://www-306.ibm.com/software/integration/wps [Stand: 19.04.2006]

[4] Microsoft BizTalk Server. URL: http://www.microsoft.com/biztalk/default.mspx [Stand: 19.04.2006]

[5] Workflow Management Coalition: "The Workflow Reference Model". 19.Jan 1995. URL: http://www.wfmc.org/standards/docs/tc003v11.pdf [Stand: 19.04.2006]

[6] BPEL Specification. URL: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ [Stand 20.08.2006]

[7] Injey, Franck; Ajab, Mohammed; Meira, Pedro Duarte; Gardner, Andrew; Verboven, Bert; Williams, Frances; Weller, Rica: Flexible Self Service Application Patterns Using Websphere and Process Choreography on z/OS. IBM 2005 (Teil der Redbook Serie)

[8] Glatz, Eduard: Betriebssysteme – Grundlagen, Konzepte, Systemprogrammierung. Heidelberg: dpunkt.verlag 2006

[9] Hohpe, Gregor; Woolf, Bobby; Enterprise Integration Patterns – Designing, Building and Deploying Messaging Solutions. Boston: Pearson Education, Inc 2004

[10] Overview of Performance Monitoring. URL: http://www.microsoft.com/technet/prodtechnol/Windows2000Pro/reskit/part6/proch27.mspx?mfr=true [Stand: 20.08.2006]

[11] WMI Architecture. URL: http://windowssdk.msdn.microsoft.com/library/default.asp?url=/library/WMISDK/wmi/wmi architecture.asp [Stand: 18.05.2006]

[12] NSClient4j. URL: https://nsclient4j.dev.java.net/ [Stand: 20.08.2006]

[13] JavaTM Virtual Machine Profiler Interface (JVMPI). URL: http://java.sun.com/j2se/1.3/docs/guide/jvmpi/jvmpi.html [Stand: 18.05.2006]

[14] Bugzilla Bug 70660 “IBM JDK Bug: Missing inlined methods from trace when profiling with JIT enabled” URL: https://bugs.eclipse.org/bugs/show_bug.cgi?id=70660 [Stand: 22.08.2006]

A. Macroflow.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<process expressionLanguage="Java" name="MacroFlow" suppressJoinFailure="yes"
targetNamespace="http://www.example.com/process51954416/" wpc:autoDelete="no" wpc:autonomy="peer"
wpc:businessRelevant="yes" wpc:compensationSphere="notSupported" wpc:displayName="MacroFlow"
wpc:executionMode="longRunning" wpc:validFrom="2003-01-01T00:00:00"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:wpc="http://www.ibm.com/xmlns/prod/websphere/business-process/v5.1/"
xmlns:wsdl="http://www.example.com/process51954416/interface/"
xmlns:wsdl0="http://www.example.com/process51954416/"
xmlns:wsdl1="http://bean.sample/InterfaceImplementation1/" xmlns:wsdl2="http://Message"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/business-process/
http://schemas.xmlsoap.org/ws/2003/03/business-process/">
   <wpc:javaGlobals>
      <wpc:import packages="com.example.www.process51954416._interface_msg.OutputMessageMessage"/>
      <wpc:import packages="java.io.File"/>
      <wpc:import packages="java.io.IOException"/>
      <wpc:import packages="java.io.InputStream"/>
      <wpc:import packages="java.io.FileInputStream"/>
      <wpc:import packages="java.io.FileNotFoundException"/>
      <wpc:import packages="java.util.*"/>
      <wpc:import packages="tools.*"/>
      <wpc:import packages="org.apache.wsif.WSIFException"/>
   </wpc:javaGlobals>
   <partnerLinks>
      <partnerLink myRole="ProcessRole" name="ProcessInterface"
partnerLinkType="wsdl:PartnerLinkType"/>
      <partnerLink name="InterfaceImplementation1" partnerLinkType="wsdl0:InterfaceImplementation1LT"
partnerRole="InterfaceImplementation1Role"/>
   </partnerLinks>
   <variables>
      <variable messageType="wsdl2:Message" name="Input"/>
   </variables>
   <sequence name="Sequence" wpc:displayName="Sequence" wpc:id="1073741825">
      <receive createInstance="yes" name="Receive" operation="StartMacroFlow"
partnerLink="ProcessInterface" portType="wsdl:ProcessPortType" variable="Input"
wpc:displayName="Receive" wpc:id="2"/>
      <invoke name="StartProfiling" operation="null" partnerLink="null" portType="wpc:null"
wpc:displayName="StartProfiling" wpc:id="13">
         <wpc:script>
            <wpc:javaCode><![CDATA[  boolean LATENCY = true;
 if (LATENCY){
   SystemInformationProxy proxy;
   try {
     proxy = new SystemInformationProxy();
     proxy.startProfileIncrease();
     //System.out.println("Start profiling after process call!");
   } catch (WSIFException e) {
          e.printStackTrace();
   }
 }]]></wpc:javaCode>
         </wpc:script>
      </invoke>
```

