

Fachhochschule  
FH JOANNEUM Gesellschaft mbH

# Enterprise Application Integration

Messaging in Industrieanlagen - Anforderungen, Marktsituation  
und Architekturparadigmen

## Diplomarbeit

zur Erlangung des akademischen Grades eines/einer  
Diplomingenieurs für technisch-wissenschaftliche Berufe (FH)  
Diplomingenieurin für technisch-wissenschaftliche Berufe (FH)

eingereicht am  
Studiengang Informationsmanagement

**Betreuer:** FH-Prof. DI Peter Salhofer  
**Firmenbetreuer:** DI(FH) Dieter Zeiml; SSI Schäfer PEEM

**eingereicht von:** Alexander Josef Markl  
**Personenkennzahl:** 0510062025

August 2009

# **Eidesstattliche Erklärung**

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Graz, August 2009

Alexander Josef Markl

# Inhaltsverzeichnis

<b>1 Allgemeines</b>	<b>i</b>
1.1 Abstract - Deutsch . . . . .	i
1.2 Abstract - English . . . . .	iii
1.3 Unternehmensvorstellung . . . . .	iv
1.4 Abgrenzung . . . . .	vi
1.5 Problembeschreibung . . . . .	vii
1.5.1 Problembereiche des Systems . . . . .	viii
1.5.2 Stärken des Systems . . . . .	xii
<b>2 Integration von Systemen</b>	<b>1</b>
2.1 Enterprise Application Integration . . . . .	2
2.1.1 File Transfer . . . . .	3
2.1.2 Shared Database . . . . .	4
2.1.3 Remote Procedure Call . . . . .	5
2.1.4 Messaging . . . . .	6
2.2 Messaging-Architektur (EAI Messaging Patterns) . . . . .	7
2.2.1 Grundfunktion und Architektur eines Messagingsystems . . . . .	8
2.2.2 Message Channels . . . . .	10
2.2.3 Messaging Endpoints . . . . .	17
2.2.4 Message Construction . . . . .	23
2.2.5 Message Routing . . . . .	32
2.2.6 Message Transformation . . . . .	36
2.2.7 System Management . . . . .	39
2.3 Architekturempfehlung . . . . .	43
<b>3 Evaluierung von Messagingsystemen</b>	<b>47</b>
3.1 Anforderungsprofil, Definitionen und Evaluierungen . . . . .	49
3.1.1 Applikationsanforderungen . . . . .	49
3.1.2 Systemplattform . . . . .	59
3.1.3 Anbieterbezogene Kriterien . . . . .	61
3.2 Kriterienkatalog und Bewertungsliste . . . . .	65
3.2.1 Einteilung der Kriterien . . . . .	66
3.2.2 Skalierung der Kriterien . . . . .	67

3.2.3	Gewichtung der Kriterien . . . . .	68
3.2.4	Berechnung der Punkte . . . . .	69
3.2.5	Berechnung der Nutzwerte . . . . .	69
3.2.6	Berechnung der effektiven Nutzwerte . . . . .	70
3.2.7	Kriterienkatalog . . . . .	70
3.2.8	KO-Kriterienliste . . . . .	71
3.2.9	Bewertungsliste . . . . .	72
3.2.10	Weitere Faktoren . . . . .	73
3.3	Evaluierung . . . . .	80
3.3.1	Marktanalyse . . . . .	80
3.3.2	Grobevaluierung . . . . .	81
3.3.3	Detailevaluierung . . . . .	100
3.3.4	Gegenüberstellung der Produkte . . . . .	113
3.3.5	Produktempfehlung . . . . .	116
<b>4</b>	<b>Conclusio</b>	<b>118</b>
<b>A</b>	<b>Anhang</b>	<b>1</b>
A.1	Kontakte . . . . .	1
A.2	Testumgebung . . . . .	2
A.3	UML . . . . .	2
A.4	Listings . . . . .	4
A.4.1	Provider Properties . . . . .	4
A.4.2	Abstrakte Klassen . . . . .	5
A.4.3	Handler Klassen . . . . .	12
A.4.4	Implementationen Message Endpoint . . . . .	16
A.4.5	Start Klassen . . . . .	17
A.4.6	Implementationen Connection Handler . . . . .	21
A.5	Kommentare zu den Bewertungen . . . . .	28
A.5.1	Apache ActiveMQ . . . . .	28
A.5.2	Sun OpenMQ . . . . .	31
A.5.3	Progress SonicMQ . . . . .	33
A.5.4	Fiorano FioranoMQ . . . . .	35

## Abbildungsverzeichnis

1.1	SSI Schäfer Promo . . . . .	iv
-----	-----------------------------	----

1.2	Systemübersicht . . . . .	vii
1.3	Boss Queues ([Manowarda1999], S.8) . . . . .	x
1.4	Evaluierung SSI Schäfer PEEM Messaging System . . . . .	xi
2.1	EAI Ebenenmodell nach Ring ([Dangelmaier2002] zit. nach K. Ring) . . . . .	2
2.2	File Transfer (vgl. [Hohpe2005], S.44) . . . . .	3
2.3	Shared Database (vgl. [Hohpe2005], S.48) . . . . .	4
2.4	Remote Procedure Call (vgl. [Hohpe2005], S.51) . . . . .	5
2.5	Messaging (vgl. [Hohpe2005], S.54) . . . . .	6
2.6	Architekturen von Messagingsystemen (vgl. [Hohpe2005], S.322, 324) . . . . .	8
2.7	Benötigte Verbindungen: Point-to-Point vs. Hub-and-Spoke . . . . .	9
2.8	EAI-Patterns Overview (vgl. [Hohpe2005], Buchdeckel hinten) . . . . .	10
2.9	Message Channel ([Hohpe2005], S.61) . . . . .	11
2.10	Point-to-Point Channel ([Hohpe2005], S.103) . . . . .	11
2.11	Publish-Subscribe Channel ([Hohpe2005], S.107) . . . . .	13
2.12	Invalid Message Channel ([Hohpe2005], S.116) . . . . .	14
2.13	Point-to-Point Channel ([Hohpe2005], S.120) . . . . .	15
2.14	Point-to-Point Channel ([Hohpe2005], S.123) . . . . .	16
2.15	Messaging Endpoints ([Hohpe2005], S.96) . . . . .	17
2.16	Messaging Gateway ([Hohpe2005], S.469) . . . . .	19
2.17	Polling Consumer ([Hohpe2005], S.494) . . . . .	20
2.18	Event-driven Consumer ([Hohpe2005], S.498) . . . . .	21
2.19	Selective Consumer ([Hohpe2005], S.516) . . . . .	22
2.20	Command Message ([Hohpe2005], S.145) . . . . .	24
2.21	Document Message ([Hohpe2005], S.148) . . . . .	26
2.22	Event Message ([Hohpe2005], S.152) . . . . .	27
2.23	Request-Reply ([Hohpe2005], S.155) . . . . .	29
2.24	Correlation Identifier ([Hohpe2005], S.164) . . . . .	31
2.25	Message Router ([Hohpe2005], S. 80) . . . . .	32
2.26	Content-based Router ([Hohpe2005], S.232) . . . . .	33
2.27	Dynamic Router ([Hohpe2005], S.244) . . . . .	34
2.28	Resequencer ([Hohpe2005], S.284) . . . . .	35
2.29	Message Broker ([Hohpe2005], S.325) . . . . .	36
2.30	Message Translator ([Hohpe2005], S.86) . . . . .	37
2.31	Canonical Data Model ([Hohpe2005], S. 356, 359) . . . . .	38
2.32	Anzahl benötigter Message Translator . . . . .	39

2.33 Control Bus ([Hohpe2005], S.541) . . . . .	40
2.34 Wire Tap ([Hohpe2005], S.548) . . . . .	41
2.35 Message Store ([Hohpe2005], S. 556) . . . . .	42
2.36 Smart Proxy ([Hohpe2005], S.561) . . . . .	43
2.37 Architekturempfehlung . . . . .	44
3.1 Auswahlverfahren (vgl. [Schreiber2003], S.36) . . . . .	47
3.2 Auswahlverfahren ([Schreiber2003], S.182) . . . . .	48
3.3 Zusammenhang Anforderungsprofil - Kriterienkatalog - Bewer- tungslisten ([Schreiber2003], S.146) . . . . .	66
3.4 geforderte vs. offerierte Leistungen ([Schreiber2003], S.192) . .	73
3.5 Apache ActiveMQ - Durchsatz . . . . .	82
3.6 Apache ActiveMQ - Webinterface . . . . .	84
3.7 Sun OpenMQ- Durchsatz . . . . .	85
3.8 Sun OpenMQ Administration Interface . . . . .	87
3.9 JBoss Messaging - Administration . . . . .	88
3.10 JBoss Messaging - Fehlverhalten . . . . .	88
3.11 IBM WebsphereMQ - Administration . . . . .	91
3.12 Progress SonicMQ - Durchsatz . . . . .	95
3.13 Progress SonicMQ - Installation . . . . .	96
3.14 Progress SonicMQ - Administration . . . . .	96
3.15 Fiorano FioranoMQ - Durchsatz . . . . .	97
3.16 Fiorano FioranoMQ - Installation . . . . .	98
3.17 Fiorano FioranoMQ - Administration . . . . .	99
3.18 Nutzwertprofil - Effektive Nutzwerte in Prozent des maximalen Nutzwertes . . . . .	113
3.19 Risikoprofile . . . . .	115
3.20 Effektive Nutzwerte (kumuliert) . . . . .	117
A.1 UML der Test Applikation . . . . .	3

## Formelverzeichnis

2.1 Anzahl der Verbindungen bei Point-to-Point Architektur . . . . .	8
3.1 Bearbeitungsdauer . . . . .	51
3.2 Durchsatz . . . . .	52
3.3 Verfügbarkeit . . . . .	55

3.4	Punkte . . . . .	69
3.5	Nutzwert . . . . .	70
3.6	effektiver Nutzwert . . . . .	70
3.7	Risiko . . . . .	76
3.8	Gesamtrisiko . . . . .	76
3.9	Gesamtrisiko (prozentuell) . . . . .	77

## Tabellenverzeichnis

3.1	Marktpositionierung . . . . .	62
3.2	Marktpositionierung - Punkteeinteilung . . . . .	63
3.3	Paarvergleichsmethode . . . . .	69
3.4	Kriterienkatalog . . . . .	71
3.5	KO-Kriterienliste . . . . .	72
3.6	Template - Bewertungsliste . . . . .	74
3.7	Skalen zur Risikobewertung . . . . .	76
3.8	Template - Risikobewertung . . . . .	77
3.9	Beispiele zu Investitions- und Betriebskosten . . . . .	79
3.10	Total Costs of Ownership . . . . .	80
3.11	Middleware Lösungen / Produkte . . . . .	81
3.12	System Requirements - Apache ActiveMQ . . . . .	83
3.13	System Requirements - Sun OpenMQ . . . . .	85
3.14	System Requirements - JBoss Messaging . . . . .	88
3.15	System Requirements - IBM Websphere . . . . .	89
3.16	System Requirements - Oracle Advanced Queuing . . . . .	92
3.17	System Requirements - Progress SonicMQ . . . . .	95
3.18	System Requirements - Fiorano FioranoMQ . . . . .	98
3.19	Grobevaluierung . . . . .	99
3.20	Bewertungsliste Apache ActiveMQ . . . . .	101
3.21	Risikobewertung Apache ActiveMQ . . . . .	102
3.22	Total Costs of Ownership - Apache ActiveMQ . . . . .	103
3.23	Bewertungsliste Sun OpenMQ . . . . .	104
3.24	Risikobewertung Sun OpenMQ . . . . .	105
3.25	Total Costs of Ownership - Sun OpenMQ . . . . .	106
3.26	Bewertungsliste Progress SonicMQ . . . . .	108
3.27	Risikobewertung Progress SonicMQ . . . . .	109
3.28	Bewertungsliste Fiorano FioranoMQ . . . . .	111

3.29 Risikobewertung Fiorano FioranoMQ . . . . .	112
3.30 Gegenüberstellung . . . . .	114
3.31 Risikobewertungen . . . . .	115
A.1 Kommentare zur Bewertung von: Apache ActiveMQ . . . . .	30
A.2 Kommentare zur Bewertung von: Sun OpenMQ . . . . .	32
A.3 Kommentare zur Bewertung von: Progress SonicMQ . . . . .	34
A.4 Kommentare zur Bewertung von: Fiorano FioranoMQ . . . . .	35

## Listings-Verzeichnis

2.1 Beispiel - Point-to-Point Channel . . . . .	12
2.2 Beispiel - Publish-Subscribe Channel . . . . .	13
2.3 Beispiel - Invalid Message Channel . . . . .	14
2.4 Beispiel - Invalid Message Channel . . . . .	16
2.5 Beispiel - Message Endpoint . . . . .	18
2.6 Beispiel - Message Gateway . . . . .	19
2.7 Beispiel - Polling Consumer . . . . .	21
2.8 Beispiel - Event-Driven Consumer . . . . .	22
2.9 Beispiel - Selective Consumer . . . . .	22
2.10 Beispiel - Command Message . . . . .	24
2.11 Beispiel - Document Message . . . . .	26
2.12 Beispiel - Event Message . . . . .	28
2.13 Beispiel - Request-Reply . . . . .	29
2.14 Beispiel - Correlation Identifier . . . . .	31
2.15 Beispiel - Message Routing . . . . .	32
2.16 Beispiel - Content-based Router . . . . .	33
2.17 Beispiel - Resequencer (Pseudo Code) . . . . .	35
2.18 Beispiel - Message Translator . . . . .	37
2.19 Beispiel - Wire Tap (Pseudo Code) . . . . .	41
3.1 Error Sun Installer . . . . .	86
3.2 Installation IBM WebsphereMQ . . . . .	90
3.3 Installation/Konfiguration - Oracle AQ . . . . .	92
A.1 JMS Provider Properties . . . . .	4
A.2 JMS Abstract Message Endpoint . . . . .	5
A.3 JMS Abstract Message Producer . . . . .	8

A.4	JMS Abstract Message Listener . . . . .	10
A.5	JMS Abstract Connection Handler . . . . .	13
A.6	JMS Properties Handler . . . . .	14
A.7	JMS Message Producer . . . . .	16
A.8	JMS Message Listener . . . . .	17
A.9	Start multiple Message Listeners . . . . .	17
A.10	Start multiple Message Producers . . . . .	19
A.11	Apache ActiveMQ Connection Handler . . . . .	21
A.12	Fiorano FioranoMQ Connection Handler . . . . .	22
A.13	IBM WebsphereMQ Connection Handler . . . . .	23
A.14	JBoss Messaging Connection Handler . . . . .	24
A.15	Oracle Advanced Queuing Connection Handler . . . . .	25
A.16	Progress SonicMQ Connection Handler . . . . .	26
A.17	Sun OpenMQ Connection Handler . . . . .	27

# **Danksagung**

An dieser Stelle möchte ich mich bei jenen Personen bedanken welche maßgeblich am Zustandekommen und gelingen dieser Diplomarbeit beteiligt waren.

Dank gebührt meinem firmenseitigen Betreuer DI(FH) Dieter Zeiml und meinem Betreuer seitens der FH FH-Prof. DI Peter Salhofer für ihre Hilfestellung und konstruktive Kritik im gesamten Verlauf der Diplomarbeitserstellung.

Dank ist auch meiner Mutter auszusprechen, die mich immer unterstützt und gefördert hat.

Besonderer Dank gebührt meiner Frau, ohne sie wäre ich nicht wo ich jetzt bin.

**D a n k e**



## Kapitel 1

# Allgemeines

### Arbeitshypothese

Es existiert eine (kommerziell) verfügbare Messaging Middleware, die den spezifischen Anforderungen des Unternehmens SSI Schäfer PEEM hinsichtlich Leistungsfähigkeit, Zuverlässigkeit, Konfigurierbarkeit sowie Integrationsfähigkeit in die bestehende Systemumgebung genügt.

---

An (commercial) Messaging Middleware which fits the specific needs of the company SSI Schäfer PEEM concerning its performance, reliability, configurability and its ability to integrate into the present system environment is available.

---

## 1.1 Abstract - Deutsch

Die Diplomarbeit wurde im Auftrag und mit der Unterstützung des Unternehmens SSI Schäfer PEEM erstellt und befasst sich mit dem Thema der Integration von heterogenen Systemen. Der Fokus liegt auf den Möglichkeiten der Integration durch Messaging Systeme.

Das einführende Kapitel „Integration von Systemen“, beschäftigt sich mit den grundlegenden Möglichkeiten sowohl Daten als auch Funktionen zwischen verteilten Systemen auszutauschen. Anhand der aufgezeigten Möglichkeiten wird eine Architekturempfehlung erarbeitet welche alle benötigten Funktionalitäten eines Messagingsystems vereint ohne dieses unnötig komplex zu gestalten.

Das Kapitel „Evaluierung von Messagingsystemen“, richtet sich primär auf

die Überprüfung der Arbeitshypothese. Anhand eines strukturierten Vorgehensmodells, der so genannten Nutzwertanalyse, werden Softwareprodukte, in diesem Fall Messagingsysteme, objektiv bewertet und einander gegenübergestellt. Das Vorgehensmodell umfasst vier Abschnitte; beginnend bei der Marktanalyse über die Grob- zur Detailevaluierung, welche sowohl die Evaluierungen der einzelnen Produkte als auch die Gegenüberstellung dieser umfasst, bis hin zur Produktempfehlung.

Im Rahmen der Marktanalyse wurde eine Auswahl an am Markt verfügbaren Produkten erstellt, welche im weiteren Verlauf eingehender betrachtet werden. Die Grobevaluierung der Produkte dient dazu die grundsätzliche Eignung dh die vollständige Erfüllung aller definierten KO-Kriterien festzustellen. Jegliche Produkte welche diese Kriterien nicht erfüllen, werden in weiterer Folge aus der Produktempfehlung ausgenommen. Die Detailevaluierung widmet sich der Beantwortung der Frage zu welchem Grad die definierten Kriterien erfüllt werden. Des Weiteren werden im Rahmen der Detailevaluierung auch die Fragen nach den, mit dem jeweiligen Produkt, verbundenen Kosten und Risiken betrachtet.

Die Ergebnisse der Detailevaluierung werden gesammelt gegenübergestellt, um eine einfache Vergleichbarkeit der einzelnen Lösungen zu garantieren. Aus den Detailevaluierungen bzw der Gegenüberstellung der Produkte kann eine Produktempfehlung abgeleitet werden, welche die optimale Lösung für die definierten Anforderungen darstellt.

## 1.2 Abstract - English

This thesis was written on behalf and with the support of the company SSI Schäfer PEEM and addresses to topic of integrating heterogeneous systems. The main focus is on the advantages of the utilization of message oriented middleware.

The introductory chapter „Integration von Systemen“, deals with the basic prospects for sharing data as well as functionality across distributed systems. By the means of the possibilities shown, a recommendation for a messaging system which incorporates all needed functionality without cluttering the system, is developed.

The chapter „Evaluierung von Messagingsystemen“, focuses on answering the hypothesis. By using a structured procedure model called „Value-Benefit analysis“, software products, in this case messaging systems, are made comparable and therefore can be set in contrast with each other. The procedure model is divided into four parts, starting with an market analysis going over to rough and detailed evaluation, which deals with the evaluations of the products as well as with the comparison of them, resulting in an product recommendation.

Within the market analysis an assortment of available products was created which will be reviewed in detail. The rough evaluation of the products ensures that the products are basically suitable which means that the products must satisfy all of the KO criteria. Any product which does not meet these criteria will not be considered in the product recommendation. The detailed evaluation addresses the question to which extend the products satisfy the criteria. Furthermore the issues of costs and risks of the given products are examined.

The results of the detailed evaluation are collectively compared with each other to guarantee a straightforward comparison of the products. Based on the detailed evaluation respectively the comparison of the products a product recommendation can be derived which identifies the ideal solution for the given requirements.

## 1.3 Unternehmensvorstellung

Das weltweit agierende Unternehmen SSI Schäfer PEEM ist Spezialist in der Lager-, Förder- und Kommisioniertechnik. Das Unternehmen widmet sich der Planung, Entwicklung und Fertigung von Leistungen und Produkten für komplexe Logistiklösungen mit modernsten Technologien an einem zentralen Standort. Beginnend bei einfachen Förderstrecken bis zu komplett automatisierten Lager-, Förder- und Kommisionieranlagen.



Abbildung 1.1: SSI Schäfer Promo

Das Unternehmen ist seit dem Jahre 2001 Teil der Unternehmensgruppe SSI Schäfer, zu der unter anderem auch der Schäfer Shop und SSI Schäfer Noell zählen, insgesamt verfügt die Unternehmensgruppe über Standorte in 53 Ländern und beschäftigt mehr als 7.500 Mitarbeiter in den verschiedensten Bereichen.

Der Erfolg des Unternehmens lässt sich auch anhand der Umsatzzahlen ablesen, so stieg der Umsatz im Jahr 2006 sprunghaft auf 60 Millionen Euro, was einer Steigerung von rund 40% nahe kommt, dieser Aufwärtstrend bestimmte auch die nächsten Jahre und pendelte sich nun bei einer jährlichen Umsatzsteigerung zwischen 10 und 30% ein.

Die Anzahl der jährlich durchgeführten Projekte liegt zwischen 20 und 30, wobei es sich hier zum Teil um kleinere Anlagen handelt, welche nur ein Umsatzvolumen von einigen 100.000 Euro haben bis zu großen Anlagen, welche bis zu 20 Millionen Euro kosten. Zusätzlich ist anzumerken das im Laufe der Jahre die Größe und Komplexität der Anlagen stetig im Steigen begriffen war und sich dieser Trend mit größter Wahrscheinlichkeit in den nächsten Jahren weiter fortsetzen wird.

Ziel des Unternehmens ist es, seinen Kunden flexible und nach deren Wünschen und Anforderungen geplante und realisierte Systeme zu liefern. Um dies zu erreichen, werden in enger Zusammenarbeit mit dem Kunden deren Anforderungen im Detail analysiert und in die Planung integriert. Dies

kennzeichnet bereits eine der Qualitäten die SSI Schäfer PEEM von anderen Anbietern am Markt abhebt.

Da es sich bei solchen Anlagen um sehr komplexe Systeme handelt, bieten viele Unternehmen lediglich Fertig-Modul-Systeme an, welche sich jedoch nur in geringem Maße an die speziellen Bedürfnisse des Kunden anpassen lassen.

Sowohl durch die hauseigene Forschungs- und Entwicklungsabteilung als auch durch exzellente Kooperationen mit externen Partnern ist es möglich, komplexe Anlagen am neuesten Stand der Technik zu realisieren. Neben dem Ziel, die Wünsche des Kunden bei der Planung zu integrieren, ist es auch möglich durch den modularen Aufbau jederzeit auf geänderte Anforderungen von Kundenseite zu reagieren. Zudem lassen sich die Systeme der Firma SSI Schäfer PEEM mit allen gängigen Logistiksystemen anderer Hersteller kombinieren und integrieren.

Neben der starken Ausrichtung auf die Kundenbedürfnisse im Rahmen der Planung und Errichtung der Anlage, werden dem Kunden diverse Möglichkeiten geboten, bei Problemen oder Fragen, mit dem Unternehmen jederzeit in Kontakt zu treten. So werden zum Beispiel, eine 24-Stunden Hotline und direkte Hilfestellung durch Remotezugriffe angeboten, was eine zusätzliche Bereicherung zu den gängigen Wartungsverträgen darstellt und dem Unternehmen eine entsprechenden Wettbewerbsvorteil verschafft.