```
      <invoke inputVariable="Input" name="Invoke1" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke1" wpc:id="4">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke inputVariable="Input" name="Invoke2" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke2" wpc:id="6">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke inputVariable="Input" name="Invoke3" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke3" wpc:id="8">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke inputVariable="Input" name="Invoke4" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke4" wpc:id="10">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke name="StopProfiling" operation="null" partnerLink="null" portType="wpc:null"
wpc:displayName="StopProfiling" wpc:id="14">
         <wpc:script>
            <wpc:javaCode><![CDATA[  boolean LATENCY = true;
  if (LATENCY){
    SystemInformationProxy proxy;
    try {
      proxy = new SystemInformationProxy();
      proxy.stopProfileIncrease();
      //System.out.println("Stop profiling!");
    } catch (WSIFException e) {
           e.printStackTrace();
    }
  }]]></wpc:javaCode>
         </wpc:script>
      </invoke>
      <reply name="Reply" operation="StartMacroFlow" partnerLink="ProcessInterface"
portType="wsdl:ProcessPortType" variable="Input" wpc:displayName="Reply" wpc:id="3"/>
   </sequence>
</process>
```

B. **Microflow.bpel**

```
<?xml version="1.0" encoding="UTF-8"?>
<process expressionLanguage="Java" name="MicroFlow" suppressJoinFailure="yes"
targetNamespace="http://www.example.com/process51954416/" wpc:autoDelete="no"
wpc:businessRelevant="yes" wpc:compensationSphere="supports" wpc:displayName="MicroFlow"
wpc:executionMode="microflow" wpc:validFrom="2003-01-01T00:00:00"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:wpc="http://www.ibm.com/xmlns/prod/websphere/business-process/v5.1/"
xmlns:wsdl="http://www.example.com/process51954416/interface/"
xmlns:wsdl0="http://www.example.com/process51954416/"
xmlns:wsdl1="http://bean.sample/InterfaceImplementation1/" xmlns:wsdl2="http://Message"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/business-process/
http://schemas.xmlsoap.org/ws/2003/03/business-process/">
   <wpc:javaGlobals>
      <wpc:import packages="com.example.www.process51954416._interface_msg.OutputMessageMessage"/>
      <wpc:import packages="java.io.File"/>
      <wpc:import packages="java.io.IOException"/>
      <wpc:import packages="java.io.InputStream"/>
      <wpc:import packages="java.io.FileInputStream"/>
      <wpc:import packages="java.io.FileNotFoundException"/>
      <wpc:import packages="java.util.*"/>
      <wpc:import packages="tools.*"/>
      <wpc:import packages="org.apache.wsif.WSIFException"/>
   </wpc:javaGlobals>
   <partnerLinks>
      <partnerLink myRole="ProcessRole" name="ProcessInterface"
partnerLinkType="wsdl:PartnerLinkType"/>
      <partnerLink name="InterfaceImplementation1" partnerLinkType="wsdl0:InterfaceImplementation1LT"
partnerRole="InterfaceImplementation1Role"/>
   </partnerLinks>
   <variables>
      <variable messageType="wsdl2:Message" name="Input"/>
   </variables>
   <sequence name="Sequence" wpc:displayName="Sequence" wpc:id="1073741825">
      <receive createInstance="yes" name="Receive" operation="StartMicroFlow"
partnerLink="ProcessInterface" portType="wsdl:ProcessPortType" variable="Input"
wpc:displayName="Receive" wpc:id="2"/>
      <invoke name="StartProfiling" operation="null" partnerLink="null" portType="wpc:null"
wpc:displayName="StartProfiling" wpc:id="13">
         <wpc:script>
            <wpc:javaCode><![CDATA[
 boolean LATENCY = true;
 if (LATENCY){
  SystemInformationProxy proxy;
  try {
   proxy = new SystemInformationProxy();
   proxy.startProfileIncrease();
   //System.out.println("Start profiling after process call!");
  } catch (WSIFException e) {
        e.printStackTrace();
  }
 }
 ]]></wpc:javaCode>
         </wpc:script>
```