Besonders hervorzuheben ist in diesem Zusammenhang, dass das Unternehmen SSI Schäfer PEEM sowohl die Anlagendokumentation als auch die jeweiligen Schulungsunterlagen in der vom Kunden gewünschten Sprache anfertigt. Dies ermöglicht es den Kunden, durch Wegfall einer Sprachbarriere ohne großen Aufwand die eigenen Mitarbeiter einzuschulen.

## 1.4 Abgrenzung

Ziel dieser Diplomarbeit ist es, aktuelle Systeme anhand definierter Kriterien zu evaluieren und hinsichtlich ihrer Eignung bezüglich eines Einsatzes im Unternehmen zu bewerten.

Die Auswahl der zu evaluierenden Produkte wird in Zusammenarbeit mit dem Unternehmen getroffen und soll sowohl Produkte aus dem Open-Source als auch aus dem proprietären Sektor umfassen, um einen möglichst vollständigen und objektiven Blick auf die aktuelle Marktsituation zu gewährleisten.

Darüber hinaus wird eine Software-Architektur beschrieben, welche das Unternehmen dabei unterstützen soll, ein entsprechendes System selbstständig zu implementieren. Hauptaugenmerk beim Erstellen der Systemarchitektur liegt darauf, zu bedenken, dass im Unternehmen eine Vielzahl an unterschiedlichen Systemen und Programmiersprachen im Einsatz sind. Diese müssen alle ohne größeren Aufwand integriert werden können, bzw. mit dem System kommunizieren können.

Nicht zu den Zielen dieser Diplomarbeit zählen, ein vollständiges Messagingsystem zu implementieren oder ein am Markt verfügbares System in die vom Unternehmen gefertigten Anlagen zu integrieren.

## 1.5 Problembeschreibung

Die Firma SSI Schäfer PEEM baut automatisierte Lager-, Logistik- und Kommissionierungssysteme. Wobei durch die Firma sowohl die Hard- als auch Software gestellt wird. Ein solches System umfasst die automatisierte Ein- und Auslagerung von Produkten in ein Lager, im welchem die Produkte in Boxen gelagert und verwaltet werden, sowie weiters eine weitreichende Unterstützung im Bereich der Auftragsabarbeitung.

Um diese komplexen Aufgaben zu erledigen, müssen diverse Systeme miteinander interagieren und kommunizieren. Die Kommunikation der unterschiedlichen Komponenten wurde durch ein eigenes Messagingsystem (Boss) realisiert. Abbildung 1.2 zeigt schematisch den Aufbau eines solchen Lagersystems.

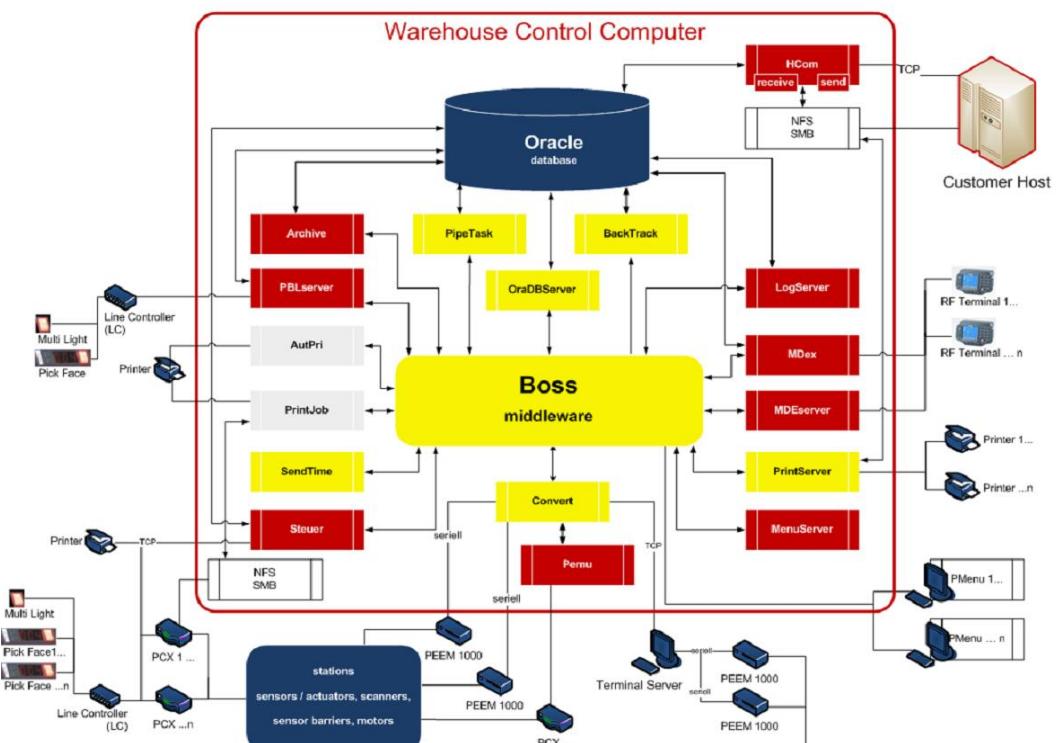


Abbildung 1.2: Systemübersicht

Sowohl während der Ein/Auslagerung als auch beim Transport werden die Boxen ständig überwacht, um den gesamten Ablauf zu optimieren und somit eine möglichst effiziente Auftragsabarbeitung zu gewährleisten.

Bei den Messages, welche in diesem System zum Einsatz kommen, handelt es sich um reine Steuerungsinformationen, welche zum Beispiel dem La-

gersystem mitteilen, dass die Box mit der ID 101010 fertig zur Auslieferung ist und dementsprechend aus dem Lagersystem ausgesteuert werden muss. Die zugehörigen Auftragsdaten sind jedoch nicht integrierter Teil der Message, diese werden vom Lagerverwaltungssystem selbstständig aus einer Datenbank bezogen.

Wie aus der Abbildung 1.2 auf Seite vii hervorgeht, stellt die Messaging Middleware einen zentralen Punkt im Gesamtsystem dar. Da bei der Entwicklung dieses Systems vor rund zehn Jahren, wichtige Punkte wie zB die Persistierung der Nachrichten vernachlässigt wurden sowie die Performance des Systems in den letzten Jahren an seine Grenzen stößt, wird nun verstärkt nach einer Ersatzlösung gesucht.

### 1.5.1 Problembereiche des Systems

Im Laufe der Tests und Durchsicht der bestehenden Dokumentation wurden die folgenden Problembereiche des aktuell eingesetzten Systems identifiziert.

- Architektur
- Algorithmus zur Abarbeitung der Queues
- Quittierung
- Handhabung von Message-Header Informationen
- Queues
- Durchsatz
- Reconnect
- Persistierung

## Architektur

Die Architektur des aktuellen Messagingsystem (Boss) orientiert sich nicht an modernen Messaging EAI Patterns (vgl. 2.2 auf Seite 7). Des Weiteren wurde die gesamte Funktionalität des Messagingssystems in einem einzelnen funktionalen Programmblöck realisiert, was dazu führt, dass dieser sehr groß und schwer zu warten wurde. Innerhalb dieses Programmblocks werden alle Aufgaben wie zB die Verwaltung der Clients, der Message-Queues sowie jegliche Fehlerbehandlung durchgeführt.

## Algorithmus zur Abarbeitung der Queues

Das System verwaltet eine Vielzahl an festen, benannten Queues sowie eine dynamische Anzahl an anonymen, nicht benannten Queues.

Diese werden mittels Round-Robin Verfahren abgearbeitet. Dies bedeutet, dass alle Queues nach der Reihe auf neue Nachrichten überprüft werden. Dies hat den gravierenden Nachteil, dass keinerlei Priorisierung einzelner Nachrichten bzw. Queues vorgenommen werden kann.

## Quittierung

Messages müssen vom Client immer quittiert werden, was bedeutet, dass der Client bestätigt, die Message vollständig erhalten zu haben.

Das zu Grunde liegende Prinzip ist durchaus sinnvoll. Es gibt im aktuellen System jedoch keinen Mechanismus, eine Quittierungsmeldung der entsprechenden Message zuzuordnen, was dazu führt, dass das System die Annahme trifft, dass jene Message quittiert wurde, welche sich am längsten in der Queue befindet. Da dies nicht zwingend der Fall sein muss, kann dies zu unvorhersehbarem (Fehl-)Verhalten des Systems führen.

Die Zuordnung der Quittierungen zu den entsprechenden Messages würde über die im Header der Messages mitgeführte Message-Nummer ermöglicht, dies wurde jedoch nicht realisiert<sup>1</sup>.

Sollten Aufgrund dessen Messages verloren gehen, so müssen die betroffenen Kommunikationspartner dies selbstständig erkennen und ein erneutes senden der Message initiieren. Wobei diese Aufgabe grundsätzlich dem Messagingsystem zuzurechnen wäre, es aber verabsäumt wurde, dies entsprechend zu berücksichtigen.

## Handhabung von Message-Header Informationen

Der Message-Header, welcher fixer Bestandteil einer jeden Message ist, beinhaltet eine Vielzahl an Daten, von denen nur ein geringer Anteil weiterverarbeitet bzw. interpretiert wird. So wird, zum Beispiel, eine eindeutige Message-Nummer mitgeführt, diese aber im weiteren Verlauf nicht berücksichtigt. Das-

<sup>1</sup>In der Dokumentation des aktuellen Systems findet sich hierzu nur der Hinweis das die im Message Header vorhandenen Daten, Message-Nummer und Message-Protokoll, nicht weiter beachtet werden

selbe gilt für die mitgeführten Informationen zum eingesetzten Protokoll. Dies kann zu einem unerwünschtem Verhalten des Systems führen, siehe hierzu die entsprechenden Ausführungen unter dem Punkt „Quittierung“ auf Seite ix.

## Queues

Die verwalteten Queues bilden eine Baumstruktur. Generell ist hier zwischen Queues mit fixem Namen und dynamischen Queues zu unterscheiden. An Parent-Queues gesendete Nachrichten werden vom System automatisiert auch an deren Child-Queues weitergereicht.

Queues mit einem fix vergebenen Namen bestehen dauerhaft, auch wenn sie gerade nicht in Verwendung sind, wohingegen dynamisch definierte Queues nur solange bestehen bleiben, wie sie auch verwendet werden. Dynamische Queues werden, nachdem der letzte Client die Verbindung trennt, gelöscht und das System gibt den belegten Speicherplatz wieder frei.

Jede Queue, unabhängig davon, ob es sich um eine fixe oder dynamische Queue handelt, kann nur eine bestimmte Anzahl an Messages aufnehmen. Wird diese überschritten, so werden vom System keine neuen Messages mehr angenommen. Diese werden umgehend verworfen, und der Client muss diese selbstständig erneut senden. Insofern kommt es zu einer Verschiebung von Aufgaben, die dem Messagingsystem zuzurechnen sind, hin zu den Clientsystemen. Weiters führt dies zu einem zusätzlichen Overhead an transferierten Nachrichten. Dieser Umstand führt zu einer starken Kopplung zwischen den Clientsystemen und dem Messagingsystem und verhindert eine klare Aufgabentrennung.

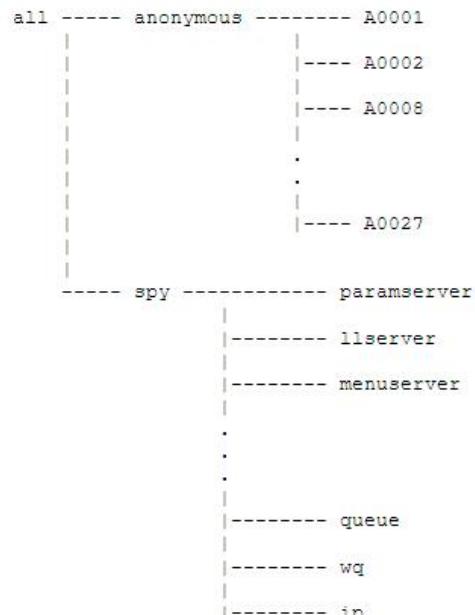


Abbildung 1.3: Boss Queues ([Manowarda1999], S.8)

## Durchsatz

Das aktuelle System unterliegt starken Schwankungen hinsichtlich des Durchsatzes, abhängig von der Größe der zu transportierenden Messages. Neben diesem Umstand, variiert auch die jeweilige Transferdauer der Messages stark.

Um das aktuelle System zu testen, wurden jeweils 10.000 Messages mit unterschiedlicher Größe an das System gesendet und die jeweiligen Zeiten zwischen Senden und Empfangen protokolliert. Bei den Tests wurde nur ein Sender und ein Empfänger verwendet. Die Auswirkungen mehrerer Clients wurde in diesem Abschnitt nicht berücksichtigt. Bei den Test befanden sich sowohl Sender als auch Empfänger am selben physikalischen Gerät, der Server wurde in einer Virtual Machine betrieben.

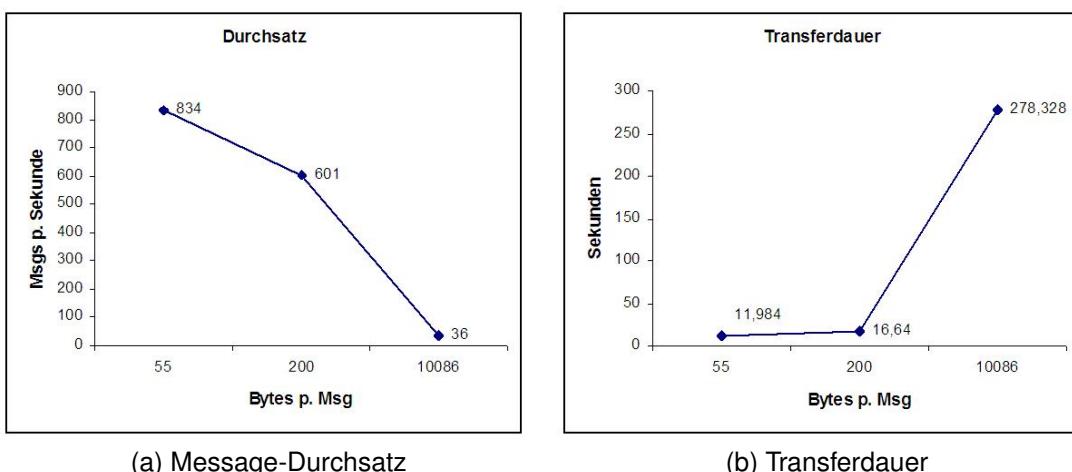


Abbildung 1.4: Evaluierung SSI Schäfer PEEM Messaging System

Abbildung 1.4a zeigt wie stark der Message Durchsatz sinkt wenn die Messagegröße auf rund 10kB ansteigt, Abbildung 1.4b zeigt deutlich, dass bei 10kB die gesamte Transferdauer<sup>2</sup> bei inakzeptablen 278,328 Sekunden<sup>3</sup> liegt.

Laut Firmeninformationen liegt der Durchschnitt der Messagegrößen bei etwa 200 Byte. In diesem Fall ist davon auszugehen das etwa 600 Messages pro Sekunde übertragen werden können.

<sup>2</sup>Zustellung aller 10.000 Messages

<sup>3</sup>dies entspricht ~ 4,7 Minuten

## Reconnect

Sollte ein Client, im Laufe einer Messageübertragung, die Verbindung zum Server verlieren, zum Beispiel durch einen Netzwerkfehler, so kann dies dazu führen, dass es dem Client nicht mehr möglich ist sich wieder zum Server zu verbinden, da die aktuelle Queue bereits belegt ist.

Dieser Zustand bleibt so lange bestehen, bis der Server erkennt, dass der Client nicht mehr verfügbar ist und die entsprechenden Aufräumarbeiten startet und die Queue somit wieder frei gibt.

## Persistierung

Das aktuelle System verfügt über keinen Mechanismus um Nachrichten zu persistieren.

Nachrichten werden so lange in den Queues behalten, bis der entsprechende Client den Empfang bestätigt. Sollte allerdings das Messagingsystem durch Fehlverhalten oder einen bewusst herbeigeführten Umstand neu gestartet werden, so gehen alle bis zu diesem Zeitpunkt nicht übertragenen Messages unweigerlich verloren.

### 1.5.2 Stärken des Systems

Das aktuelle System besitzt auch eine Reihe an Vorteilen, welche im Verlauf der Suche nach einem Ersatzsystem als Minimalanforderungen anzusehen sind.

Als Stärken wurden die folgenden Punkte identifiziert.

- Sprachunabhängigkeit
- Abhängigkeiten zu Fremdsystemen
- Wartungsfreiheit

## Sprachunabhängig

Die gesamte Kommunikation zwischen dem System und den Clients basiert auf Sockets und ist somit als sprachunabhängig<sup>4</sup> zu bezeichnen.

<sup>4</sup>dh das der Einsatz verschiedenster Programmiersprachen möglich ist

Dies bringt den Vorteil mit sich, dass von jeglichem System aus Messages gesendet und empfangen werden können. Da im Unternehmen nicht nur rein auf Java gesetzt wird, sondern auch auf eine Vielzahl weiterer Produkte, zum Beispiel Perl, ist dies als klare Stärke zu identifizieren.

## Unabhängigkeit zu Fremdsystemen

Das System weist keinerlei Abhängigkeiten zu externen Systemen oder Programmen auf und der Source-Code kann vollständig vom Unternehmen verwaltet und bearbeitet werden.

Die, unter Punkt 1.5.1 auf Seite viii, aufgezeigten Schwächen und Probleme des aktuellen Systems, resultieren in Abhängigkeiten zu internen Systemen. Die Unzulänglichkeiten des Systems werden durch die beteiligten Clientsysteme ausgeglichen. Dies stellt jedoch eine vollkommen unbefriedigende Situation dar, da es zu einer Vermischung von Aufgaben zwischen den Systemen kommt.

## Wartungsfreiheit

Da sich das System seit mehr als 10 Jahren ohne gravierende Änderungen im Produktiveinsatz befindet, lässt sich sagen, dass das System als abgeschlossen bzw. wartungsfrei zu bezeichnen ist.

# 2

## Kapitel 2

# Integration von Systemen

Im Bereich der modernen EDV Systeme ist es notwendig, verschiedenste Systeme zu integrieren und diese miteinander interagieren zu lassen.

Vielfach ist es in Unternehmen der Fall, dass es eine Vielzahl an Software-Systemen oder, genereller gesprochen, an Prozessen gibt, welche alle auf einen gemeinsamen Daten- bzw. Wissensstand zurückgreifen müssen.

Betrachtet man den einfachen Geschäftsfall einer Adressänderung durch einen Kunden, so wird es in der Regel der Fall sein, dass diese Information in mehreren unternehmensweiten Systemen benötigt wird, zum Beispiel bei der Verrechnungsstelle oder im CRM<sup>1</sup> System. Neben diesen beiden würde sich noch eine Vielzahl an weiteren betroffenen Systemen finden lassen. (vgl. [Hohpe2005], S.1ff)

Das Ziel der Integration verteilter Systeme liegt nun darin, eine Möglichkeit zu schaffen, Änderungen und Neuerfassungen von Daten unternehmensweit konsistent und in möglichst wenigen Schritten sowie weitestgehend automatisiert zu gestalten. Um dieses Ziel zu erreichen, wurden im Laufe der Zeit die verschiedensten Ansätze entwickelt, welche je nach Situation unterschiedlich geeignet sind, die Ziele des Unternehmens zu unterstützen.

In diesem Zusammenhang wird oftmals der Begriff Middleware verwendet, wobei im allgemeinen ein System bezeichnet wird, welches es unterschiedlichen Anwendungen erlaubt auf verteilte Ressourcen zuzugreifen. (vgl. [ITWISSEN2009a])

Schreiber definiert den Begriff Middleware wie folgt.

*"Eine Reihe von systemnahen Softwarekomponenten zur Kommunikation unterschiedlicher Betriebssysteme, SQL-Dialekten, Netzwerkprotokollen und zur Ergänzung von Systemsoftware zum Beispiel für Remote Procedure Calls (RPC), Transaktionsmonitoren, Load Balancing, Message-Queuing usw." ([Schreiber2003], S.306)*

<sup>1</sup>Customer Relationship Management

## 2.1 Enterprise Application Integration

Enterprise Application Integration (EAI) beschreibt unterschiedliche Möglichkeiten, komplexe Software-Systeme miteinander zu verbinden, um eine optimale Ausrichtung auf die Geschäftsprozesse zu ermöglichen.

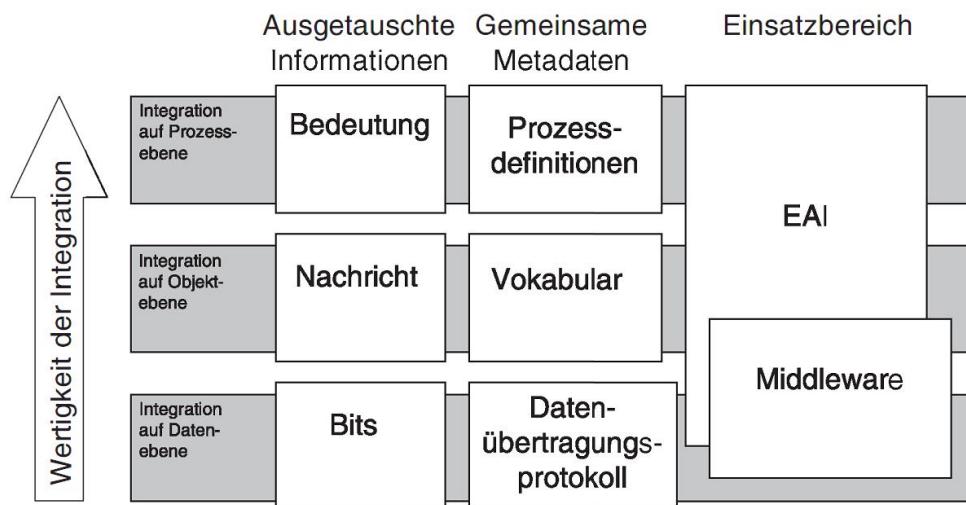


Abbildung 2.1: EAI Ebenenmodell nach Ring ([Dangelmaier2002] zit. nach K. Ring)

Wie in Abbildung 2.1 deutlich zu erkennen ist, ist die Ausrichtung bezüglich der Prozessintegration im Bereich der EAI sehr deutlich, wohingegen sich die historisch früher entwickelte Middleware auf eine Integration hinsichtlich der Objekte bzw. Daten fokussiert.

Hinsichtlich den Zielen dieser Arbeit, wird der weitere Fokus auf die Bereiche der Middleware gelegt und nur ein kurzer Überblick über die weiteren Bereiche der EAI gegeben.

Hohpe und Woolf haben die unterschiedlichsten Möglichkeiten bzw. Stile der Integration in vier große Gruppen zusammengefasst (vgl. [Hohpe2005], S.41). Diese Gruppierungen lassen sich größtenteils dem Bereich der Middleware zuordnen, finden sich aber in ähnlicher oder gleicher Form auch im umfassenderen Bereich der EAI wieder.

**File Transfer** Die Kommunikation zwischen den unterschiedlichen Systemen erfolgt auf Basis von Dateien, welche von einem Kommunikationspartner erzeugt und von anderen wiederum gelesen werden.

**Shared Database** Die Kommunikationspartner stellen Daten in einer geteilten Datenbank zur Verfügung. Dies bedeutet, dass alle Systeme, soweit notwendig, Lese- und Schreibrechte auf diese Datenbank besitzen.

**Remote Procedure Call** Die beteiligten Systeme stellen Funktionen zur Verfügung. Dies bedeutet, dass andere Systeme die bereitgestellten Funktionen aufrufen und so mit diesem interagieren können.

**Messaging** Die Kommunikationspartner tauschen Informationen und Funktionalitäten anhand von Nachrichten aus. Meist wird dazu ein Vermittler für die Nachrichten herangezogen.

## 2.1.1 File Transfer

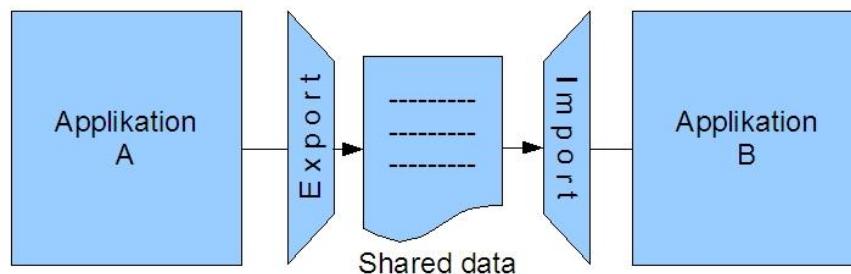


Abbildung 2.2: File Transfer (vgl. [Hohpe2005], S.44)

Erfolgt die Integration unterschiedlicher Systeme über den Austausch von Informationen durch Dateien, wie in Abbildung 2.2 dargestellt, so ergeben sich diverse Vor- als auch Nachteile.

Zu den Vorteilen gehört, dass die unterschiedlichen Systeme keinerlei Wissen über den Aufbau der anderen Systeme haben müssen. Es müssen sich lediglich alle beteiligten Systeme auf ein gemeinsames Datenformat einigen, bzw. müssen die einzelnen Systeme dafür Sorge tragen, dass die in den Dateien verspeicherten Informationen korrekt und vollständig interpretiert werden. Zu diesem Zweck kommen in den meisten aktuellen Systemen XML Dateien zum Einsatz.

Es ergeben sich aus dieser Integrationsmethode jedoch auch eine Reihe von Nachteilen. So ist zum Beispiel das Lesen und Verarbeiten, bzw. das Schreiben von Dateien ein ressourcenintensiver Vorgang. Aus diesem Grund wird der Informationsaustausch via Dateien meist nur in einem bestimmten Intervall durchgeführt, um einen zu hohen Overhead zu verhindern. Zusätzlich

muss eines der Systeme mit der Verwaltung der auszutauschenden Dateien beauftragt werden.

Ein weiterer Nachteil dieser Variante besteht darin, dass Informationen meist in einem festen Intervall zwischen den Systemen abgeglichen werden. In der Zwischenzeit kann es dazu kommen, dass unterschiedliche Systeme, unterschiedliche Informationen zum selben Objekt aufweisen. (vgl. [Hohpe2005], S.43-46)

## 2.1.2 Shared Database

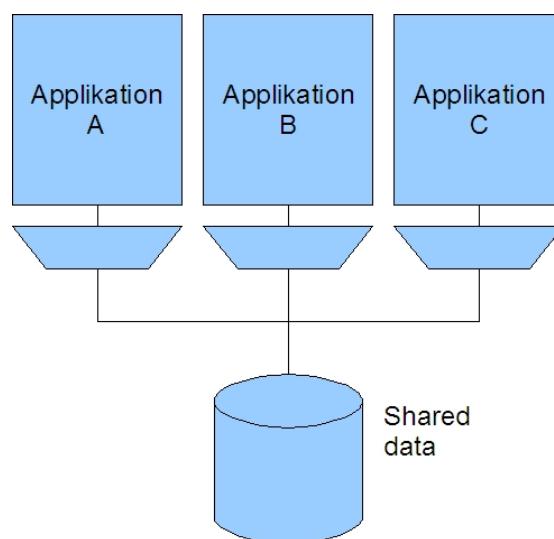


Abbildung 2.3: Shared Database (vgl. [Hohpe2005], S.48)

Der Fakt, dass beinahe 100% aller aktuellen Programmiersprachen und Entwicklungsumgebungen über die Möglichkeit verfügen, mittels Structured Query Language (SQL) mit den unterschiedlichsten Datenbanken zu kommunizieren, legt nahe, eine Form der Integration auf Ebene von Datenbanken, wie in Abbildung 2.3 dargestellt, zu realisieren. Der Vorteil dieser Integrationsvariante besteht darin, dass alle beteiligten Systeme auf ein und die selbe Datenbank zu greifen und somit sichergestellt werden kann, dass Daten in jedem Fall aktuell und konsistent sind.

Erkennbare Nachteile in einem solchen Szenario sind, zum Beispiel, der Umstand, dass ein gemeinsames Datenbankschema gefunden werden muss, welches die speziellen Anforderungen jedes einzelnen Systems in Bedacht nimmt. Dies lässt sich oftmals nur sehr umständlich realisieren bzw. resultiert

in sehr komplexen Schemata. Vielfach ist es aber auch gänzlich unmöglich ein gemeinsames Schema zu finden.

Ein weiterer Nachteil liegt darin, dass eine Applikation durch die Sperrung von Datensätzen andere Systeme in deren Arbeit beeinträchtigen kann. Vergleichbar mit dem bei *File Transfer* auf Seite 3 beschriebenen Intervall, in welchem die Daten synchronisiert werden, so liegt es auch hier in der Verantwortung der jeweiligen Clientsysteme zu überprüfen, ob neue bzw. aktualisierte Daten vorhanden sind. (vgl. [Hohpe2005], S. 47-49)

### 2.1.3 Remote Procedure Call

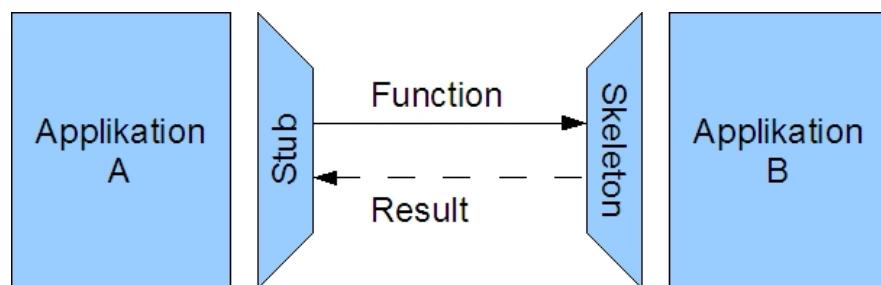


Abbildung 2.4: Remote Procedure Call (vgl. [Hohpe2005], S.51)

Die zuvor beschriebenen Integrationsvarianten *File Transfer* ( 2.1.1 auf Seite 3) und *Shared Database* ( 2.1.2 auf der vorherigen Seite) fokussieren sich auf den Austausch von Informationen zwischen den beteiligten Systemen. Der in Abbildung 2.4 dargestellte Ansatz von Remote Procedure Call (RPC), auch oftmals als Remote Procedure Invocation bezeichnet, basiert nicht mehr rein auf dem Austausch von Informationen, sondern darauf, dass jedes der Systeme Funktionen bereitstellt, welche für andere Systeme relevante Aufgaben erledigen.

Funktionen, die auf solche Weise anderen Systemen zur Verfügung gestellt werden, lassen sich analog zu lokalen Funktionen verwenden. Es muss jedoch darauf hingewiesen werden, dass der Aufruf einer Funktion, welche durch ein anderes System bereitgestellt wird, ein Vielfaches an Zeit in Anspruch nehmen kann als der Aufruf einer lokalen Funktion.

Als aktuelles Beispiel für RPC können Web Services genannt werden. Web Services bieten ihre Funktionalität über das Internet an, so lassen sich Funktionen von externen Organisationen einbinden. Ein Nachteil besteht jedoch

darin, dass man keinerlei Einfluss darauf hat, wie diese Funktionen die Daten bearbeiten, noch wie sich die Gestaltung der Funktion im Laufe der Zeit ändert.

Das größte Manko im Zusammenhang mit RPC liegt darin, dass die Verfügbarkeit der jeweiligen Funktionen nicht zu jedem Zeitpunkt garantiert werden kann. Genauso wenig lässt sich die Zeitspanne zwischen dem Aufrufen der Funktion und dem Erhalt einer Antwort abschätzen. (vgl. [Hohpe2005], S.50-52)

## 2.1.4 Messaging

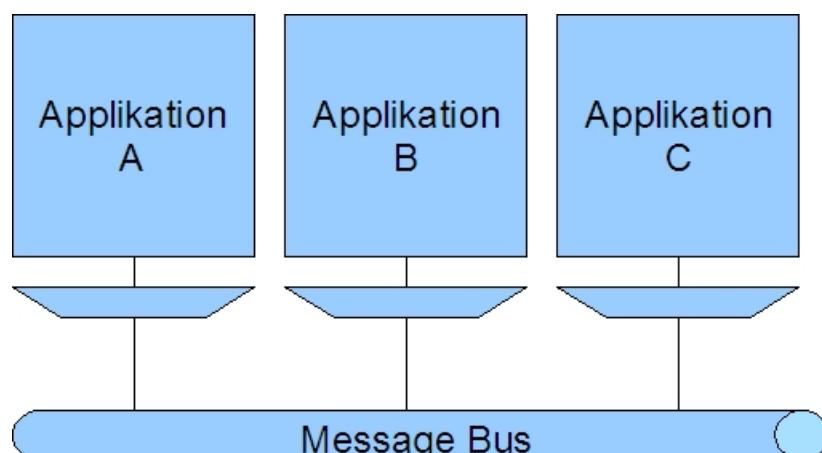


Abbildung 2.5: Messaging (vgl. [Hohpe2005], S.54)

Abbildung 2.5 zeigt den grundsätzlichen Aufbau einer Integration durch Messaging, auch als Message-oriented Middleware (MOM) bezeichnet. Messaging folgt einem ähnlichen Grundprinzip wie *File Transfer* (siehe 2.1.1 auf Seite 3), jedoch sind die auszutauschenden Datenmenge wesentlich kleiner und der Austausch der Daten erfolgt wesentlich schneller. Dh., es werden innerhalb einer Sekunde eine Vielzahl an Messages ausgetauscht.

Messaging liegt ein asynchrones Prinzip zugrunde, dh., dass Sender und Empfänger einer Nachricht nicht zur selben Zeit verfügbar sein müssen. Des Weiteren bedeutet es, dass der Sender nicht auf die Antwort des Empfängers warten muss und somit an seinen anstehenden Tätigkeiten weiterarbeiten kann.

Durch Messaging können sowohl Daten, welche in der Nachricht gekapselt sind, als auch Funktionen ausgetauscht werden. Im Gegensatz zu *RPC*

(siehe 2.1.3 auf Seite 5) besteht jedoch keine enge Kopplung zwischen den Systemen. (vgl. [Hohpe2005], S.53-56)

In einer Vielzahl der Fälle sind MOM Systeme als so genannte Message Queuing Middleware (MQM) realisiert. (vgl. [Middleware2009])

MQM Systeme verwenden so genannte Queues, um die zu transferierenden Nachrichten bis zu deren Übermittlung persistent zu halten. Das bedeutet, dass keine der Nachrichten im Laufe der Übermittlung zwischen den Kommunikationspartnern verloren gehen kann. (vgl. [Microsoft2009])

Als Mechanismus für die Verteilung von Nachrichten durch Messagingsysteme werden in der Literatur zwei grundlegende Ansätze beschrieben, zum einen Publish/Subscribe und zum anderen Point-to-Point. Die Entscheidung welcher Mechanismus zum Einsatz kommt, hängt davon ab, ob die Nachrichten nur zu einem, oder zu einer Gruppe von Empfängern übertragen werden müssen. In diesem Sinne stellt Publish/Subscribe eine Möglichkeit dar, One-To-Many Szenarien zu realisieren und Point-To-Point bildet One-To-One Szenarien ab. (vgl. [Reussner2009], S.101)

## 2.2 Messaging-Architektur (EAI Messaging Patterns)

*"Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zu einander und zur Umgebung sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen." ([Reussner2009], S.1)*

Im folgenden sollen die wichtigsten Patterns für die Erstellung eines Messagingsystems beschrieben werden. Es werden nur solche Patterns beschrieben, welche für das Unternehmen aktuell relevant sind. Um weitere Informationen zu Messaging-Patterns zu erhalten, wird auf das Buch Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions von Hohpe, Woolf verwiesen ([Hohpe2005]).

## 2.2.1 Grundfunktion und Architektur eines Messagingsystems

Die Aufgabe eines Messagingsystems besteht darin, den Austausch von Nachrichten zwischen den unterschiedlichsten Applikationen zu ermöglichen. Generell können zwei verschiedene Architekturen unterschieden werden. Zum einen die Point-to-Point Architektur (vgl. Abbildung 2.6a) und zum anderen die Hub-and-Spoke Architektur (vgl. Abbildung 2.6b). (vgl. [Hohpe2005], S.322-326)

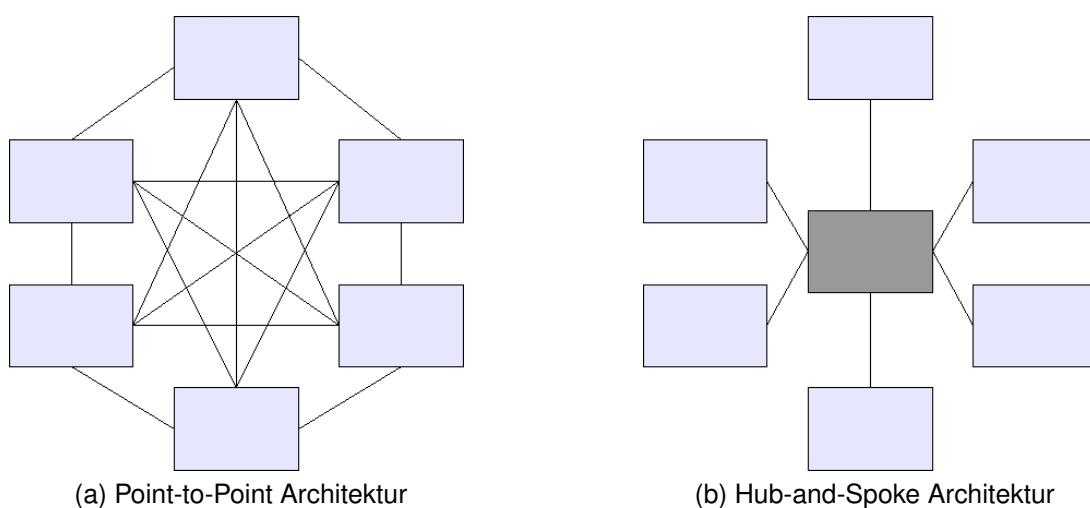


Abbildung 2.6: Architekturen von Messagingsystemen (vgl. [Hohpe2005], S.322, 324)

Im Falle der Point-to-Point Architektur wird jeweils zwischen den beteiligten Kommunikationspartnern eine direkte Verbindung aufgebaut. Der Nachteil dieses Systems besteht darin, dass es zu einer Vielzahl an Verbindungen kommt und deren Anzahl exponentiell ansteigt. Die Anzahl der benötigten Verbindungen in einer Point-to-Point Architektur lässt sich nach Formel 2.1 berechnen, wobei  $n$  die Anzahl der Endpunkte dh Applikationen definiert.

### Formel 2.1: Anzahl der Verbindungen bei Point-to-Point Architektur

$$\text{Anzahl der Verbindungen} = \frac{n * (n - 1)}{2}$$

Bei der Hub-and-Spoke Architektur übernimmt ein zentrales System (Hub) die

Verteilung der Nachrichten, somit ist es nur noch nötig, Verbindungen zwischen den Clients und dem Hub aufzubauen. Damit entspricht die Anzahl der Clients der Anzahl der benötigten Verbindungen. In diesem Sinne steigt die Anzahl der Verbindungen linear zur Anzahl der Clients.

Abbildung 2.7 verdeutlicht den Zusammenhang zwischen der Anzahl der Clients und der Anzahl der benötigten Verbindungen in den beiden genannten Architekturen.

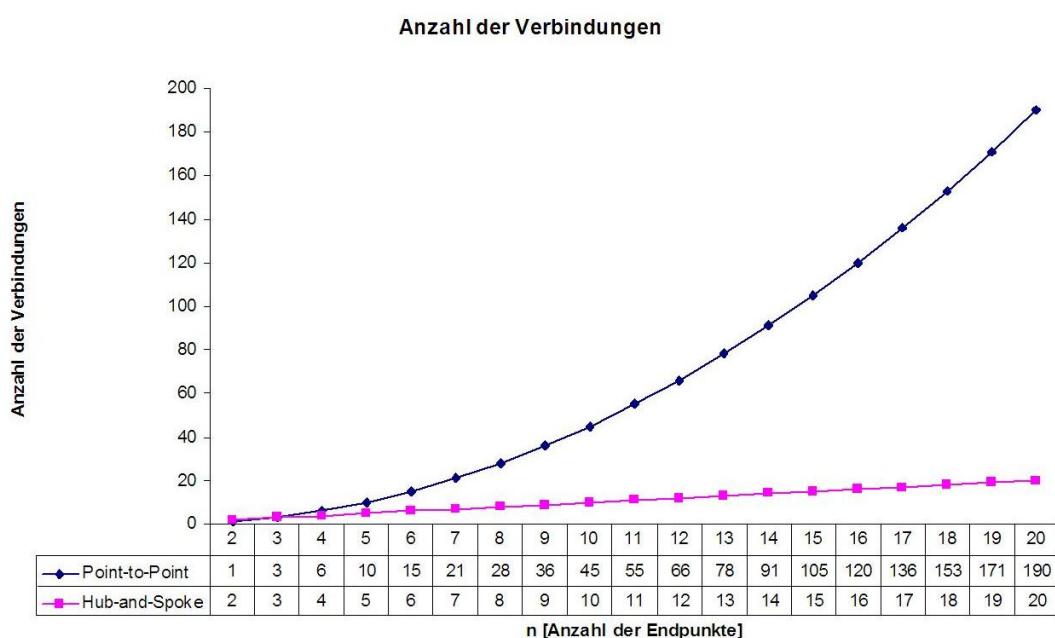


Abbildung 2.7: Benötigte Verbindungen: Point-to-Point vs. Hub-and-Spoke

Hohpe und Woolf ([Hohpe2005]) beschreiben eine Vielzahl an Patterns, die in einem Messagingsystem zum Einsatz kommen können und sollten. Diese wurden in sechs große Gruppen eingeteilt, welche in Abbildung 2.8 auf der nächsten Seite dargestellt werden.

1. Messaging Channels
2. Messaging Endpoints
3. Message Construction
4. Message Routing
5. Message Transformation
6. System Management

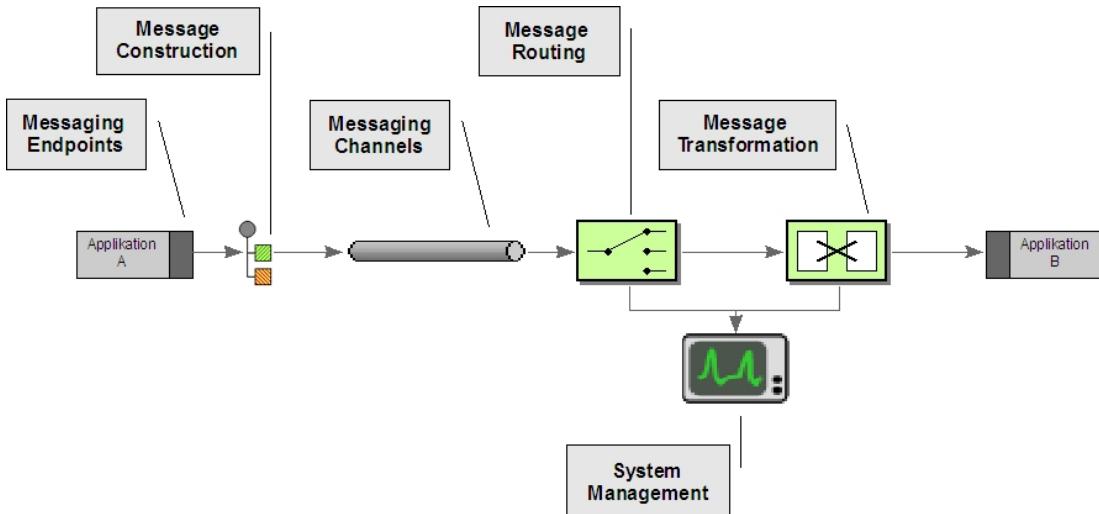


Abbildung 2.8: EAI-Patterns Overview (vgl. [Hohpe2005], Buchdeckel hinten)

Im Weiteren sollen die wichtigsten Patterns der jeweiligen Gruppe beschrieben werden, welche benötigt werden, um ein vollwertiges Messagingsystem gemäß den Anforderungen des Unternehmens zu entwickeln.

## 2.2.2 Message Channels

Message Channels (siehe Abbildung 2.9 auf der nächsten Seite) dienen dazu, die Kommunikation zwischen verschiedene Applikationen zu ermöglichen.

Nachrichten werden über Message Channels zwischen den unterschiedlichsten Applikationen ausgetauscht. Um ein effizientes System zu schaffen, werden nicht alle Nachrichten über ein und den selben Channel übertragen, sondern es gibt für unterschiedliche Anforderungen unterschiedliche Channels. Grundsätzlich lassen sich zwei große Gruppen von Message Channels unterscheiden, zum einen *Point-to-Point Channels* (siehe Seite 11) welche eine One-to-One Kommunikation ermöglichen, und zum anderen *Publish-Subscribe Channels* (siehe Seite 12) welche eine One-to-Many Kommunikation ermöglichen. (vgl. [Hohpe2005], S.60ff, 99ff)

Bei Message Channels handelt es sich um eine unidirektionale Variante, wie zwei oder mehrere Applikationen Daten untereinander austauschen können. Dies ist keine technische Gegebenheit, sondern ergibt sich als logische Konsequenz. Würden Applikationen bzw. Messaging Endpoints (siehe hierzu Punkt 2.2.3 auf Seite 17) ein und den selben Message Channel verwenden um Nachrichten zu senden und zu empfangen, so würde jede Applikation die eigenen Nachrichten empfangen.