```
      </invoke>
      <invoke inputVariable="Input" name="Invoke1" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke1" wpc:id="4">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke inputVariable="Input" name="Invoke2" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke2" wpc:id="6">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke inputVariable="Input" name="Invoke3" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke3" wpc:id="8">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke inputVariable="Input" name="Invoke4" operation="doNoOp" outputVariable="Input"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"
wpc:displayName="Invoke4" wpc:id="10">
         <wpc:undo doActionIsTransacted="yes" inputVariable="Input" operation="doNoOp"
partnerLink="InterfaceImplementation1" portType="wsdl1:InterfaceImplementation1"/>
      </invoke>
      <invoke name="StopProfiling" operation="null" partnerLink="null" portType="wpc:null"
wpc:displayName="StopProfiling" wpc:id="14">
         <wpc:script>
            <wpc:javaCode><![CDATA[
 boolean LATENCY = true;
 if (LATENCY){
   SystemInformationProxy proxy;
   try {
     proxy = new SystemInformationProxy();
     proxy.stopProfileIncrease();
     //System.out.println("Stop profiling!");
   } catch (WSIFException e) {
          e.printStackTrace();
   }
 }
 ]]></wpc:javaCode>
         </wpc:script>
      </invoke>
      <reply name="Reply" operation="StartMicroFlow" partnerLink="ProcessInterface"
portType="wsdl:ProcessPortType" variable="Input" wpc:displayName="Reply" wpc:id="3"/>
   </sequence>
</process>
```

C. Aufbau der Message Driven Beans

C.1 Client MDB

```
/**
* onMessage – Message has been received on Pipe
*/
public void onMessage(javax.jms.Message msg) {
   try{
      //1) GET MESSAGE
      message = ((TextMessage)msg).getText();
      String messageFlowControl = msg.getStringProperty("MessageFlowControl");
      //2) PROFILING
      //2.1) CASE OBJECTS PER MINUTE
      if (OBJECTS_PER_MINUTE){
         if (messageFlowControl.equalsIgnoreCase("Stop"));
            //Stop Counting
            counter = 0;
            firstMessageReceived = false;
            return;
         } else if (firstMessageReceived){
            //Start Counting
            firstMessageReceived = true;
            SystemInformationProxy proxy = new SystemInformationProxy();
            proxy.setCurrentTime();
            time = proxy.getCurrentTime();
            time = time + (INTERVALL*1000);
            counter = counter + 1;
            System.out.println("Start counting Objects per Intervall!");
         } else {
            //Do Counting Step
            if (System.currentTimeMillis() < time){
               counter = counter + 1;
            } else{
               System.out.println("Es wurden " + counter + " Durchläufe in " + INTERVALL + " Sekunden
                                   durchgeführt!");
            System.out.println("Reset-Vorgang wird gestartet!");
            messageFlowControl = "Stop";
         }
      }