Wird zwischen Applikationen eine Two-way Kommunikation benötigt so müssen zwei gesonderte Message Channels für das Übertragen der Anfrage (Request) und der Antwort (Reply) verwendet werden (siehe *Request-Reply* auf Seite 28). (vgl. [Hohpe2005], S.100)

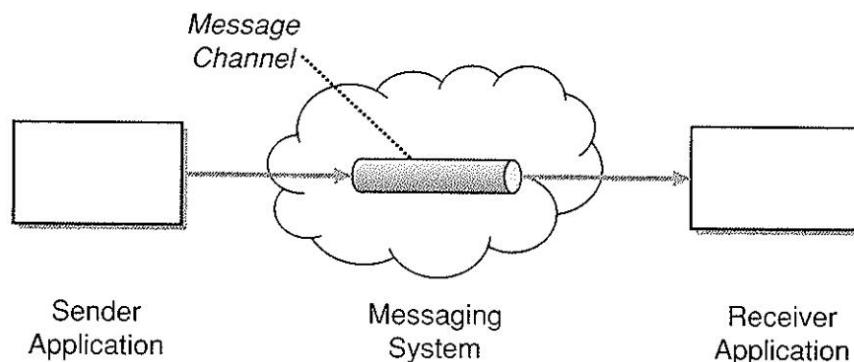


Abbildung 2.9: Message Channel ([Hohpe2005], S.61)

## Point-to-Point Channel

Point-to-Point Channels (siehe Abbildung 2.10) ermöglichen es, Daten von einer Applikation zu exakt einer anderen Applikation zu transferieren. Prinzipiell können mehrere Applikationen auf ein und demselben Point-to-Point Channel Nachrichten empfangen, es wird in diesem Fall durch den Message Channel sicher gestellt, dass nur einer der möglichen Empfänger die Nachricht bekommt. Sollte gewünscht sein, dass alle Empfänger die Nachricht erhalten, so muss ein *Publish-Subscribe Channel* (siehe Seite 12) verwendet werden. (vgl. [Hohpe2005], S.103-105)

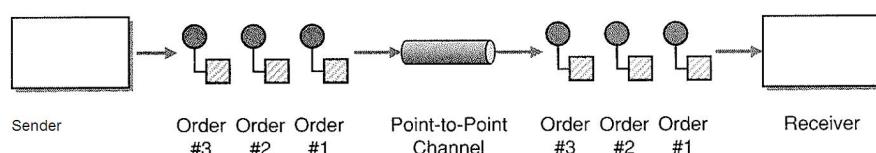


Abbildung 2.10: Point-to-Point Channel ([Hohpe2005], S.103)

In JMS werden Point-to-Point Channels als Queues bezeichnet. Listing 2.1 auf der nächsten Seite zeigt wie eine Queue zum Senden und Empfangen von Nachrichten verwendet werden kann.

**Listing 2.1: Beispiel - Point-to-Point Channel**

```
1  /** Example showing how to use Queues */
2
3  /** Initialize */
4  Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
5  connection.start();
6  Destination destination = session.createQueue(myQueue); //create
    → a Destinantion Object which matches a predefined Message
    → queue
7
8  /** MessageProducer - used for sending messages to a destination
    → */
9  producer = session.createProducer(destination);
10 TextMessage msg = session.createTextMessage();
11 msg.setText(status);
12 producer.send(msg);
13
14 /** MessageConsumer - used for receiving messages from a
    → destination */
15 MessageConsumer consumer = session.createConsumer(destination);
16 Message recvMsg = consumer.receive(50);
```

## Publish-Subscribe Channel

Publish-Subscribe Channels (siehe Abbildung 2.11 auf der nächsten Seite), bieten die Möglichkeit ein One-to-Many Szenario zu realisieren. Nachrichten, welche an einen Publish-Subscribe Channel versendet werden, werden an alle auf diesen Channel empfangenden Applikationen verteilt. Das bedeutet, dass jede Applikation, die diese Nachricht empfangen möchte, eine 1:1 Kopie dieser Nachricht erhält. (vgl. [Hohpe2005], S.106-110)

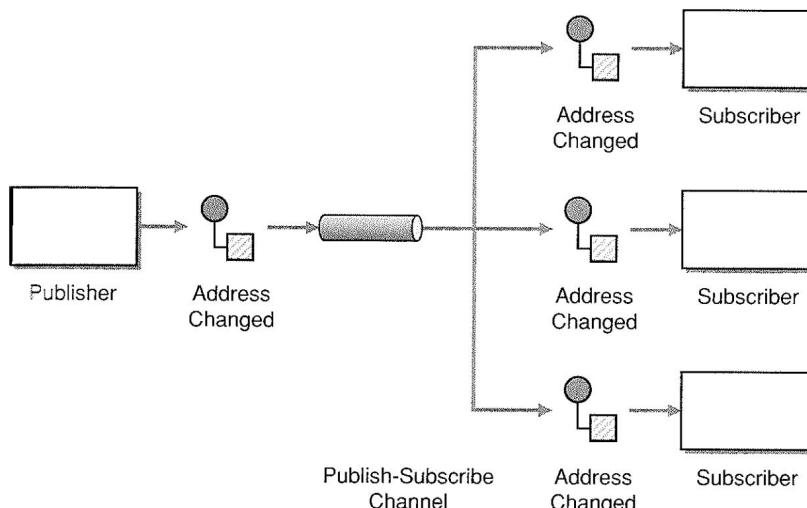


Abbildung 2.11: Publish-Subscribe Channel ([Hohpe2005], S.107)

Meist werden über einen Publish-Subscribe Channel *Event Messages* (siehe Seite 27) versendet, welche alle betroffenen Applikationen über ein Ereignis informieren. Über Publish-Subscribe Channels lassen sich zwischen null und n Applikationen über auftretende Ereignisse informieren, im Gegensatz zu Point-to-Point, wo jeweils nur ein Empfänger die Nachricht bekommen kann.

In JMS werden Publish-Subscribe Channels als Topics bezeichnet. Listing 2.2 zeigt wie diese verwendet werden können.

#### Listing 2.2: Beispiel - Publish-Subscribe Channel

```

1  /** Example showing how to use Queues */
2
3  /** Initialize */
4  Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
5  connection.start();
6  Destination destination = session.createTopic(myTopic); //create
   -a Destinantion Object which matches a predefined Message
   -queue
7
8  /** MessageProducer - used for sending messages to a destination
   - */
9  producer = session.createProducer(destination);
10 TextMessage msg = session.createTextMessage();
11 msg.setText(status);
12 producer.send(msg);
13
14 /** MessageConsumer - used for receiving messages from a
   -destination */
15 MessageConsumer consumer = session.createConsumer(destination);
16 Message recvMsg = consumer.receive(50);

```

## Invalid Message Channel

Der Invalid Message Channel (siehe Abbildung 2.12) stellt einen *Point-to-Point Channel* (siehe Seite 11) mit besonderer Bedeutung dar. Er ist dazu gedacht, fehlerhafte Nachrichten aufzunehmen. Als Empfänger eines Invalid Message Channels sollte eine Applikation stehen, welche diese Nachrichten bearbeitet und analysiert.

Ein Invalid Message Channel ist nicht als normaler Kommunikationskanal gedacht und sollte unter keinen Umständen als solcher verwendet werden, auch wenn dies technisch möglich wäre. (vgl. [Hohpe2005], S.115-118)

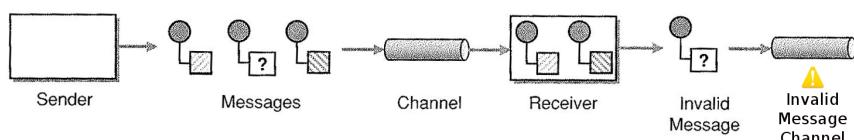


Abbildung 2.12: Invalid Message Channel ([Hohpe2005], S.116)

Sollte ein Empfänger eine Nachricht nicht validieren oder interpretieren können, so sollte diese Nachricht an einen Invalid Message Channel weitergereicht werden und der Sender der Nachricht darüber informiert werden.

Listing 2.3 zeigt anhand von JMS wie ein Empfänger eine fehlerhafte Nachricht an einen definierten Invalid Message Channel weiterreichen kann.

### Listing 2.3: Beispiel - Invalid Message Channel

```
1  /** Example showing how to use Invalid Message Channels */
2
3  /** initialize Channels */
4  Destination requestQueue = session.createQueue("requestChannel") \
5      ;
6  Destination invalidMessagesQueue = session.createQueue(" \
7      -InvalidMessageChannel");
8
9
10 /** initialize consumer */
11 MessageConsumer consumer = session.createConsumer(requestQueue);
12
13 /** initialize producer - to reroute invalid messages */
14 MessageProducer rerouteInvalidMessages = session.createProducer( \
15     invalidMessagesQueue);
16
17 /** receive message from requestQueue */
18 Message request = consumer.receive(50);
19
20 /** validate message - check if it is a JMS TextMessage if it is \
21     not the message will be rerouted to the Invalid Message \
22     Channel */
```

```
17 if (request instanceof TextMessage) {  
18     /** request message is valid */  
19     /** perform an action */  
20 } else {  
21     rerouteInvalidMessages.send(request);  
22 }
```

## Dead Letter Channel

Dead Letter Channels (siehe Abbildung 2.13) stellen, ähnlich wie Invalid Message Channels, einen *Point-to-Point Channel* (siehe Seite 11) mit besonderer Bedeutung dar. Im Gegensatz zu Invalid Message Channels, werden in Dead Letter Channels jedoch nicht invalide Messages aufgefangen, sondern Messages, welche durch das Messagingsystem nicht erfolgreich zugestellt werden konnten.

Dead Letter Channels werden nicht für die normale Kommunikation zwischen Applikationen herangezogen und sollten generell nicht in *Messaging Endpoints* (siehe Seite 17) verwendet werden. (vgl. [Hohpe2005], S.119-121)

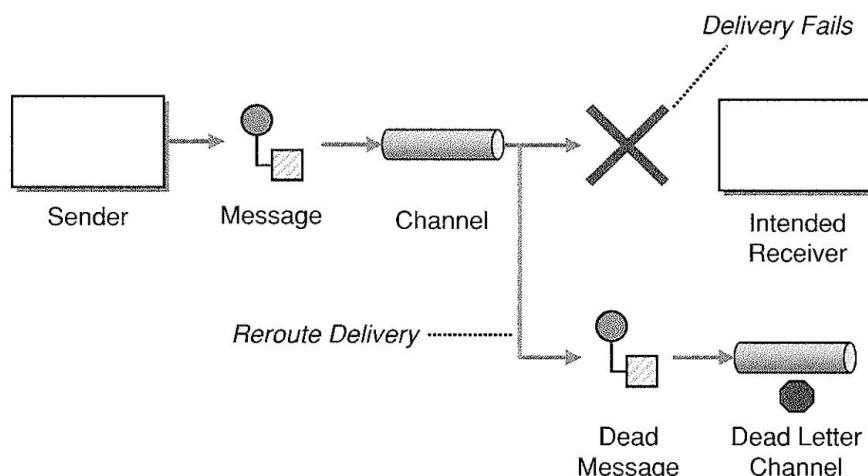


Abbildung 2.13: Point-to-Point Channel ([Hohpe2005], S.120)

Dead Letter Channels dienen ausschließlich dem Messagingsystem, um nicht zu stellbare Nachrichten in einem gesonderten Channel zu halten. Nachrichten sollten mit einer Time-To-Live oder einer anderen Form der Kennzeichnung versehen werden, welche beschreibt, in welchem Zeitraum die Nachricht von Relevanz ist und zugestellt werden soll.

## Guaranteed Delivery

Bei Guaranteed Delivery (siehe Abbildung 2.14), werden Nachrichten in einem nicht flüchtigen Speichermedium (zB HDD) verspeichert, dies kann auch als persistentes Messaging bezeichnet werden. Durch die zusätzliche Aufgabe die Nachrichten zu verspeichern, wird die Performance des Messagingsystem negativ beeinflusst.

Durch den Einsatz von Guaranteed Delivery können Nachrichten, bei einem Ausfall des Messagingsystems, wieder hergestellt und zugestellt werden. (vgl. [Hohpe2005], S122-126)

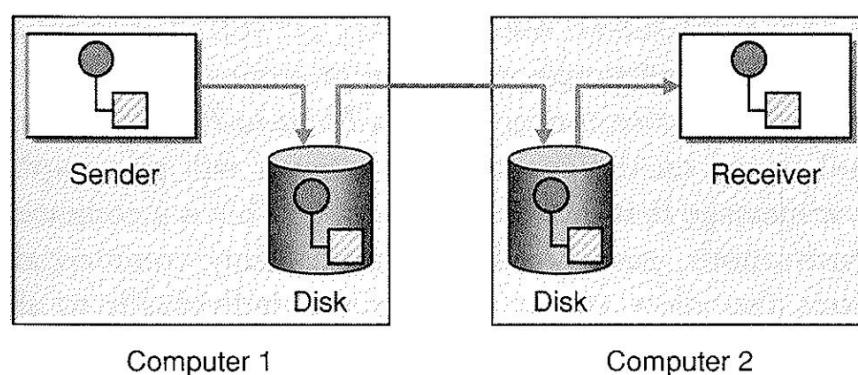


Abbildung 2.14: Point-to-Point Channel ([Hohpe2005], S.123)

JMS Nachrichten werden standardmäßig persistent versendet. Des Weiteren können entweder einzelne Nachrichten persistent versendet werden, oder der Sender (MessageProducer) an sich als persistent definiert werden, wodurch jegliche Nachricht dieses Senders persistent versendet wird. In Listing 2.4 werden diese Möglichkeiten dargestellt.

### Listing 2.4: Beispiel - Invalid Message Channel

```
1  /** Example showing how to Guaranteed Delivery can be used */
2
3  /** initialize Channels */
4  Destination destination = session.createQueue("myQueue");
5
6  /** initialize producer */
7  MessageProducer producer = session.createProducer(destination);
8  Message msg = session.createTextMessage();
9
10 /** set persistence property at message level */
11 producer.send(msg, javax.jms.DeliveryMode.PERSISTENT, javax.jms.\
12     -Message.DEFAULT_PRIORITY, javax.jms.Message.\
13     -DEFAULT_TIME_TO_LIVE);
```

12

```
13  /** set persistence property at producer level - this is the <  
   -default behaviour*>/  
14 producer.setDeliveryMode(javax.jms.DeliveryMode.PERSISTENT);  
15 producer.send(msg)
```

## 2.2.3 Messaging Endpoints

Ein Message Endpoint (siehe Abbildung 2.15) dient dazu, eine beliebige Applikation, durch Nutzung einer durch das Messagingsystem bereitgestellten API, an das Messagingsystem anzubinden. Durch Messaging Endpoints wird es einer Applikation ermöglicht Messages zu senden und zu empfangen. Für das Senden und das Empfangen wird jeweils ein eigener Messaging Endpoint benötigt.

*„The Message Endpoint encapsulates the messaging system from the rest of the application and customizes a general messaging API for a specific application and task.“ ([Hohpe2005], S.96)*

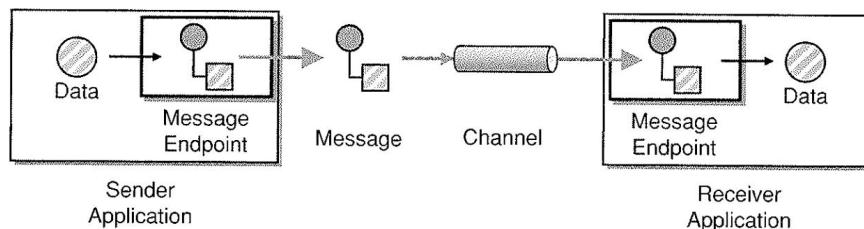


Abbildung 2.15: Messaging Endpoints ([Hohpe2005], S.96)

Grundsätzlich muss zwischen den Aufgaben einer Applikation und der Nutzung eines Messagingsystems zur Erfüllung derer differenziert werden. Diese beiden Sphären einer Applikation sollten unter keinen Umständen vermischt werden.

Messaging Endpoints können in den unterschiedlichsten Varianten gestaltet werden, so zum Beispiel als Messaging Gateway, Polling Consumer, Event-Driven Consumer oder Selective Consumer. Diese Varianten dienen alle dazu, Messages zu versenden oder zu empfangen, fokussieren sich jedoch auf unterschiedliche Problemstellungen. Das Versenden von Messages geschieht immer auf die selbe Art und Weise, in dem die Message an das Messagingsystem versandt wird. Unterschiedliche Varianten sind jedoch beim Empfangen von Messages denkbar und notwendig.

Listing 2.5 auf der nächsten Seite zeigt welche Messaging Endpoint in JMS verfügbar sind.

#### Listing 2.5: Beispiel - Message Endpoint

```
1  /** JMS Messaging Endpoints */
2
3 /**
4  * MessageProducer and MessageConsumer can work with Queues
5  * and Topics because they both represent a Destination.
6  * There are Queue and Topic specific classes available,
7  * but since JMS1.1 it is recommended to use
8  * MessageProducer and MessageConsumer.
9  * Queue(s): QueueSender and QueueReceiver
10 * Topic(s): TopicPublisher and TopicSubscriber
11 */
12
13 /**
14 Session session = connection.createSession(false, Session.＼
15     →AUTO_ACKNOWLEDGE);
16 connection.start();
17
18 /**
19  * For implementing a Point-to-Point Scenario use - JMS Queues \
20  * →*/
21 Destination destination = session.createQueue(myQueue);
22
23 /**
24  * MessageProducer - used for sending messages to a destination＼
25  * → */
26 producer = session.createProducer(destination);
27 TextMessage msg = session.createTextMessage();
28 msg.setText(status);
29 producer.send(msg);
30
31 /**
32  * MessageConsumer - used for receiving messages from a \
33  * →destination */
34 MessageConsumer consumer = session.createConsumer(destination);
35 Message recvMsg = consumer.receive(50);
36 if(recvMsg instanceof TextMessage){
37     TextMessage tmpTxtMsg = (TextMessage) recvMsg;
38     /**
39      * perform an action */
40 }
41
42 /**
43  * TopicPublisher and TopicSubscriber
44  * → */
45 TopicPublisher publisher = session.createTopic(topic);
46 TopicSubscriber subscriber = session.createSubscriber(topic);
47
48 /**
49  * TopicPublisher - used for publishing messages to a Topic \
50  * →Topic */
51 publisher.publish(message);
52
53 /**
54  * TopicSubscriber - used for receiving messages from a Topic \
55  * →Topic */
56 subscriber.receive(50);
57
58 /**
59  * QueueSender and QueueReceiver
60  * → */
61 QueueSender sender = session.createSender(queue);
62 QueueReceiver receiver = session.createReceiver(queue);
63
64 /**
65  * QueueSender - used for sending messages to a Queue \
66  * →Queue */
67 sender.send(message);
68
69 /**
70  * QueueReceiver - used for receiving messages from a Queue \
71  * →Queue */
72 receiver.receive(50);
```

## Messaging Gateway

Ein Messaging Gateway (siehe Abbildung 2.16 auf der nächsten Seite) dient dazu den Messaging Endpoint relevanten Code in einer eigenen Klasse zu

kapseln und der Applikation die benötigten Methoden zur Verfügung zu stellen.  
 (vgl. [Hohpe2005], S.469)

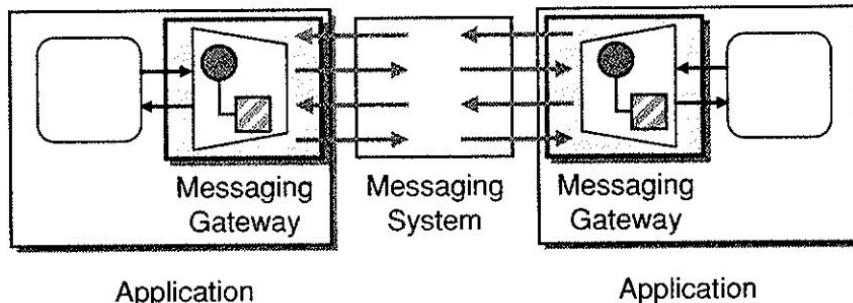


Abbildung 2.16: Messaging Gateway ([Hohpe2005], S.469)

Die Trennung zwischen Messaging Endpoint relevantem Code und der restlichen Applikation bringt den Vorteil, dass bei einer Änderung der Messaging API, sei es durch ein Update oder durch den Wechsel zu einem anderen Messagingsystem Anbieter, dieser einfacher anzupassen ist, da der Code nicht über die gesamte Applikation verstreut ist.

Messaging Gateways eignen sich sowohl zum Senden als auch zum Empfangen von Nachrichten. Message Gateways können entweder synchron oder asynchron gestaltet werden. Listing 2.6 zeigt die Grundstruktur eines synchronen Messaging Gateways und wie dieses verwendet werden kann.

#### Listing 2.6: Beispiel - Message Gateway

```

1  /** How to create and use a synchronous Messaging Gateway in JMS */
→ */
2
3  /** Message Gateway */
4  public class MsgGateway {
5
6      public boolean updateStatus(boolean status) {
7
8          /** initialization */
9          Session session = connection.createSession(false, Session.→AUTO_ACKNOWLEDGE);
10         connection.start();
11         Destination destination = session.createQueue(myQueue);
12         producer = session.createProducer(destination);
13
14         /** create and send TextMessage */
15         TextMessage msg = session.createTextMessage();
16         msg.setText(status);
17         producer.send(msg);
18
19         /**

```

```
20      * Wait for response from the messaging system, which sends a \n
21          →TextMessage containing 'true'==success or 'false'==fail\n
22  */\n23  MessageConsumer consumer = session.createConsumer(destination) \n
24      →;\n25  Message recvMsg = consumer.receive(50);\n26  TextMessage tmpTxtMsg = (TextMessage) recvMsg;\n27  return tmpTxtMsg.getText();\n28 }\n29 }\n30\n31 public class Application{\n32     MsgGateway msgGateway = new MsgGateway();\n33\n34     boolean tmp = msgGateway.updateStatus(false);\n35     if(tmp == TRUE){\n36         /** Status update was successful */\n37     }else{\n38         /** Status update failed */\n39     }\n40 }
```

## Polling Consumer

Der Polling Consumer (siehe Abbildung 2.17) stellt eine synchrone Variante eines Message Empfängers dar. In diesem Fall wird in einem festgelegten Intervall durch den Polling Consumer am Messagingsystem nachgefragt, ob neue Messages vorhanden sind.

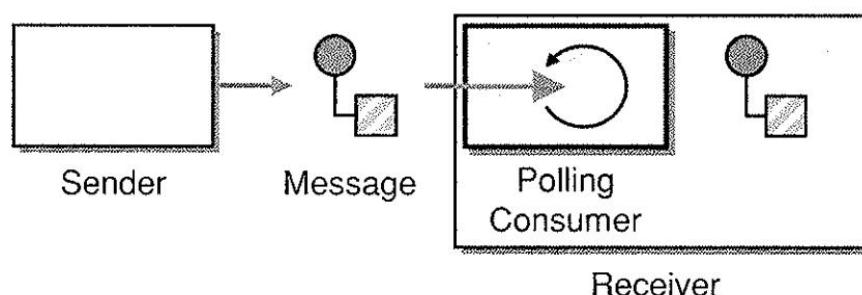


Abbildung 2.17: Polling Consumer ([Hohpe2005], S.494)

Durch dieses Verhalten kann verhindert werden, dass der Consumer schneller Messages zugestellt bekommt als er diese verarbeiten kann. Der Nachteil dieses synchronen Abfragens liegt darin, dass der Consumer in der Zwischenzeit keinerlei andere Funktionen ausführen kann.

In diesem Zusammenhang sind zwei Extremfälle zu bedenken. Zum einen, dass durch ein zu langsames Polling Intervall Messages am Messagingsystem aufgestaut werden könnten, und zum anderen, dass sollten keine Messages für diesen Consumer einlagen, dieser keine weiteren Bearbeitungen durchführen kann. (vgl. [Hohpe2005], S.494ff)

In Listing 2.7 wird dargestellt wie ein Polling Consumer in JMS realisiert werden kann.