      //2.2) CASE LATENCY
      } else if (LATENCY){
         if (messageFlowControl.equalsIgnoreCase("Stop"));
            //Stop Counting
            counter = 0;
            firstMessageReceived = false;
            return;
         }
         if (firstMessageReceived){
            firstMessageReceived = true;
            //Initialize Profiler
            SystemInformationProxy proxy = new SystemInformationProxy();
            proxy.initializeProfilers("PaF");
            proxy.startCPUUsageCalculation();
            System.out.println("Start profiling Latency!");
```

```
    } else {
        if (counter < OUTER*INNER){
        //Do Profiling Step
        counter = counter + 1;
        SystemInformationProxy proxy = new SystemInformationProxy();
        if ((counter%(OUTER*INNER))==0){
            //Create Profiling Information and Initiate Profiling End
            proxy.printProfilingInformation();
            proxy.stopCPUUsageCalculation();
            proxy.interruptProfilers();
            System.out.println("Profiling-Execution-End (Latency)");
            System.out.println("Reset-Vorgang wird gestartet!");
            messageFlowControl = "Stop";
        } else if ((counter%INNER)==0){
            //Create Profiling Information
            proxy.printProfilingInformation();
            System.out.println("Profiling-Information written (Latency)");
        }
    }
}

//2.3) CASE RESPONSE
} else {
    SystemInformationProxy proxy = new SystemInformationProxy();
    if (messageFlowControl.equalsIgnoreCase("Stop")){
        //Stop Profiling
        firstMessageReceived = false;
       counter = 0;
       return;
    } else if (firstMessageReceived){
        //Start Profiling
        firstMessageReceived = true;
        proxy.initializeProfilers("PaF");
        proxy.startCPUUsageCalculation();
        proxy.startProfileIncrease();
        System.out.println("Start profiling Response!");
    } else {
        //Do Profiling Step
        if (counter < OUTER*INNER){
            counter = counter + 1;
            proxy.stopProfileIncrease();
            if ((counter%(OUTER*INNER))==0){
                //Create Profiling Information and Initiate Profiling End
                proxy.printProfilingInformation();
                proxy.stopCPUUsageCalculation();
                proxy.interruptProfilers();
                System.out.println("Profiling-Execution-End (Response)");
                System.out.println("Reset-Vorgang wird gestartet!");
                messageFlowControl = "Stop";
             } else if ((counter%INNER)==0){
                 //Create Profiling Information
                proxy.printProfilingInformation();
                 System.out.println("Profiling-Information written (Response)");
                proxy.startProfileIncrease();
            } else {
                proxy.startProfileIncrease();
            }
        }
```

```
            }
        }

        //3) SEND MESSAGE
        sendMessage(message, messageFlowControl, "Pipe1_CF", "Pipe1_Queue1");
    } catch (Exception e) {
        e.getMessage();
        e.printStackTrace();
    }
}
```

C.2 Filter1 MDB

```
/**
* onMessage – Message has been received on Pipe
*/
public void onMessage(javax.jms.Message msg) {
    try{
        //System.out.println("Filter1: " + ((TextMessage)msg).getText());
        //1) GET MESSAGE
        String message = ((TextMessage)msg).getText();
        String messageFlowControl = msg.getStringProperty("MessageFlowControl");
        //2) PROFILING
        //2.1) CASE LATENCY
        if ((LATENCY)&&(!OBJECTS_PER_MINUTE)){
            //Start Profiling
            SystemInformationProxy proxy = new SystemInformationProxy();
            //System.out.println("Start profling!");
            proxy.startProfileIncrease();
        }
        //3) DO INTEGRATION
        InterfaceImplementation1Proxy aProxy = new InterfaceImplementation1Proxy();
        String returnValue = aProxy.doNoOp(((TextMessage)msg).getText());
        //4) SEND MESSAGE
        sendMessage(message, messageFlowControl,"Pipe2_CF" , "Pipe2_Queue1"); //send message to Pipe2
    } catch (Exception e) {
        e.getMessage();
        e.printStackTrace();
    }
}
```

C.3 Filter2 MDB und Filter3 MDB