#### Listing 2.7: Beispiel - Polling Consumer

```
1  /** Example for a JMS Polling Consumer */
2
3  MessageConsumer consumer = session.createConsumer(destination);
4
5  /** The consumer checks the destination every 50ms for new \
6   * -messages */
7  Message recvMsg = consumer.receive(50);
8  if(recvMsg instanceof TextMessage) {
9      TextMessage tmpTxtMsg = (TextMessage) recvMsg;
10     /** perform an action */
11 }
```

## Event-Driven Consumer

Im Gegensatz zu einem Polling Consumer, wird durch den Event-Driven Consumer (siehe Abbildung 2.18) nicht aktiv am Messagingsystem nach neuen Messages gefragt. Langt am Messagingsystem eine Message für diesen Consumer ein, so wird dieser durch das Messagingsystem darüber informiert.

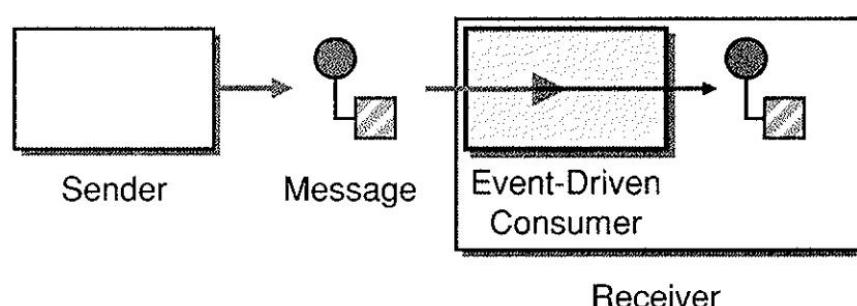


Abbildung 2.18: Event-driven Consumer ([Hohpe2005], S.498)

Dies bringt den Vorteil, dass es sich um ein asynchrones Verhalten handelt und der Consumer somit in der Zwischenzeit andere Aufgaben bearbeiten kann. (vgl. [Hohpe2005], S.498ff)

Listing 2.8 auf der nächsten Seite zeigt wie ein Event-Driven Consumer in JMS realisiert werden kann.

**Listing 2.8: Beispiel - Event-Driven Consumer**

```
1  /** Example for a JMS Event-Driven Consumer */
2  public class EventDrivenConsumer implements MessageListener{
3      @Override
4      public void onMessage(Message msg) {
5          /* do something usefull with the received message */
6      }
7  }
8
9  /** How to use the Event-Driven Consumer in an application */
10 public class Application{
11     Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
12     Destination destination = session.createQueue(myQueue);
13     MessageConsumer msgConsumer = session.createConsumer(destination);
14     msgConsumer.setMessageListener(new EventDrivenConsumer());
15 }
```

## Selective Consumer

Sollten mehrere Empfänger Messages aus einer Queue empfangen wollen, diese jedoch nach unterschiedlichen Kriterien gefiltert, so kommen Selective Consumer (siehe Abbildung 2.19) zum Einsatz. Diese überprüfen, bevor die Message empfangen wird, ob die in der Message übertragenen Parameter mit dem Filterkriterium übereinstimmen.

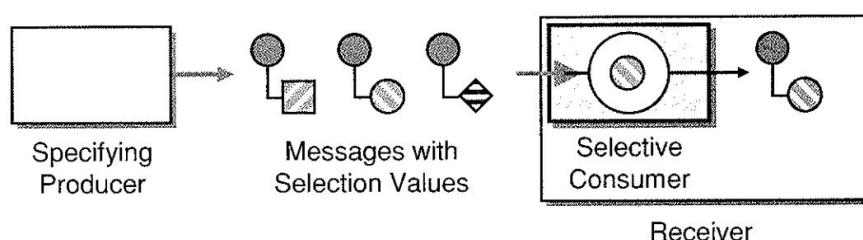


Abbildung 2.19: Selective Consumer ([Hohpe2005], S.516)

So können zum Beispiel über eine Queue die unterschiedlichsten Messages bzw. Messagetypen übertragen werden, zB Auftragsstart („start-job“) und Auftragsende („end-job“). Je nachdem welcher Parameter gesetzt wird, reagiert ein unterschiedlicher Empfänger auf die Messages.

Listing 2.9 zeigt wie ein Selective Consumer in JMS realisiert werden kann.

#### Listing 2.9: Beispiel - Selective Consumer

```
1  /** Example for a Selective JMS Consumer */
2
3  /** MessageProducer */
4  TextMessage msg = session.createTextMessage();
5  msg.setText (Job=010101 start);
6  msg.setStringProperty("type", "start-job");
7  MessageProducer producer = session.createProducer(myQueue);
8  producer.send(msg);
9
10 /** Selective MessageConsumer */
11 String msgSelector = "type = 'start-job'"
12 MessageConsumer selConsumer = session.createConsumer(myQueue,
   ↴msgSelector);
```

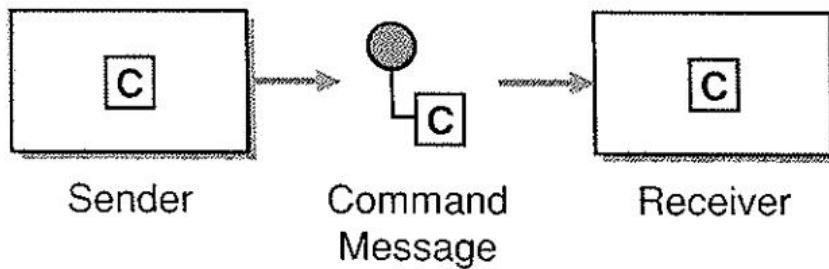
## 2.2.4 Message Construction

Es gibt unterschiedliche Intentionen, warum Messages zwischen Applikationen ausgetauscht werden. Es lassen sich drei Varianten erkennen; erstens Messages welche dazu führen sollen, dass der Empfänger eine Funktion ausführt (siehe *Command Message*), zweitens, Messages die Daten von einer Applikation zu einer anderen transferieren ohne eine Reaktion darauf zu erwarten (siehe *Document Message* auf Seite 25) und drittens, Messages welche den Empfänger über ein Ereignis in der sendenden Applikation informieren (siehe *Event Message* auf Seite 27). (vgl. [Hohpe2005], S.143-144)

In vielen Fällen ist es notwendig einen Reply auf eine Message zu senden (siehe *Request-Reply* auf Seite 28) und diese entsprechend zuzuordnen (siehe *Correlation Identifier* auf Seite 30).

## Command Message

Der Einsatz von Command Messages (siehe Abbildung 2.20 auf der nächsten Seite) ermöglicht es, ähnlich wie bei *RPC* (vgl. Punkt 2.1.3 auf Seite 5), Funktionen eines entfernten Systems aufzurufen. Im Gegensatz zum synchron ablaufenden *RPC* ermöglicht die Verwendung von Command Messages einen asynchronen Aufruf von Funktionen.



**C** = getLastTradePrice('DIS');

Abbildung 2.20: Command Message ([Hohpe2005], S.145)

Command Messages stellen zB in JMS keinen eigenen Typ von Messages dar, sondern beschreiben nur, dass Aufgrund dieser Message am Empfänger eine Funktion ausgeführt werden soll. Der Funktionsaufruf kann zB in einer JMS TextMessage, oder auch als serialisiertes Objekt in einer JMS ObjectMessage verpackt werden.

Ein Beispiel für eine Command Message stellt ein SOAP<sup>2</sup> Request dar, welcher eine XML Struktur innerhalb der Message kapselt, welche sowohl den Funktionsaufruf an sich als auch die benötigten Parameter beschreibt. Weiterführende Informationen zu SOAP sind unter <http://www.w3.org/TR/soap/> zu finden.

Listing 2.10 zeigt wie eine Command Message verwendet werden kann, um eine Funktion aufzurufen. Es wird nur gezeigt wie die Message empfangen, ausgewertet und die Funktion aufgerufen wird. Das Senden des Ergebnisses funktioniert analog zum Senden der Anfrage, es kommt jedoch eine andere Queue zum Einsatz. Siehe mehr dazu unter *Request-Reply* auf Seite 28.

#### Listing 2.10: Beispiel - Command Message

```
1  /** Example for a Command Message using JMS */
2
3  /** MessageProducer - send a request to another application*/
4  TextMessage msg = session.createTextMessage();
5  msg.setText("<exec><method>getLastTradePrice</method><param><id><br>
6  -DIS</id></param></exec>");
7  MessageProducer producer = session.createProducer(myQueue);
8  producer.send(msg);
9
10 /** MessageConsumer - executes the function specified in the <br>
11 -message with the given parameters */
12 MessageConsumer consumer = session.createConsumer(myQueue);
```

<sup>2</sup>Simple Object Access Protocol

```
11 Message recvMsg = consumer.receive(50);
12 if(recvMsg instanceof TextMessage){
13     TextMessage tmpTxtMsg = (TextMessage) recvMsg;
14     /** extract methodname from message body */
15
16     String methodName = ...; //getLastTradePrice
17     /** extract parameter id from message body */
18
19     String id = ...; //id=DIS
20
21     /** find and call method */
22     /** lookup all methods which are defined in this class */
23     Method[] m = this.getClass.getDeclaredMethods();
24     for (Method method : m){
25         /** search for a method with name = methodName*/
26         if(method.getName().equals(methodName)){
27             /** check that the given parameters match the parameter \
28                 →definition from the method */
29             if(params.size == method.getParameterTypes().length){
30                 /** check that the type of the parameters match */
31                 /** set flag if method could be identified or not */
32                 boolean foundMethod = ...; /** true | false */
33                 if (foundMethod){
34                     /** execute method */
35                     try{
36                         String returnValue = (String) method.invoke(this, id);
37                         /** send a response message with the returnValue after \
38                             →successfull method execution */
39                     }catch(Exception e){...}
40                 }
41             }
42         }
43
44     /** Method that shall be executed */
45     public String getLastTradePrice(String id){
46         /** find last trade price */
47         return lastTradePrice;
48     }
```

## Document Message

Document Messages (siehe Abbildung 2.21 auf der nächsten Seite) werden verwendet, um Daten zwischen verschiedenen Applikationen auszutauschen. Im Gegensatz zu *Command Messages* (siehe auf Seite 23), wird von der sendenden Applikation nicht zwingend eine Reaktion erwarten. Vielfach geht es schlicht darum, Informationen auf einen gemeinsamen Stand zu halten. (vgl. [Hohpe2005], S.147-148)

Ebenso wie Command Messages, stellen Document Messages keinen eigenen Typ dar, sondern beschreiben lediglich deren Funktion.

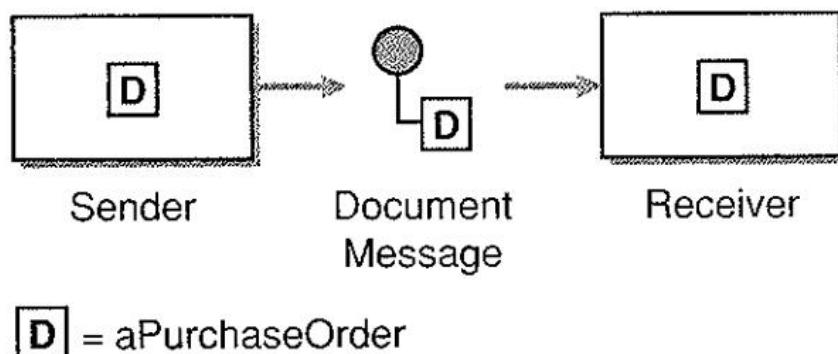


Abbildung 2.21: Document Message ([Hohpe2005], S.148)

In JMS gibt es die unterschiedlichsten Möglichkeiten Daten in einer Message zu kapseln. Handelt es sich um reinen Text, bzw um Daten, welche sich in Form von Strings abbilden, lassen so liegt es nahe, eine JMS TextMessage zu verwenden. Sollen Objekte zwischen den Applikationen ausgetauscht werden, so kann auf JMS ObjectMessage zurückgegriffen werden, welche eine serialisierte Abbildung des Objektes beinhaltet.

Listing 2.11 zeigt wie Daten in einer Message übertragen werden können.

#### Listing 2.11: Beispiel - Document Message

```
1  /** Example for a Document Message using JMS */
2
3  /** create a String containing XML to represent the order, the \
     -other possibility would be to create an XML Document and use \
     - its serialized form as message content by using an \
     -ObjectMessage instead of an TextMessage */
4
5  String order = "<order>
6          <customerId>0101</customerId>
7          <item>
8              <itemId>1234</itemId>
9              <quantity>3</quantity>
10             </item>
11         </order>";
12
13 TextMessage msg = session.createTextMessage();
14 msg.setText(order);
15 producer.send(msg);
```

## Event Message

Event Messages (siehe Abbildung 2.22) informieren entfernte Applikationen darüber, dass in der sendenden Applikation ein Event ausgelöst wurde, welcher für weitere Applikationen von Interesse sein könnte. Im Gegensatz zu *Document Messages* (siehe auf Seite 25), ist der Inhalt dieser Messages oft vernachlässigbar, da ihr reines Vorhanden sein bereits eine entsprechende Aktion auslösen kann.

Event Messages werden zumeist über Publish-Subscribe Channels (siehe Publish-Subscribe Channel auf Seite 12) verteilt, da ein Event für mehr als eine Applikation von Interesse sein kann. Im Gegensatz zu Document Messages, welche meist nur für einen einzelnen Empfänger bestimmt sind. (vgl. [Hohpe2005], S.151-153)

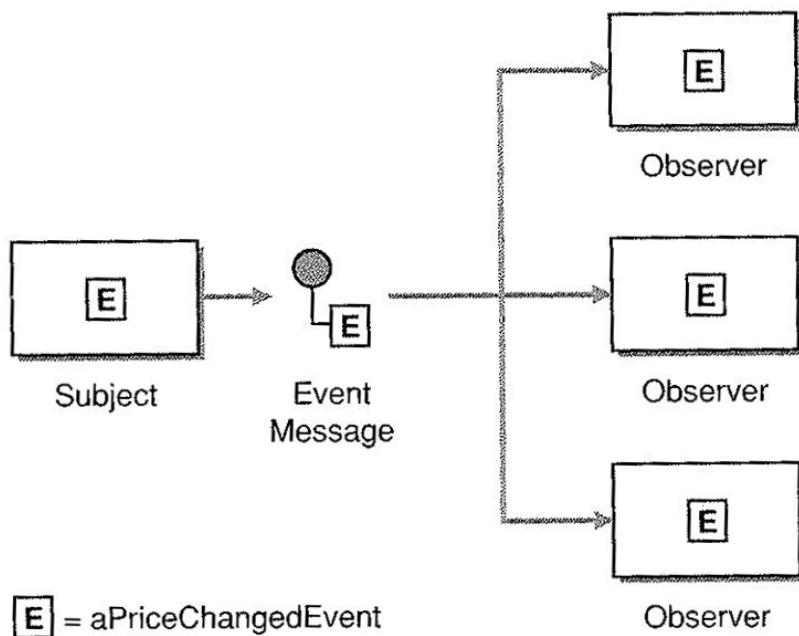


Abbildung 2.22: Event Message ([Hohpe2005], S.152)

Ebenso wie Command Messages und Document Messages, stellt eine Event Message keinen eigenen Message Typ dar, sondern kann durch jegliche JMS Message Typen (zB TextMessage, ObjectMessage) realisiert werden. Da jedoch bei Event Messages der Inhalt der Message nur einen geringen Informationsgehalt bietet, sollte die Verwendung von JMS TextMessages in den meisten Fällen allen Anforderungen genügen.

Wird eine Event Message empfangen, so muss der Empfänger entschei-

den, wie er diese Information behandelt und entsprechend darauf reagieren. Der Sender erwartet jedoch keinerlei Antwort auf eine Event Message.

Werden im Zuge der Übertragung einer Event Message, in deren Body auch Daten übertragen, so kann man bereits von einer Document Message sprechen. Eine vollständige Trennung zwischen Document und Event Message wird in vielen Fällen in der Praxis nur schwer möglich sein.

Listing 2.12 zeigt wie eine reine Event Message in JMS realisiert werden kann.

#### Listing 2.12: Beispiel - Event Message

```
1  /** Example for an Event Message using JMS */
2
3  /** create an empty TextMessage and set a String Message \n
   *  -Property indicating the type of event occurred */
4
5  final static String PRICECHNGD = "aPriceChangedEvent";
6  TextMessage msg = session.createTextMessage();
7  msg.setStringProperty("EVENTTYPE",PRICECHND);
8  producer.send(msg);
```

## Request-Reply

Request-Reply (siehe Abbildung 2.23 auf der nächsten Seite) wird verwendet, um eine Two-way Kommunikation zwischen verschiedenen Applikationen zu ermöglichen. Wie bereits bei *Command Messages* (siehe auf Seite 23) beschrieben, können über Messages Methoden in entfernten Applikationen aufgerufen werden. In den meisten Fällen wird auf einen Methodenaufruf ein Rückgabewert erwartet. Um einen Rückgabewert zu übertragen, liegt der Einsatz von *Document Messages* (siehe auf Seite 25) nahe.

Betrachtet man die Request Message als Methodenaufruf, so sind grundsätzlich drei verschiedene Rückgabewerte möglich; erstens **void**, zweitens **Return value** und drittens **Exception**. (vgl. [Hohpe2005], S.154-158)

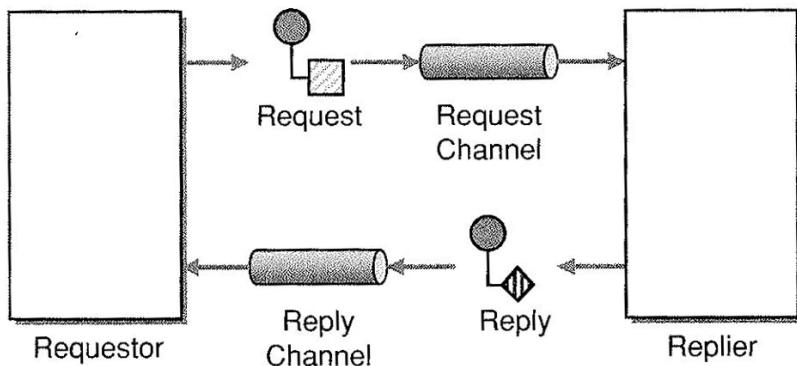


Abbildung 2.23: Request-Reply ([Hohpe2005], S.155)

Da Message Channels (siehe Punkt 2.2.2 auf Seite 10) nur Nachrichten in eine Richtung transferieren können, müssen zwei unterschiedliche Channels eingesetzt werden, um eine Two-way Kommunikation zu ermöglichen. Meist ist der Name des Channels welcher für die Reply Message verwendet wird, Teil der Request Nachricht. (vgl. [Hohpe2005], S.154-158)

Das Übermitteln des Namens des Reply Channels wurde von Hohpe and Woolf als eigenes Pattern (**Return Address**) beschrieben. Siehe weitere Informationen hierzu unter ([Hohpe2005], S.159ff). Da dieses jedoch in direktem Zusammenhang mit Request-Reply steht, wird dieses nicht gesondert behandelt.

Listing 2.13 zeigt ein einfaches Request-Reply Beispiel anhand von JMS.

#### Listing 2.13: Beispiel - Request-Reply

```

1  /** Example for Request-Reply using JMS */
2
3  /** create a QueueRequestor and send a request Message */
4  Destination requestQueue = session.createQueue(RequestChannel);
5  Destination replyQueue = session.createQueue(ReplyChannel);
6
7  Message requestMsg = session.createTextMessage();
8  requestMsg.setText("<exec>...</exec>");
9  requestMsg.setJMSReplyTo(replyQueue); //set JMS Property - \
   -JMSReplyTo - to specify where the reply message is expected
10
11 QueueRequestor requestor = new QueueRequestor(session, \
      -requestQueue);
12
13 /** send the request message and wait for the result */
14 Message reply = requestor.request(requestMsg);
15
16 /** Receive a request (synchronous) and send the reply to the \
      -given Channel */
17 Destination requestQueue = session.createQueue(RequestChannel);

```

```
18 MessageConsumer consumer = session.createConsumer(requestQueue);  
19 Message requestMsg = consumer.receive(50);  
20  
21 /** perform the action defined in the request message and create  
→ a reply message */  
22 Message replyMsg = session.createTextMessage();  
23 replyMsg.setText("REPLY");  
24 Destination replyQueue = requestMsg.getJMSReplyTo(); //read the  
→reply channel information from the request message and  
→create the corresponding destination  
25 MessageProducer replier = session.createProducer(replyQueue);  
26 replier.send(replyMsg);
```

Da eine Applikation durchaus mehrere Request Messages versenden kann, muss eine Möglichkeit geschaffen werden, eine Reply Message der entsprechenden Request Message zuzuordnen. Um dies zu realisieren, werden so genannte *Correlation Identifiers* (siehe auf dieser Seite) eingesetzt.

## Correlation Identifier

Wie bereits unter *Request-Reply* (siehe auf Seite 28) beschrieben, ist es Vielfach der Fall, dass Applikationen auf eine Message (Request) eine entsprechende Antwort (Reply) erwarten. Um in einem asynchronen System die Antworten den entsprechenden Anfragen korrekt zuordnen zu können, kommen Correlation Identifiers zum Einsatz.

Wie in Abbildung 2.24 auf der nächsten Seite dargestellt, wird mit jedem Request eine eindeutige MessageID übertragen. Diese MessageID wird durch den Replier in die CorrelationID umgewandelt. Somit lässt sich am Requestor eindeutig feststellen, welcher Reply aufgrund welches Requests empfangen wurde. (vgl. [Hohpe2005], S.163-169)

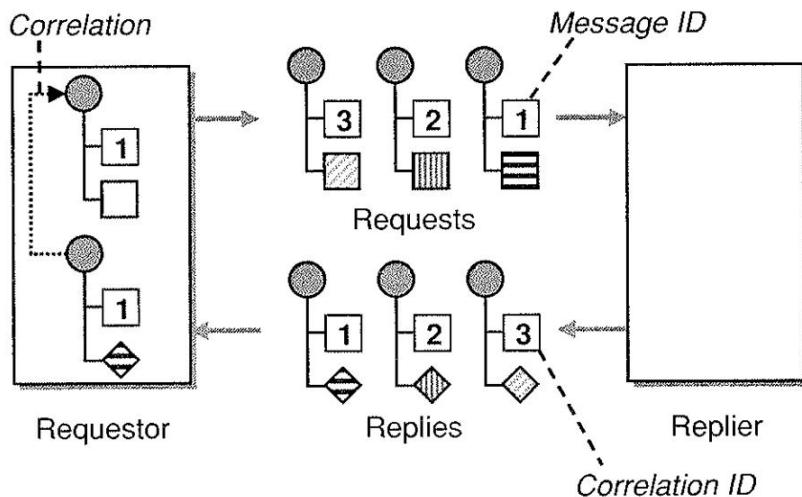


Abbildung 2.24: Correlation Identifier ([Hohpe2005], S.164)

JMS fügt jeder übertragenen Nachricht automatisch eine MessageID hinzu. Vielfach ist diese aber nur schlecht geeignet, die Geschäftsprozesse zu unterstützen. Es kann zum Beispiel der Fall eintreten, dass eine Applikation bereits eine eindeutige ID für eine Transaktion verwendet. In diesem Fall wäre es effizienter, diese TransaktionsID als CorrelationID zu verwenden. (vgl. [Hohpe2005], S.166)

Listing 2.14 zeigt wie die JMS MessageID als CorrelationID verwendet werden kann.