```
/**
* onMessage – Message has been received on Pipe
*/
public void onMessage(javax.jms.Message msg) {
    try{
        //System.out.println("Filter2: " + ((TextMessage)msg).getText());
        //DO NoOp
        InterfaceImplementation1Proxy aProxy = new InterfaceImplementation1Proxy();
        String returnValue = aProxy.doNoOp(((TextMessage)msg).getText());
        //Send message
```

```
        sendMessage(((TextMessage)msg).getText(), msg.getStringProperty("MessageFlowControl"),
                        "Pipe3_CF", "Pipe3_Queue1");
    } catch (Exception e) {
        e.getMessage();
        e.printStackTrace();
    }
}
```

C.4 Filter4 MDB

```
/**
* onMessage – Message has been received on Pipe
*/
public void onMessage(javax.jms.Message msg) {
    try{
        //System.out.println("Filter1: " + ((TextMessage)msg).getText());
        //1) GET MESSAGE
        String message = ((TextMessage)msg).getText();
        String messageFlowControl = msg.getStringProperty("MessageFlowControl");
        //2) DO INTEGRATION
        InterfaceImplementation1Proxy aProxy = new InterfaceImplementation1Proxy();
        String returnValue = aProxy.doNoOp(((TextMessage)msg).getText());
        //3) PROFILING
        //3.1) CASE LATENCY
        if ((LATENCY)&&(!OBJECTS_PER_MINUTE)){
            //Start Profiling
            SystemInformationProxy proxy = new SystemInformationProxy();
            //System.out.println("Start profling!");
            proxy.startProfileIncrease();
        }