**Listing 2.14: Beispiel - Correlation Identifier**

```

1  /** Example showing how to use Correlation Identifiers */
2
3  /** read the MessageID from the Message and set CorrelationID */
4  String id = requestMsg.getJMSMessageID();
5  replyMsg.setJMSCorrelationID(id);
6
7  /** receive reply message, read correlation ID and perform the \n
     -corresponding action */
8  Message replyMsg = consumer.receive(50);
9  String correlationId = replyMsg.getJMSCorrelationID();
10
11 /** search for the request message and continue processing - \n
      -request message IDs may be stored in a HashMap (\n
      -pendingRequests = new HashMap<String, Message>() ) */
12 Message requestMsg = pendingRequests.get(correlationId);
13 /** perform whatever may be needed here */

```

Es besteht die Wahrscheinlichkeit, dass Komponenten des Messagingsystems Message Header Informationen verändern und dadurch der Einsatz der MessageID als Correlation Identifier nicht möglich ist. Siehe hierzu, zum Beispiel,

Wire Tap auf Seite 40. Für diese Fälle muss eine andere Information als Correlation Identifier herangezogen werden.

## 2.2.5 Message Routing

Message Routing (siehe Abbildung 2.25) erlaubt eine starke Separierung zwischen den Applikationen, die Messages senden und jenen, die diese empfangen. Durch Message Routing wird es ermöglicht, dass eine Applikation nur einen einzelnen Channel benötigt, um mit n Applikationen zu kommunizieren.

Es besteht jedoch ein klarer Unterschied zwischen Message Routing und *Publish-Subscribe* (siehe auf Seite 12). Mit Hilfe eines Message Routers werden Nachrichten aufgrund bestimmter Eigenschaften oder deren Inhalt von einem Channel zu n verschiedenen Channels weitergeleitet. Im Gegensatz dazu, wird bei Publish-Subscribe die Nachricht an alle Empfänger weitergeleitet. (vgl. [Hohpe2005], S.225-229)

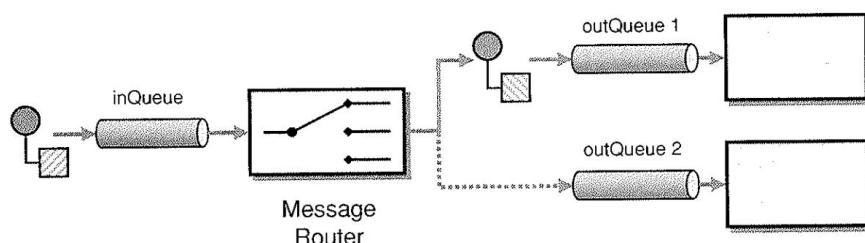


Abbildung 2.25: Message Router ([Hohpe2005], S. 80)

Listing 2.15 zeigt anhand von JMS wie ein einfacher Message Router realisiert werden kann.

### Listing 2.15: Beispiel - Message Routing

```
1  /** Example showing Message Routing based on the Message Type */
2
3  /** create consumer and producer(s) */
4  MessageConsumer receiver = session.createConsumer("myQueue");
5  MessageProducer producerA = session.createProducer("channelA");
6  MessageProducer producerB = session.createProducer("channelB");
7
8  /** read Message */
9  Message msg = receiver.receive(50);
10 /** lookup what type of Message it is. If it is an instance of
     -TextMessage it should be routed to Channel A otherwise to
     -Channel B */
11
```

11

```

12 if(msg instanceof TextMessage) {
13     producerA.send(msg);
14 } else{
15     producerB.send(msg);
16 }
```

## Content-based Router

Content-based Router (siehe Abbildung 2.26) bieten eine einfache Möglichkeit, Nachrichten aufgrund ihrer Attribute oder ihres Inhaltes an den korrekten Empfänger weiterzuleiten.

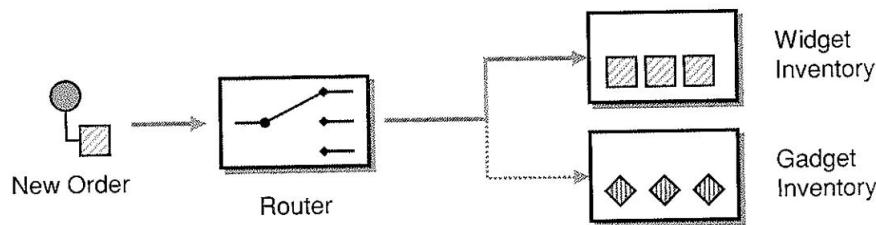


Abbildung 2.26: Content-based Router ([Hohpe2005], S.232)

Listing 2.16 zeigt ein einfaches JMS Beispiel für einen Content-based Router.

### Listing 2.16: Beispiel - Content-based Router

```

1  /** Example showing Message Routing based on the message content
   * /
2
3  /** create consumer and producer(s) */
4  MessageConsumer receiver = session.createConsumer("myQueue");
5  MessageProducer producerA = session.createProducer("channelA");
6  MessageProducer producerB = session.createProducer("channelB");
7
8  /** read Message */
9  Message msg = receiver.receive(50);
10 /** if the content of the message matches the pattern "100:.*"
    -> send the message to Channel A otherwise send it to Channel B
    -> */
11
12 if(msg instanceof TextMessage) {
13     TextMessage tmp = (TextMessage) msg;
14     String content = tmp.getText();
15     // Create the pattern
16     Pattern p = Pattern.compile("100:.*");
17     // Create a matcher with the message content as input
18     Matcher m = p.matcher(content);
19     boolean matches = m.find();
20     if(matches) {
```

```

21     producerA.send(msg);
22 }else{
23     producerB.send(msg);
24 }
```

Der Nachteil dieser Lösung besteht darin, dass wenn neue Consumer hinzugefügt oder bestehende entfernt werden, der Source Code des Content-based Routers angepasst werden muss. In diesem Sinne ist der Content-based Router in dieser Form nur bedingt flexibel einsetzbar.

Um diese Probleme handzuhaben kann ein Dynamic Router (siehe Abbildung 2.27) verwendet werden.

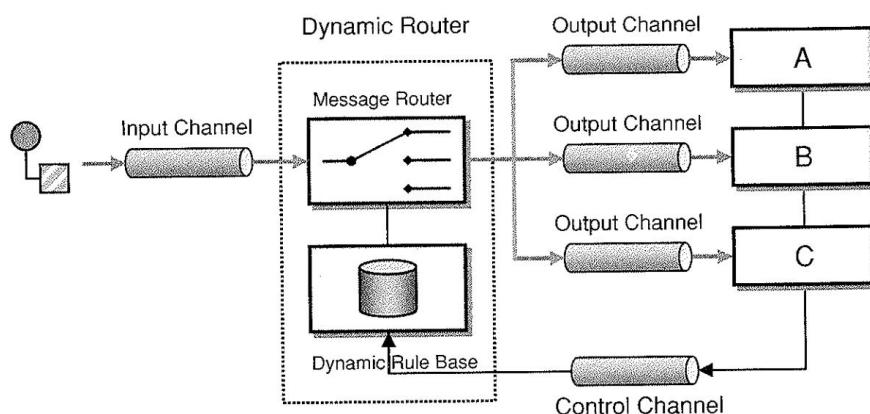


Abbildung 2.27: Dynamic Router ([Hohpe2005], S.244)

Im Gegensatz zu einem normalen Content-based Router werden, die Regeln nach denen die Nachrichten verteilt werden, nicht in den Source Code geschrieben, sondern konfiguriert.

Applikationen, welche durch den Router Nachrichten erhalten möchten, melden sich mit den benötigten Informationen (zB Channel, Parameter für das Message Routing) über einen Control Channel am Dynamic Router an bzw. ab. (vgl. [Hohpe2005], S.243-248)

## Resequencer

Resequencer (siehe Abbildung 2.28 auf der nächsten Seite) werden dazu verwendet, um Nachrichten in einem Channel neu zu ordnen. Dies kann über die Attribute oder den Inhalt der Nachricht geschehen. So lassen sich über einen Resequencer anhand der JMS Priority Nachrichten mit höherer Priorität vor-

anreihen, womit diese vor den anderen Nachrichten zugestellt werden. (vgl. [Hohpe2005], S.283-293)

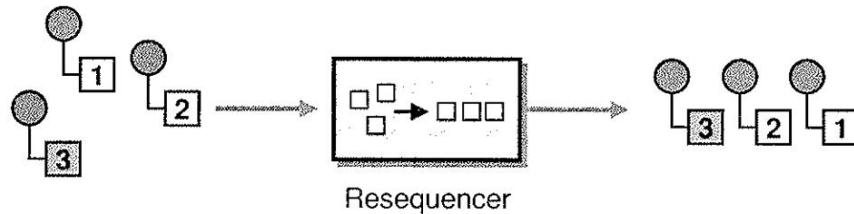


Abbildung 2.28: Resequencer ([Hohpe2005], S.284)

Listing 2.17 zeigt in Pseudo Code die Grundsätzliche Funktionsweise eines Resequenczers.

**Listing 2.17: Beispiel - Resequencer (Pseudo Code)**

```
1  /** Pseudo Code Example showing a resequencer - ordering ↴
   * →messages by their priority */
2
3  /** if a new message arrives:
4   * 1. get all pending messages from the channel
5   *    - this messages are already ordered
6   * 2. compare their priority
7   * 3. insert the new message
8   */
9
10 onMessage(Message msg){
11     boolean inserted = false;
12     Messages[] pending = channel.getPendingMessages(); // 1.
13     foreach(Message tmp in pending){
14         if(msg.getPriority() > tmp.getPriority()){ // 2.
15             pending.addBefore(tmp, msg); // 3. - add the new message ↴
16             →before tmp
17             inserted = true;
18             break;
19         }
20     }
21     if(!inserted){
22         pending.addAsLast(msg); // add the message at the last ↴
23         →position because it has the lowest priority
24     }
25 }
```

## Message Broker

Message Broker stellen kein Design Pattern, sondern ein Architektur Pattern dar. Ein Message Broker stellt einen zentralen Punkt zur Verteilung der Nach-

richten dar. In diesem Sinne kann ein Message Broker einer Hub-and-Spoke Architektur (siehe Abbildung 2.6b auf Seite 8) gleichgesetzt werden.

(vgl. [Hohpe2005], S.322-326)

Message Broker sollten als dynamische Content-based Router realisiert werden, um auch während der Laufzeit des Systems neue Verbindungen und Parametrierungen zuzulassen. Es lassen sich, wie in Abbildung 2.29 dargestellt, Message Broker untereinander verbinden, was sowohl die Wartbarkeit, als auch die Verfügbarkeit des Gesamtsystems erhöht, da es keinen Single-Point of Failure mehr gibt. Des Weiteren muss jeder Message Broker nur einen Teil der Nachrichten bearbeiten, dadurch bleibt das Gesamtsystem übersichtlich.

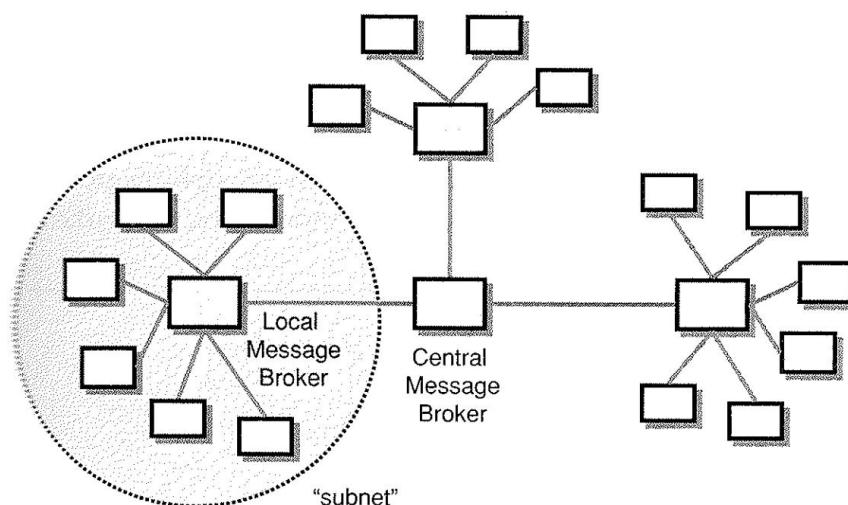


Abbildung 2.29: Message Broker ([Hohpe2005], S.325)

## 2.2.6 Message Transformation

Im Rahmen der Enterprise Application Integration stellt sich das Problem, dass unterschiedliche Applikationen eine unterschiedliche Sicht auf Daten haben und diese somit auch unterschiedlich darstellen. Aus diesem Grund wird die Message Transformation herangezogen, um Daten von einer Darstellung bzw. einem Format in ein anderes umzuwandeln. (vgl. [Hohpe2005], S. 327)

Es gibt zum einen die Möglichkeit, von je einem Format in ein anderes zu konvertieren, (siehe dazu *Message Translator* auf der nächsten Seite), oder aber von jedem Format zu einem gemeinsamen Format zu konvertieren (siehe dazu *Canonical Data Model* auf Seite 38).

## Message Translator

Ein Message Translator (siehe Abbildung 2.30) dient dazu, Nachrichten von einem Format in ein anderes zu übersetzen. Dies ist oftmals notwendig da unterschiedliche Applikationen ein und die selben Daten in einer anderen Repräsentation erwarten.

Werden zum Beispiel XML Daten in einer Nachricht versandt, so ist es sehr wahrscheinlich, dass der Sender und der Empfänger unterschiedliche XML Darstellungen benötigen. (vgl. [Hohpe2005], S.85-94)

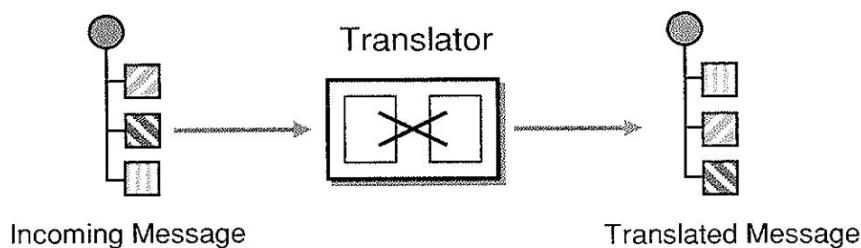


Abbildung 2.30: Message Translator ([Hohpe2005], S.86)

In Listing 2.18 wird aufgezeigt, wie XML Daten mittels XLS von einem Format in ein anderes übertragen werden können.

### Listing 2.18: Beispiel - Message Translator

```
1  /** read the XML Content from a message and transform it */
2
3  if(msg instanceof TextMessage) {
4      TextMessage tmp = (TextMessage) msg;
5      String xml = tmp.getText();
6      /**
7       * The String xml may look like this:
8       * <customer>
9       *   <firstname>Alexander</firstname>
10      *   <lastname>Marktl</lastname>
11      * </customer>
12     */
13     StringWriter writer = new StringWriter();
14
15    /**
16     * a XSLT Stylesheet may look like this
17     */
18    String xslt = "<?xml version=\"1.0\"?>" +
19        "<xsl:stylesheet version=\"1.0\" xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\">" +
20        "<xsl:template match=\"/\\/*\">" +
21        "  <customer><name>" +
22        "    <xsl:value-of select=\"customer/firstname\"/>" +
23        "  </xsl:text> </xsl:template>" +
24        "  <xsl:value-of select=\"customer/lastname\"/>" +
```

```

23             "</name></customer>" +
24             "</xsl:template>" +
25             "</xsl:stylesheet>";
26
27     Transformer transformer = TransformerFactory.newInstance().  

28         newTransformer(new StreamSource(new StringReader(xslt)));
29     transformer.transform(new StreamSource(new StringReader(xml)),  

30         new StreamResult(writer));
31
32     String result = writer.toString();
33     /** the result would look like this:  

34      * <customer><name>Alexander Markt1</name></customer>  

35      */
36 }

```

## Canonical Data Model

Bei einem Canonical Data Model (CDM) (siehe Abbildung 2.31a) werden alle benötigten Formate in ein gemeinsames Format (siehe Abbildung 2.31b) transformiert und wieder zurück. Dadurch reduziert sich die Anzahl der benötigten Message Translator erheblich (vgl. Abbildung 2.32 auf der nächsten Seite).

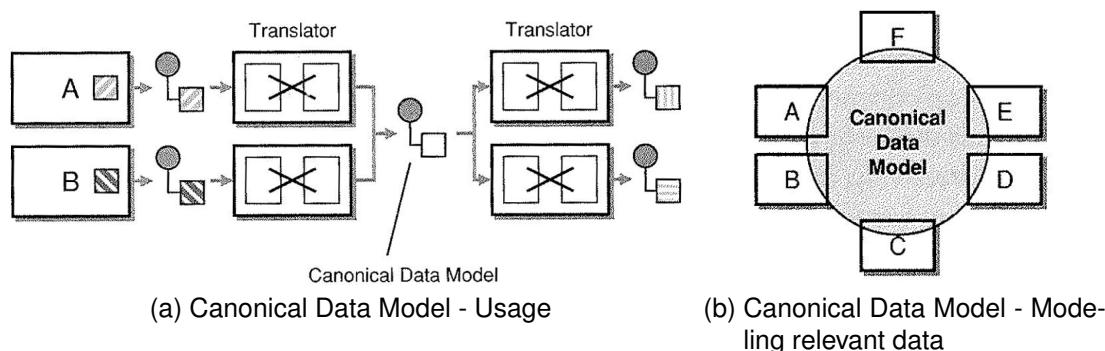


Abbildung 2.31: Canonical Data Model ([Hohpe2005], S. 356, 359)

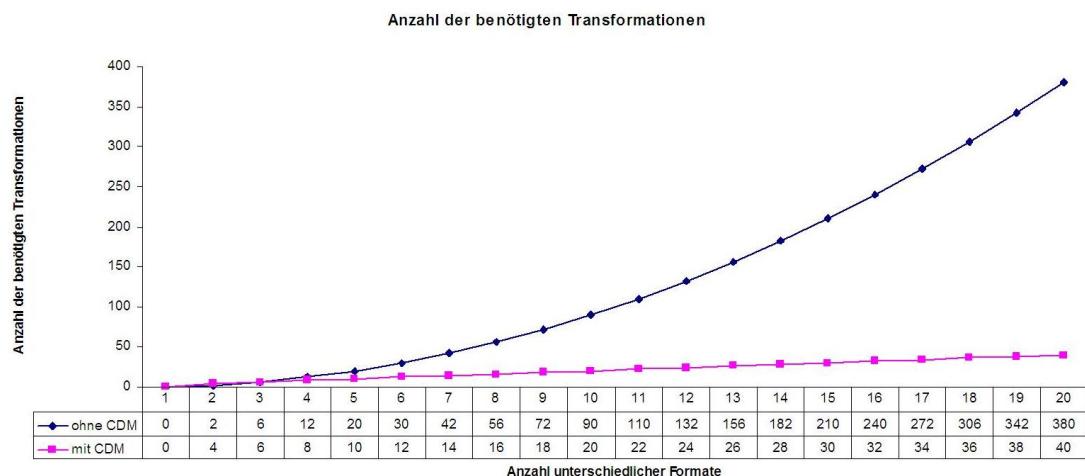


Abbildung 2.32: Anzahl benötigter Message Translator

## 2.2.7 System Management

Um ein Messagingsystem effizient betreiben und warten zu können, müssen Möglichkeiten geschaffen werden, sowohl eine Gesamtsicht auf das System zu schaffen als auch Nachrichten im Detail zu analysieren.

Die Gesamtsicht auf ein Messagingsystem kann über einen Control Bus gewährleistet werden, um Nachrichten im Detail zu betrachten, kann ein Wire Tap eingesetzt werden. (vgl. [Hohpe2005], S. 537-539)

## Control Bus

Der Control Bus (siehe Abbildung 2.33 auf der nächsten Seite) dient auf der einen Seite dazu, das System zu monitoren und auf der anderen dazu, neue Konfigurationen während der Laufzeit des Systems an die beteiligten Komponenten zu übertragen.

Um diese Aufgaben bearbeiten zu können, müssen alle Komponenten zwei unterschiedliche Kommunikationskanäle verwenden. Einen für die normale Übertragung von Nachrichten und einen für die Übertragung von Monitoring- und Steuerungsinformationen. Dadurch, dass Message Channels unidirektional sind, müssen für den Control Bus mindestens zwei Channels vorgesehen werden. (vgl. [Hohpe2005], S.540-541)

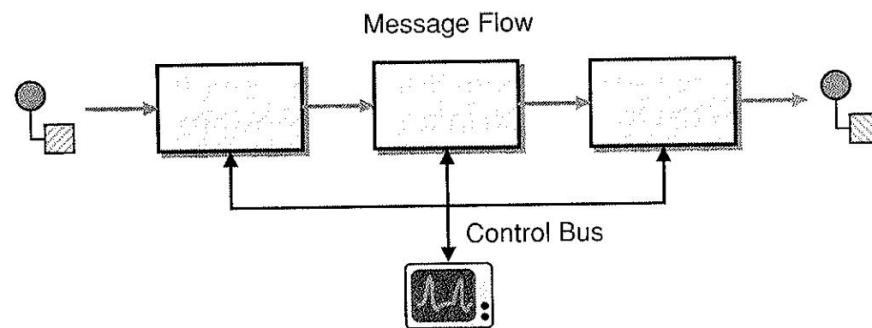


Abbildung 2.33: Control Bus ([Hohpe2005], S.541)

Nachrichten, die über den Control Bus übertragen werden, können in fünf Gruppen zusammengefasst werden: Konfigurations-, Heartbeat-, Test-, Exception- und Statistik-Nachrichten. Im Idealfall sollte pro Gruppe ein eigener Channel definiert werden. (vgl. [Hohpe2005], S.542-544)

**Konfigurations-Nachrichten** Dienen dazu, Komponenten während der Laufzeit neu zu parametrisieren. Dadurch kann die Anzahl der Konfigurationsdateien für die einzelnen Komponenten im Idealfall auf null reduziert werden.

**Heartbeat-Nachrichten** Alle Komponenten senden in einem spezifizierten Intervall kurze Nachrichten an den Control Bus, um anzugeben, dass sie fehlerfrei funktionieren.

**Test-Nachrichten** Test-Nachrichten werden dazu verwendet, um festzustellen, ob eine Komponente Nachrichten vollständig und korrekt bearbeitet.

**Exception-Nachrichten** Fehler in einer Komponente werden an den Control Bus weitergeleitet, um ein zentrales Fehler Monitoring zu gewährleisten.

**Statistik-Nachrichten** Geben, zum Beispiel, Auskunft über den durchschnittlichen Durchsatz an Nachrichten, oder wieviele Nachrichten aktuell in einem Channel gehalten werden.

## Wire Tap

Wire Taps (siehe Abbildung 2.34 auf der nächsten Seite) werden dazu verwendet, Nachrichten, die über einen Point-to-Point Channel übertragen werden, zu analysieren. Um dies zu ermöglichen, empfängt der Wire Tap die Nachrichten und sendet diese unverändert<sup>3</sup> sowohl an den ursprünglichen Output Channel

<sup>3</sup>Durch das Messaging System können Headerdaten zB MessageID verändert werden, es wird lediglich garantiert das der Message Content unverändert bleibt

als auch an einen weiteren Channel, der zu monitoring Zwecken herangezogen wird.

Bei Publish-Subscribe Channels ist der Einsatz von Wire Taps sinnlos, da um die selbe Funktionalität zu erreichen, einfach ein weiterer Empfänger hinzugefügt werden kann. (vgl. [Hohpe2005], S.547-550)

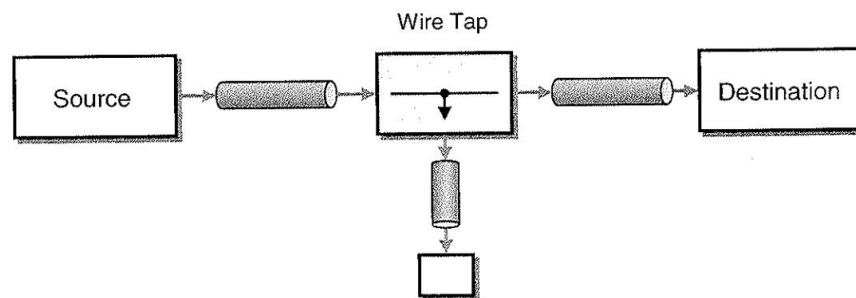


Abbildung 2.34: Wire Tap ([Hohpe2005], S.548)

Da durch den Wire Tap neue Nachrichten mit dem Selben Inhalt wie die empfangene Nachricht generiert werden, aber Änderungen im Message Header zu erwarten sind, muss darauf geachtet werden, dass als *Correlation Identifier* (siehe auf Seite 30) nicht die MessageID herangezogen wird.

Listing 2.19 zeigt in Pseudo Code die Funktionsweise eines Wire Taps.

**Listing 2.19: Beispiel - Wire Tap (Pseudo Code)**

```
1  /** read message from the input channel, duplicate it and send \
   *  →it to the output channel as well as to the monitoring \
   *  →channel */
2
3  Message inputMsg = InputChannel.getMessage();
4  Message duplicatedMsg = new Message(inputMsg.getContent());
5
6  OutputChannel.putMessage(inputMsg);
7  MonitoringChannel.putMessage(duplicatedMsg);
```

## Message Store

Ein Message Store (siehe Abbildung 2.35 auf der nächsten Seite), ist dazu gedacht wichtige Informationen der übertragenen Nachrichten für eine spätere Analyse zu verspeichern. Dies darf nicht mit einer Persistierung der Nachrichten zur Garantierung der Zustellung verwechselt werden (siehe dazu *Guaranteed Delivery* auf Seite 16).

Da zur Analyse der Nachrichten nicht zwangsläufig auch der Inhalt der Nachrichten benötigt wird, muss abgewogen werden, welche Informationen relevant sind und somit verspeichert werden müssen. Wichtige Informationen können, zum Beispiel, Sender, Empfänger und Messagegröße sein. (vgl. [Hohpe2005], S.555-557)

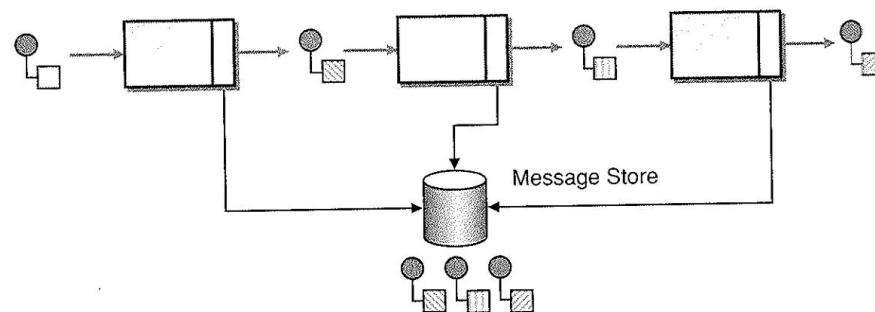


Abbildung 2.35: Message Store ([Hohpe2005], S. 556)

Um den normalen Transport der Nachrichten nicht übergebührend zu beeinflussen, können die Nachrichten, zum Beispiel, über einen *Wire Tap* (siehe auf Seite 40) an einen eigenen Channel weitergeleitet werden, welcher in einen Message Store mündet.

Die Daten werden meist in einer Datenbank abgelegt. Hierzu ist es notwendig, die zu speichernden Inhalte der Nachrichten zu kennen und das Datenbank Schema entsprechend aufzubauen, um eine spätere Analyse der Daten optimal zu unterstützen. (vgl. [Hohpe2005], S.555-557)

## Smart Proxy

Smart Proxys werden, ähnlich wie Wire Taps (siehe *Wire Taps* auf Seite 40), dazu verwendet Nachrichten, zu analysieren bzw. an eine Kontroll Instanz weiterzuleiten. Im Gegensatz zu einem einfachen Wire Tap, können mittels Smart Proxys Request-Reply Szenarien (siehe *Request-Reply* auf Seite 28) überwacht werden.

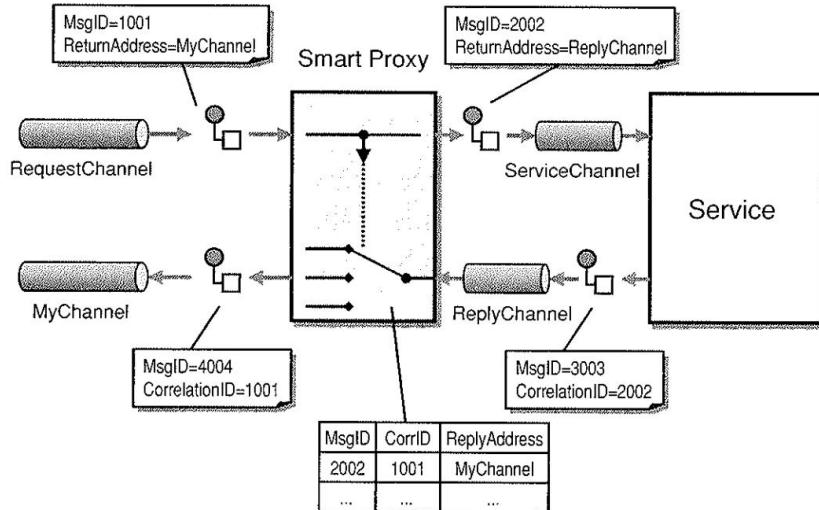


Abbildung 2.36: Smart Proxy ([Hohpe2005], S.561)

In vielen Fällen ist der Channel, auf dem sich der Sender eine Antwort erwartet, in der Nachricht enthalten (zB JMSReplyTo - Property). Um sowohl den Request als auch den Reply überwachen zu können, muss ein Smart Proxy die Request Nachricht abfangen, den Channel auslesen auf welchem die Antwort erwartet wird, diese Information verspeichern und die Nachricht an den Empfänger übermitteln.

Die Reply Nachricht wird ebenfalls abgefangen und aufgrund der gespeicherten Informationen der Request Nachricht an den richtigen Channel weitergeleitet. Dieser Ablauf wird in Abbildung 2.36 schematisch dargestellt.

## 2.3 Architekturempfehlung

Aufgrund der in den Punkten 2.1 auf Seite 2 und 2.2 auf Seite 7 beschriebenen Integrationsmöglichkeiten und Entwurfsmuster soll an dieser Stelle eine Empfehlung für die Architektur eines Messagingsystems, welches den Anforderungen der Firma SSI Schäfer PEEM genügt, aufgezeigt werden.

Abbildung 2.37 auf der nächsten Seite zeigt eine mögliche Architektur für ein Messagingsystem. Der Aufbau dieser Architektur basiert auf den EAI Patterns und orientiert sich an den Grundsätzen welche in Abbildung 2.8 auf Seite 10 dargestellt werden.

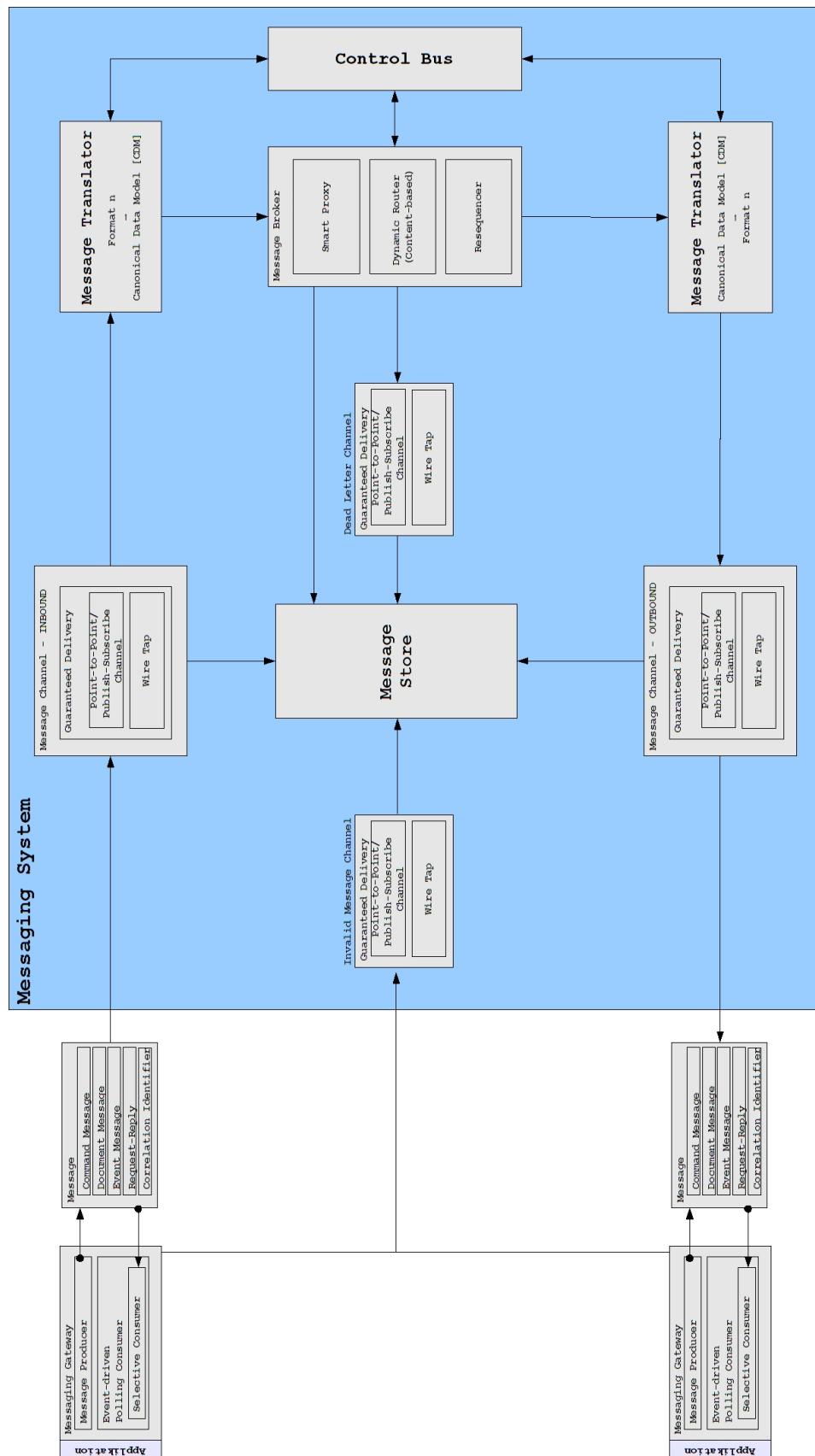


Abbildung 2.37: Architekturempfehlung

Auf Seite der Applikationen sollte ein Messaging Gateway eingesetzt werden, um eine klare Trennung zwischen den Aufgaben der Applikation und dem notwendigen Messaging Code zu gewährleisten. Die Message Consumer sollten, je nach Anforderung, als Polling bzw. Event-driven Consumer ausgeführt werden, welche jeweils wiederum als Selective Consumer realisiert werden, um irrelevante bzw. fälschlich an diesem Empfänger weitergeleitete Messages zu ignorieren.

Die zu übertragenden Messages sollten, wie auf Seite 23 beschrieben, je nach Anforderung als Command-, Document-, Event- Message ausgeführt werden. Um eine Two-way Kommunikation zwischen den einzelnen Applikationen zu realisieren, müssen die Anforderungen von Request-Reply und des Correlation Identifiers berücksichtigt werden.

Da sichergestellt werden muss, dass jegliche Message zugestellt wird, muss jeder Channel unter Bedacht auf Guaranteed Delivery aufgebaut werden. Je nach Anforderung können hier sowohl Point-to-Point als auch Publish-Subscribe Channels zum Einsatz kommen. Unter Bedachtnahme auf die aktuelle Situation im Unternehmen, werden in erster Linie Point-to-Point Channels als notwendig erachtet.

Um neue Applikationen möglichst ohne großen Aufwand anbinden zu können, sollte ein Canonical Data Model (CDM) gefunden werden und jede Nachricht von jeweiligen Quellformat zum CDM und im Anschluss in das benötigte Zielformat transformiert werden. Grundsätzlich wäre es empfehlenswert, den Inhalt der Nachrichten in XML zu strukturieren.

Der Message Broker sollte als dynamischer Content-based Router realisiert werden. So kann der Nachrichtenfluss während der Laufzeit des Systems gesteuert werden. Zusätzlich sollte ein Resequencer zum Einsatz kommen, um Nachrichten mit höherer Priorität in den Channels entsprechend vor die anderen Nachrichten zu reihen.

Ein Smart Proxy bzw. einzelne Wire Taps sorgen dafür, dass wichtige Informationen über Nachrichten in einem zentralen Message Store verspeichert werden, welcher durch eine eigene Applikation zur Analyse der Daten ausgewertet wird.

Um das Messagingsystem zu überwachen und an einer zentralen Stelle zu konfigurieren, sollte ein Control Bus realisiert werden. Dieser stellt einen zentralen Punkt des Messagingsystems dar, da er aktiv das Verhalten der Komponenten (zB der Message Router) beeinflusst und durch das Heartbeat Signal erkennt, welche Applikationen derzeit vollfunktionsfähig sind. Des Wei-

teren werden alle auftretenden Fehler an den Control Bus gemeldet, ebenso wie statistische Informationen.

Um fehlerhafte Nachrichten gesondert zu behandeln, muss auf einen Invalid Message Channel zurückgegriffen werden. Es muss dafür Sorge getragen werden, dass es einen Prozess gibt, der Nachrichten in diesem Channel ausliest und entsprechend darauf reagiert.

Weiters muss ein Dead Letter Channel verwendet werden, um Nachrichten, welche nicht erfolgreich zugestellt werden konnten, aufzufangen. Analog zum Invalid Message Channel muss auch hier ein eigener Prozess vorgesehen werden, welcher Nachrichten in diesem Channel bearbeitet.

# 3

## Kapitel 3

# Evaluierung von Messagingsystemen

Der strukturierte Evaluierungsprozess (vgl. Abbildung 3.1) hilft Produkte objektiv einander gegenüber zu stellen, um somit die optimale Lösung aus der Vielzahl der verfügbaren Produkte heraus zu filtern (vgl. Abbildung 3.2 auf der nächsten Seite). (vgl. [Schreiber2003], S.36, 220)

Als Basis für die Evaluierungen dienen die Definitionen der Anforderungen (siehe 3.1 auf Seite 49). Aus den Anforderungen werden in weiterer Folge der Kriterienkatalog und die entsprechenden Bewertungslisten abgeleitet (siehe 3.2 auf Seite 65). Anhand der Bewertungslisten können die Produkte auf einer einheitlichen Basis verglichen werden.

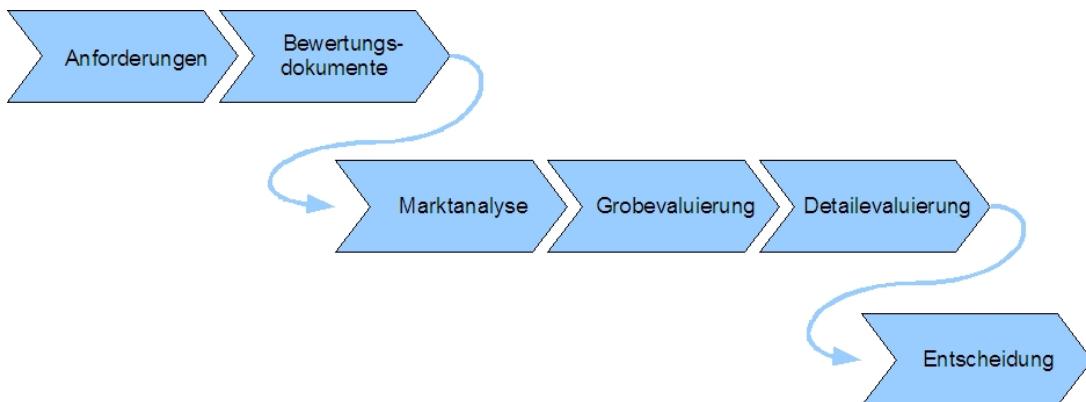


Abbildung 3.1: Auswahlverfahren (vgl. [Schreiber2003], S.36)

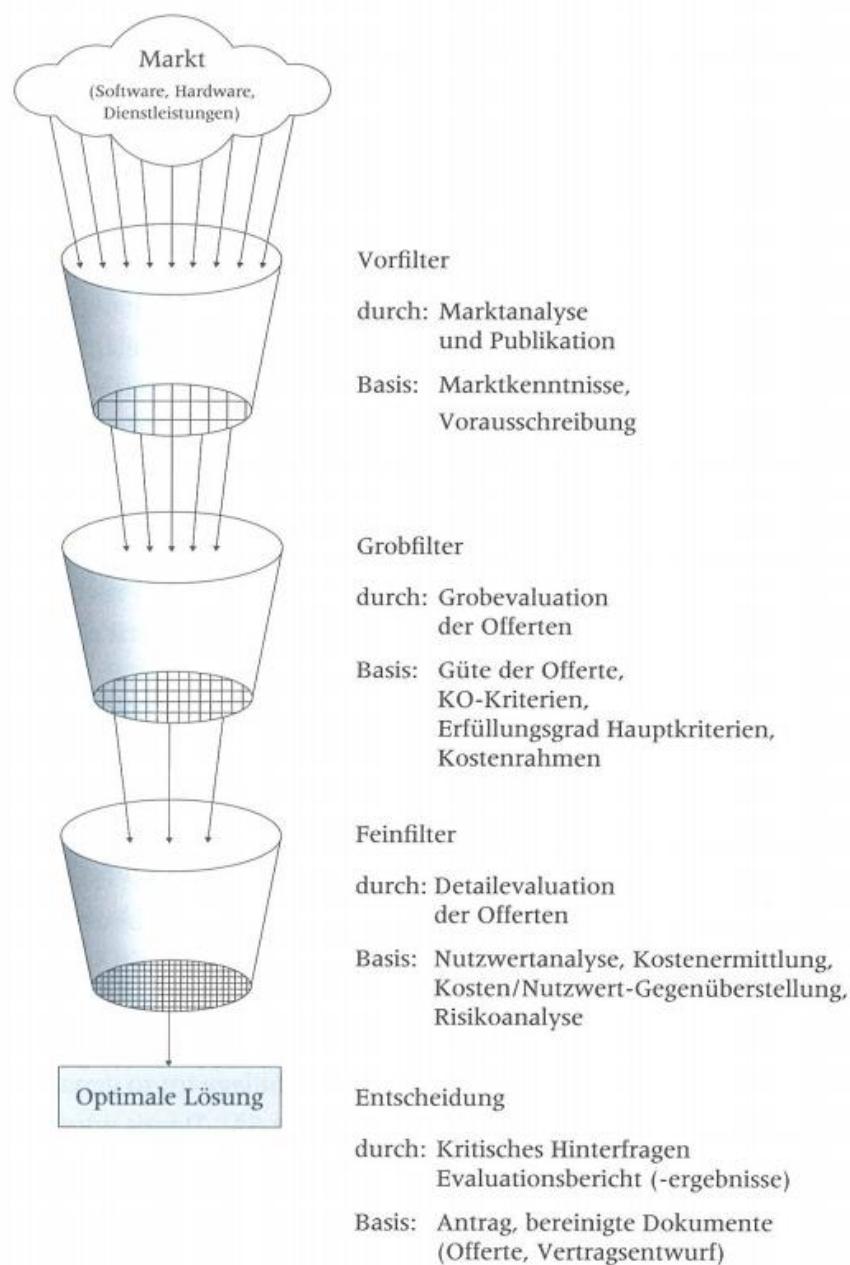


Abbildung 3.2: Auswahlverfahren ([Schreiber2003], S.182)

Um eine möglichst umfassende Sicht auf die aktuelle Marktsituation zu gewährleisten, werden die zu evaluierenden Produkte sowohl aus dem Bereich der proprietären Lösungen als auch aus dem Bereich der Open Source Lösungen ausgewählt. Im Rahmen dieser Arbeit wurden nur Produkte in Betracht genommen, welche sich nicht mehr in einem Beta-Status oder früher befinden und in diesem Sinne vollwertig einsetzbar sind.

## 3.1 Anforderungsprofil, Definitionen und Evaluierungen

Das Anforderungsprofil beschreibt, welche Funktionen und Leistungen durch das System bzw durch dessen Hersteller abgedeckt werden müssen und dient in weiterer Folge als Grundlage bei der Erstellung des Kriterienkataloges. Der direkte Zusammenhang zwischen dem Anforderungsprofil und dem Kriterienkatalog wird in Abbildung 3.3 auf Seite 66 dargestellt. Die Anforderungen an das System wurden in Absprache mit dem Unternehmen und im speziellen mit den betroffenen Entwicklern erarbeitet und definiert. Die einzelnen Anforderungen an das System wurden in drei Gruppen zusammengefasst.

- Applikationsanforderungen
- Systemplattform
- Anbieterbezogene Kriterien

Die Informationen zu den Punkten Evaluierung beziehen sich darauf, wie im Laufe des Evaluierungsprozesses die jeweiligen Kriterien getestet und bewertet werden können. Hinsichtlich der verwendeten Punkteskala und weiterführenden Informationen zu den Bewertungen wird auf Punkt 3.2 auf Seite 65 verwiesen.

### 3.1.1 Applikationsanforderungen

#### Funktionen/Features

##### Bearbeitungsdauer

**DEFINITION:** Die Bearbeitungsdauer beschreibt den Zeitraum zwischen dem Senden der Nachricht an das Messagingsystem und deren Zustellung an den Client. Nicht zur Bearbeitungsdauer werden die Bearbeitung der Nachricht am Client und eine mögliche Antwort gerechnet.

**ANFORDERUNGEN:** Da es sich im Bereich der Industrieanlagensteuerung meist um zeitkritische Informationen handelt, muss darauf geachtet werden,

dass diese Informationen innerhalb des dafür definierten Zeitfensters zuge stellt werden.