        //4) SEND MESSAGE
        sendMessage(message, messageFlowControl, "PipeStart_CF", "PipeStart_Queue1");
    } catch (Exception e) {
        e.getMessage();
        e.printStackTrace();
    }
}
```

D. **Messdaten**

	Time			CPU Time			Memory Allocation		
	1Filter	**4Filter**	**8Filter**	**1Filter**	**4Filter**	**8Filter**	**1Filter**	**4Filter**	**8Filter**
PaF Verarbeitungs-Zeitraum	2,84	23,55	59,92	1,11	10,29	22,77	19,48	1234,01	2853,08
PaF Antwort-Zeitraum	18,00	41,46	70,77	7,13	17,67	30,12	816,29	2030,39	3649,74
Macroflow Verarbeitungs-Zeitraum	39,23	148,381	561,11	11,13	40,35	88,52	533,59	2830,38	5678,72
Macroflow Antwort-Zeitraum	157,87	310,04	559,54	43,95	76,08	122,58	3057,93	5315,01	8118,90
Microflow Verarbeitungs-Zeitraum	3,17	8,18	12,43	1,34	4,17	6,55	126,65	462,82	908,68
Microflow Antwort-Zeitraum (EJB)	9,64	15,17	20,32	4,51	7,73	10,43	176,76	545,53	1035,43
Microflow Antwort-Zeitraum (JMS)	22,32	32,68	43,67	11,06	16,43	22,15	1493,82	2105,88	2921,91
	IO Read			**IO Write**			**CPU Usage**		
	1Filter	**4Filter**	**8Filter**	**1Filter**	**4Filter**	**8Filter**	**1Filter**	**4Filter**	**8Filter**
PaF Verarbeitungs-Zeitraum	0,00	0,05	0,80	0,02	67,16	158,11	0,42	0,43	0,37
PaF Antwort-Zeitraum	0,05	0,21	1,44	44,77	112,09	201,51	0,40	0,43	0,42
Macroflow Verarbeitungs-Zeitraum	5,85	1,47	2,35	57,08	216,71	403,28	0,30	0,26	0,15
Macroflow Antwort-Zeitraum	21,60	8,51	6,77	249,28	416,97	640,39	0,29	0,26	0,22
Microflow Verarbeitungs-Zeitraum	0,00	0,00	0,00	0,00	0,00	0,00	0,47	0,48	0,00
Microflow Antwort-Zeitraum (EJB)	0,00	0,00	0,01	1,67	3,17	5,84	0,44	0,47	0,50
Microflow Antwort-Zeitraum (JMS)	0,33	0,39	0,34	24,80	27,70	30,56	0,48	0,50	0,47

14 - Messdaten zur Anzahl der Filter

	Time			CPU Time			Memory Allocation		
	1KB	**25KB**	**50KB**	**1KB**	**25KB**	**50KB**	**1KB**	**25KB**	**50KB**
PaF Verarbeitungs-Zeitraum	16,86	20,80	23,55	7,98	9,05	10,29	262,63	715,32	1234,01
PaF Antwort-Zeitraum	13,97	32,52	41,46	10,93	14,32	17,67	338,69	1115,13	2030,39
Macroflow Verarbeitungs-Zeitraum	100,60	143,35	148,38	24,68	31,28	40,35	704,56	1783,11	2830,38
Macroflow Antwort-Zeitraum	179,00	263,81	310,04	49,95	62,34	76,08	1208,20	3253,18	5315,01
Microflow Verarbeitungs-Zeitraum	4,01	5,65	8,18	1,84	2,44	4,17	49,64	260,63	462,82
Microflow Antwort-Zeitraum (EJB)	10,51	13,11	15,17	5,15	6,19	7,73	133,61	342,72	545,53
Microflow Antwort-Zeitraum (JMS)	20,45	24,98	32,68	8,45	12,34	16,43	278,47	1179,29	2105,88
	IO Read			**IO Write**			**CPU Usage**		
	1KB	**25KB**	**50KB**	**1KB**	**25KB**	**50KB**	**1KB**	**25KB**	**50KB**
PaF Verarbeitungs-Zeitraum	0,00	0,00	0,05	18,41	44,25	67,16	0,46	0,43	0,43
PaF Antwort-Zeitraum	0,06	0,06	0,21	18,76	73,08	112,09	0,41	0,44	0,43
Macroflow Verarbeitungs-Zeitraum	0,90	1,20	1,47	64,35	132,42	216,71	0,28	0,23	0,26
Macroflow Antwort-Zeitraum	3,75	3,52	8,51	101,26	265,32	416,97	0,28	0,24	0,26
Microflow Verarbeitungs-Zeitraum	0,00	0,00	0,00	0,11	0,00	0,00	0,48	0,47	0,48
Microflow Antwort-Zeitraum (EJB)	0,00	0,00	0,00	3,14	3,17	3,17	0,48	0,47	0,47
Microflow Antwort-Zeitraum (JMS)	0,28	0,29	0,39	10,24	19,99	27,70	0,42	0,48	0,50

15 - Messdaten zur Nachrichtengröße

	Time	**CPU Time**	**Memory Allocation**	**IO Read**	**IO Write**	**CPU Usage**
Queue Verarbeitungs-Zeitraum	23,55	10,29	1234,01	0,05	67,16	0,43
Queue Antwort-Zeitraum	41,46	17,67	2030,39	0,21	112,09	0,43
Topic Verarbeitungs-Zeitraum	29,35	12,08	1248,70	0,22	121,91	0,42
Topic Antwort-Zeitraum	48,36	19,28	2055,50	0,21	203,22	0,40

16 - Messdaten zu den Topics

	Time	CPU Time	Memory Allocation	IO Read	IO Write	CPU Usage
Microflow Verarbeitungs-Zeitraum	10,48	5,12	1071,84	0,00	0,21	0,36
Microflow Antwort-Zeitraum	40,22	21,52	4108,92	2,66	30,09	0,53
PaF Verarbeitungs-Zeitraum	1166,55	33,87	3260,84	21,34	187,46	0,03
PaF Antwort-Zeitraum	1028,82	40,40	3726,33	21,56	214,44	0,04

17 - Messdaten zu dem Scatter and Gather Muster