Zeitkritische Informationen müssen in Echtzeit verarbeitet werden, der Begriff Real-Time bzw. Echtzeitverarbeitung muss nicht zwangsläufig darauf hindeuten, dass eine Verarbeitung von Daten oder deren Transfer besonders schnell erfolgen muss. Generell wird mit diesem Begriff nur ausgesagt, dass eine Verarbeitung nicht länger dauern darf als der entsprechende Ablauf in der realen Welt Zeit in Anspruch nimmt. Eine entsprechende Definition liegt der JSR1-RTSJ<sup>1</sup> zugrunde.

*„The programming environment must provide abstractions necessary to allow developers to correctly reason about temporal behavior of application logic. It is not necessary fast, small or exclusively for industrial control; it is about predictability of the execution of application logic with respect to time.“ ([BrianGoetz2008])*

Aufgrund dieser Definition lässt sich festhalten, dass bei Real-Time Systemen nicht die schnelle Verarbeitung das ausschlaggebende Kriterium ist, sondern vielmehr der Umstand, dass die jeweiligen Zeitspannen fest definiert sind und es so dem Entwickler ermöglicht wird, die Dauer der Bearbeitung exakt zu bestimmen und seine Arbeit darauf auszurichten.

Um eine korrekte Funktion des Gesamtsystems zu gewährleisten, wurde firmenintern festgelegt, dass die Bearbeitungsdauer einen Schwellwert von einer Millisekunde pro Nachricht nicht überschreiten sollte. Als maximal akzeptierte Bearbeitungsdauer wird, vom Unternehmen, eine Zeitspanne von zwei Millisekunden angesehen.

**EVALUIERUNG:** Die Bearbeitungsdauer einer einzelnen Nachricht wird durch entsprechende Test eruiert, hierfür werden 10.000 Nachrichten von einem Client an das Messagingsystem übertragen und von einem weiteren Client abgerufen. Die Übertragung der Messages erfolgt in einem persistenten Übertragungsmodus. Die Größe einer einzelnen Nachricht wurde auf 200Byte festgelegt, da dies den Messagegrößen entspricht, die im Unternehmen auftreten.

Um die Bearbeitungsdauer einer Nachricht zu erhalten, wird die Dauer der gesamten Transaktion, dh Zustellung aller 10.000 Nachrichten, gemessen. Im Anschluss wird die Bearbeitungsdauer für eine Nachricht aus der gesamten

<sup>1</sup> Java Specification Request - Real-time Specification for Java

Transaktionsdauer heraus gerechnet (siehe Formel 3.1 auf der nächsten Seite).

#### Formel 3.1: Bearbeitungszeit

$$\text{Bearbeitungszeit pro Nachricht} = \frac{t_{trans}}{m}$$

$t_{trans}$  = Dauer der Transaktion [ms]

$m$  = Anzahl der übertragenen Nachrichten

---

Sollte die berechnete Bearbeitungszeit weniger als eine Millisekunde betragen, so kann die Höchstbewertung vergeben werden. Liegt der Wert zwischen einer und zwei Millisekunden so kann eine Bewertung mit zwei erfolgen. Alles was die maximal zulässige Bearbeitungszeit von zwei Millisekunden überschreitet muss mit null bewertet werden.

## Durchsatz

**DEFINITION:** Der Punkt Durchsatz steht in engem Zusammenhang mit der Bearbeitungszeit (siehe auf Seite 49) und beschreibt, wieviele Nachrichten das System innerhalb eines bestimmten Betrachtungszeitraumes verarbeiten, dh von einem Sender an einen Empfänger vermitteln kann. Der Betrachtungszeitraum wurde auf eine Sekunde festgelegt.

**ANFORDERUNGEN:** Das aktuelle System muss derzeit rund 200-300 Nachrichten in der Sekunde verarbeiten, dies steigt aber kontinuierlich an. Um auf die steigenden Anforderungen der kommenden Jahre entsprechend schnell und flexibel reagieren zu können, sollte das eingesetzte Messagingssystem einen Durchsatz von 1000 Nachrichten in der Sekunde schaffen. Dies ergibt sich durch die gewünschte Bearbeitungszeit, welche auf einen Wert von einer Millisekunde pro Nachricht festgelegt wurde (siehe auf Seite 49). Die maximal zulässige Bearbeitungszeit wurde auf zwei Millisekunden festgelegt, woraus sich ein Minimaldurchsatz von 500 Nachrichten pro Sekunde ergibt, welcher nicht unterschritten werden darf.

**EVALUIERUNG:** Der Durchsatz wird ebenso wie die Bearbeitungsdauer durch das Versenden von 10.000 Nachrichten zwischen zwei Clients getestet. Der Durchsatz wird in Folge laut Formel 3.2 berechnet.

**Formel 3.2: Durchsatz**

$$\text{Durchsatz} = \frac{m}{t_{trans}}$$

$m$  = Anzahl der übertragenen Nachrichten

$t_{trans}$  = Dauer der Transaktion [ms]

---

Sollte der Durchsatz einen Wert von 1000 Nachrichten pro Sekunde überschreiten, so kann die Höchstbewertung vergeben werden. Liegt der Durchsatz zwischen 750 und 1000 Nachrichten pro Sekunde kann eine Bewertung mit drei erfolgen. Bei einem Durchsatz zwischen 500 und 750 Nachrichten pro Sekunden kann eine Bewertung mit zwei vorgenommen werden. Jeglicher Durchsatz, der unterhalb von 500 Nachrichten pro Sekunde liegt, muss als unzureichend klassifiziert werden und in diesem Sinne mit null bewertet werden.

## Priorisierung

**DEFINITION:** Unterschiedliche Messages haben eine unterschiedliche Dringlichkeit, das bedeutet, dass Messages mit einer höheren Dringlichkeit auch mit höherer Priorität bearbeitet werden müssen. Sollten in einer Queue zB 100 Nachrichten mit der geringer Priorität vorhanden sein und es langt eine Message mit höherer Priorität ein, so muss diese vor den bereits vorhandenen Messages bearbeitet bzw zugestellt werden.

**ANFORDERUNGEN:** Das Messagingsystem muss eine Möglichkeit zur Priorisierung von Messages bereitstellen.

**EVALUIERUNG:** Es wird überprüft ob eine Priorisierung möglich ist, wie diese intern gehandhabt wird ist jedoch unerheblich und wird im Laufe der Eva-

luerungen nicht beachtet. Sollte das Messagingsystem die Priorisierung nativ unterstützen, so kann die Höchstbewertung vergeben werden. Sollte eine Priorisierung durch Erweiterungspakete möglich sein, kann eine Bewertung mit zwei Punkten vorgenommen werden. Sollte generell keine Unterstützung möglich sein, ist eine Bewertung mit null Punkten vorzunehmen.

## Persistierung

**DEFINITION:** Persistenz sagt aus, dass Objekte und deren Informationsgehalt dauerhaft auf einem nicht flüchtigem Medium<sup>2</sup> verspeichert werden, um diese nach einem Ausfall oder geplanten Systemstopp wieder zur Verfügung zu haben. Eine mögliche Definition des Begriffs Persistierung lautet wie folgt.

*„Bei Objekten ist es die Eigenschaft, dass das Objekt auch nach einem ausgeführten Prozess noch existiert. Persistente Objekte überdauern kurzzeitige Unterbrechungen oder Spannungsabfälle ohne, dass sich ihr Informationsinhalt ändert.“ ([ITWISSEN2009])*

**ANFORDERUNGEN:** Die Messages müssen bis zur Bestätigung des vollständigen Empfanges vom Messagingsystem persistent gehalten werden. Die Persistierung der Messages darf sich nicht negativ auf die Performance (Bearbeitungsdauer und Durchsatz) des Messagingsystems auswirken. Die Persistierung sollte entweder gegen eine ins Messagingsystem integrierte Lösung oder eine externe Datenbank erfolgen. Bei Persistierung in eine externe Datenbank ist darauf zu achten, dass Oracle Datenbanken unterstützt werden müssen.

**EVALUIERUNG:** Es wird überprüft, ob das Messagingsystem eine Möglichkeit zur Persistierung bietet, jedoch werden die internen Abläufe zur Persistierung nicht evaluiert. Erfolgt die Persistierung in ein internes System, so kann die Höchstbewertung vergeben werden. Erfolgt die Persistierung in eine externe Datenbank, so müssen zumindest die Datenbankprodukte der Firma Oracle unterstützt werden. Sollte dies nicht der Fall sein, so muss eine Bewertung mit null erfolgen. Wird generell keine Persistierung angeboten, so muss ebenfalls eine Bewertung mit null vergeben werden.

<sup>2</sup>zum Beispiel: Festplatte, Band-Speicher

## Logging

**DEFINITION:** Logging soll die Administration und das Monitoring eines Systems unterstützen, in dem es Informationen zu den Objekten und deren Status bzw Fehler- und Erfolgsmeldungen protokolliert. Ullenboom hebt die Wichtigkeit und Möglichkeiten des Loggings hervor.

*„Das Loggen (Protokollieren) von Informationen über Programmzustände in externe Speicher ist ein wichtiger Teil, um später den Ablauf und Zustände von Programmen rekonstruieren und verstehen zu können. Mit einer Logging-API lassen sich Meldungen in eine XML-Datei, Textdatei oder Datenbank schreiben oder über einen Chat verbreiten.“  
([Ullenboom2009])*

**ANFORDERUNGEN:** Das Logging muss an die jeweiligen Erfordernisse angepasst werden können, dh, dass im laufenden Betrieb zB keine DEBUG Einträge aufgenommen werden. Dies soll dazu führen, eventuelle Probleme zielgerichtet und effizient zu erkennen. Die Logging Einträge müssen in eine Datei verspeichert werden können, um eine spätere Nachvollziehbarkeit zu gewährleisten.

Da im Unternehmen log4j (<http://logging.apache.org/log4j>) bereits erfolgreich im Einsatz ist, wäre es wünschenswert, wenn dies auch durch das Messagingsystem verwendet werden würde.

**EVALUIERUNG:** Es wird überprüft, ob das Messagingsystem ein konfigurierbares Logging bietet. Wenn das Messagingsystem auf log4j setzt, so kann die Höchstbewertung vergeben werden, sollte eine andere Loggingtechnologie zum Einsatz kommen, so hat eine Bewertung mit zwei zu erfolgen. Wenn durch das Messagingsystem keinerlei Logging Möglichkeiten geboten werden, so muss eine Bewertung mit null erfolgen.

## Software-Merkmale

### Verfügbarkeit

**DEFINITION:** Piedad und Hawkins beschreiben drei unterschiedliche Stufen der Verfügbarkeit, beginnend von High Availability (Das System ist während

der vereinbarten Zeiten, zB Montag bis Freitag 7:00 bis 18:00, durchgehend verfügbar, es gibt keinerlei ungeplante Ausfälle) über Continuous Operations (Das System ist 24 Stunden, 7 Tage die Woche verfügbar, es gibt keinerlei ungeplante Ausfälle) bis hin zu Continous Availability (Das System ist 24 Stunden, 7 Tage die Woche verfügbar, es gibt keinerlei geplante und ungeplante Ausfälle). (vgl. [Piedad2001], S. 16)

Im Rahmen der Industrieanlagensteuerung muss abgewogen werden, welche dieser Stufen den größten Nutzen bietet. Da ein Großteil aller Anlagen der Firma SSI Schäfer PEEM im Schichtbetrieb<sup>3</sup> arbeiten, finden wir mit einer Verfügbarkeit in der Stufe von High Availability das Auslangen.

Unter der Annahme, dass es in jedem System zu unvorhersehbaren Verhalten und somit auch zu Ausfällen kommen kann, lässt sich die Verfügbarkeit des Systems laut Formel 3.3 berechnen.

### Formel 3.3: Verfügbarkeit

$$Verfügbarkeit = \frac{t_b - t_u}{t_b} * 100$$

$t_u$  = Zeitraum der Unverfügbarkeit

$t_b$  = Betrachtungszeitraum

---

Der Betrachtungszeitraum ( $t_b$ ) entspricht der Betriebsdauer einer Anlage innerhalb einer Schicht und beträgt im Schnitt elf Stunden.

**ANFORDERUNGEN:** Das Unternehmen SSI Schäfer PEEM garantiert seinen Kunden eine Verfügbarkeit von 99,3% betreffend die Gesamtanlage. In diesem Sinne muss die Verfügbarkeit des Messagingsystems höher liegen als diese 99,3%. In Zusammenarbeit mit dem Unternehmen wurde eine Verfügbarkeit von 99,9% festgelegt.

Unter der Annahme, dass eine Verfügbarkeit von 99,9% gewährleistet werden soll, ist mit einer maximalen Ausfallzeit von 0,01 Stunden, entspricht 6 Minuten, innerhalb des Betrachtungszeitraumes zu rechnen.

---

<sup>3</sup>zB: Montag - Freitag 7:00 - 18:00

**EVALUIERUNG:** Tests im Hinblick auf die Verfügbarkeit fokussieren sich darauf die Dauer der Reparaturzeiten zu erfassen und diese in Folge entsprechend zu optimieren. (vgl. [Microsoft2009a])

*„Das Testen der Verfügbarkeit bedeutet, dass eine Anwendung für einen vorher festgelegten Zeitraum ausgeführt wird, Fehlerereignisse und Reparaturzeiten erfasst werden und die prozentuale Verfügbarkeit mit der ursprünglichen Vereinbarungen über das Betriebsniveau verglichen wird.“ ([Microsoft2009a])*

Das Testen der Verfügbarkeit muss soweit wie möglich an das reale spätere Einsatzgebiet angepasst werden. Daraus ergibt sich, dass Verfügbarkeitstest eine kosten- und zeitintensive Aufgabe darstellen.

Derartige Langzeittests sind im Rahmen dieser Arbeit nicht vorgesehen, insofern wird bei der Beurteilung der Verfügbarkeit auf die im Rahmen der Tests erfahrene Verfügbarkeit zurückgegriffen. Des Weiteren werden die Herstellerangaben in die Beurteilung einbezogen.

Sollten im Laufe der Evaluierungen kein Probleme hinsichtlich der Verfügbarkeit festgestellt werden, und auch vom Hersteller keinerlei gegenteilige Angaben zur Verfügbarkeit gemacht werden, so kann die Höchstbewertung vergeben werden. Andernfalls ist eine Bewertung mit null vorzunehmen.



**Achtung!** Da keinerlei Langzeittest durchgeführt werden, kann keine definitive Aussage zur Verfügbarkeit getroffen werden. Die Bewertung entspricht somit lediglich einer Prognose.

## Skalierbarkeit

**DEFINITION:** In Anlehnung an Josef Schreiber, definiert sich die Skalierbarkeit eines Systems daran, dass zum einen die Hardware auf die Bedürfnisse des Kunden optimiert werden kann und zum anderen daran, dass ein und die selbe Software auf Systemen unterschiedlicher Leistungsfähigkeit eingesetzt werden kann. (vgl. [Schreiber2003], S. 91)

In Bezug auf Messagingsysteme wird der Terminus Skalierbarkeit in der relativen Unabhängigkeit zur Anzahl der vom System verwalteten Message-Queues bzw. der verwalteten Sender und Empfänger definiert. Dies bedeutet,

dass eine steigende Zahl an Sendern und/oder Empfängern weder die Bearbeitungszeit noch den Durchsatz signifikant negativ beeinflussen darf.

**ANFORDERUNGEN:** Es muss sowohl die Bearbeitungsdauer (siehe auf Seite 49) als auch der Durchsatz (siehe auf Seite 51) unabhängig von der Anzahl der Sender/Empfänger sein.

**EVALUIERUNG:** Sollten sowohl Bearbeitungsdauer als auch Durchsatz innerhalb der definierten Anforderungen bleiben, so kann die Höchstbewertung vergeben werden. Unterliegen diese Punkte jedoch Schwankungen, welche die definierten Bereich überschreiten, so muss dies entsprechend negativ bewertet werden. Sollte die Überschreitung nicht mehr als 10% betragen, so kann eine Bewertung mit zwei erfolgen, alles darüber hinaus muss mit null bewertet werden.

Im Rahmen der Tests wird die Anzahl an Sendern/Empfängern kontinuierlich gesteigert, beginnend bei jeweils zehn Sendern/Empfängern bis zu 50 Sendern/Empfängern, wobei die Steigerung in zehner Schritten erfolgt.

## Sprachunabhängigkeit

**DEFINITION:** Sprachunabhängigkeit bedeutet, dass ein System nicht auf eine einzelne Programmiersprache beschränkt sein darf. Das System muss jegliche Messages verarbeiten können, unabhängig davon, in welcher Sprache Sender und Empfänger geschrieben wurden. Daraus folgt, dass das Messagingsystem die entsprechenden APIs zur Verfügung stellen muss.

**ANFORDERUNGEN:** Als Minimalanforderung gelten die im Unternehmen im Einsatz befindlichen Sprachen Java, Perl und plsql. Eine breite Unterstützung hinsichtlich aktueller Programmiersprachen ist wünschenswert.

**EVALUIERUNG:** Die Testprogramme werden durchgehend in Java verfasst (siehe Anhang A.2). Die Unterstützung hinsichtlich der anderen Sprachen wird nur laut Herstellerangaben überprüft. Werden die Minimalanforderungen erfüllt, so muss die Höchstbewertung vergeben werden, ist dies nicht der Fall,

so muss eine Bewertung mit null erfolgen. Sollten andere Programmiersprachen durch den Einsatz von STOMP (<http://stomp.codehaus.org/>) eingebunden werden können, so kann eine Bewertung von drei vergeben werden, da ein zusätzliches System benötigt wird, welches weitere Unsicherheitsfaktoren dem Gesamtsystem hinzufügt. Dies muss auch entsprechend im Kriterium *Abhängigkeiten zu Fremdsystemen* berücksichtigt werden.

## Abhängigkeiten zu Fremdsystemen

**DEFINITION:** Als Abhängigkeit werden zB die Verwendung eines zusätzlichen Application Servers, einer zusätzlichen Programmbibliothek oder eines Frameworks angesehen. Dem gegenüber wird eine JVM<sup>4</sup> oder ANT nicht als Abhängigkeit gesehen, da diese vorrausgesetzt werden können.

**ANFORDERUNGEN:** Abhängigkeiten zu Fremdsystemen stellen ein essentielles Problem dar, da jedes System zusätzliche Unsicherheiten ins Gesamtsystem einbringt. In diesem Sinne soll das Messagingsystem keinerlei zusätzliche Abhängigkeiten mit sich bringen.

**EVALUIERUNG:** Hinsichtlich der Beurteilung der Abhängigkeiten zu Fremdsystemen, ist ein System, das keinerlei Abhängigkeiten aufweist mit der höchsten Punktzahl zu bewerten. Zusätzliche Abhängigkeiten sind entsprechend negativ zu beurteilen. Sollte ein System mehr als drei Abhängigkeiten aufweisen, ist dies auf jeden Fall mit null Punkten zu beurteilen. Im Bereich von ein bis drei Abhängigkeiten ist eine Bewertung von zwei zu vergeben.

## Wartbarkeit

**DEFINITION:** Die Wartbarkeit eines Systems kann als Maß für die Leichtigkeit, mit der ein System geändert werden kann bzw. mit welchem Aufwand Fehler behoben oder neue Funktionalitäten dem System hinzugefügt werden können, angesehen werden. (vgl. [Reussner2009], S.271)

---

<sup>4</sup> Java Virtual Machine

**ANFORDERUNGEN:** Das System soll einfach installier- und wartbar sein. Die Konfiguration bzw das Monitoring soll sowohl über ein vollwertiges Management Interface als auch über ein Terminal möglich sein. Dieser Punkt steht in engem Zusammenhang mit den Anforderungen an das Logging (siehe Punkt 3.1.1 auf Seite 54).

**EVALUIERUNG:** Die Wartbarkeit ist stark subjektiv geprägt, da Personen mit unterschiedlichem Kenntnisstand und Erfahrungen diesen Punkt anders bewerten würden. Im Zuge dieser Evaluierung wird die Wartbarkeit nach dem Zeitaufwand beurteilt welcher notwendig ist, das System zu installieren. Des Weiteren wird das Vorhandensein eines Management Interfaces und dessen Funktionsumfang bewertet.

Der Zeitaufwand für die Installation, Konfiguration und verfassen der Testprogramme sollte gesamt nicht mehr als fünf Stunden betragen. In einem Bereich von unter fünf Stunden ist die Höchstbewertung zu vergeben. Bei einem Zeitaufwand bis zu zehn Stunden kann eine Bewertung mit zwei vorgenommen werden, was darüber hinaus, geht muss mit null bewertet werden.

### 3.1.2 Systemplattform

#### Betriebssystem(e)

##### UNIX

**DEFINITION:** Die Anlagen der Firma SSI Schäfer PEEM, bauen zum Großteil auf das Betriebssystem AIX von IBM<sup>5</sup> oder bei kleineren Anlagen auf SuSe Linux<sup>6</sup> auf. Generell lässt sich sagen, dass eine Vielzahl der Anwendungen der Anlagen auf einem UNIX System bzw. dessen Befehlen basieren.

**ANFORDERUNGEN:** Um ein neues Messagingsystem zu integrieren, muss es die im Einsatz befindlichen Betriebssysteme, in diesem Fall IBM AIX sowie SuSe Linux, unterstützen. Ein Wechsel auf ein anderes Betriebssystem würde umfangreiche Änderungen und Anpassungen des bestehenden Systems nach sich ziehen.

---

<sup>5</sup><http://www.ibm.com/aix>

<sup>6</sup><http://www.novell.com/de-de/linux/>

**EVALUIERUNG:** Es müssen die Systeme IBM AIX und SuSe Linux unterstützt werden. Ist dies der Fall so ist eine Bewertung mit der Höchstnote vorzunehmen. Werden die genannten Systemplattformen nicht, oder nur eines der Systeme unterstützt, so ist eine Bewertung mit null zu vergeben. Die Unterstützung weiterer Betriebssysteme hat keinerlei Einfluss auf die Bewertung.

## Kommunikationsinfrastruktur

### Local Area Network (LAN)

**DEFINITION:** Ethernet LAN stellt eines der gebräuchlichsten Netzwerke zum Austausch von Daten zwischen allen an das Netzwerk angeschlossenen Komponenten zB Rechner, Drucker dar. Ethernet wurde durch das Institute of Electrical and Electronics Engineers (IEEE) standardisiert (siehe <http://ieee802.org/3/>). Eine einfache Definition eines Local Area Networks lautet wie folgt.

*„Das Local Area Network (LAN) – beschreibt ein Netzwerk, das an ein einzelnes zusammenhängendes Areal gebunden ist, also etwa einen Raum, ein Gebäude oder maximal ein zusammenhängendes (Firmen-) Gelände.“ ([Kersken2008])*

Neben LAN gibt es noch weitere Netzwerke auf Ethernetbasis zB Metropolitan Area Network (MAN), Wide Area Network (WAN) und Global Area Network (GAN) welche jedoch aufgrund ihrer Ausdehnung keine Relevanz innerhalb dieser Arbeit finden.

**ANFORDERUNGEN:** Als Kommunikationsmedium kommt bei Anlagen der Firma SSI Schäfer PEEM ein 100MBit, bzw. bei großen Anlagen, ein 1GBit LAN zum Einsatz.

Dieses dient jedoch nicht exklusiv dem Austausch von Nachrichten, sondern wird auch durch andere Systeme verwendet. Die Zeitspanne zwischen dem Senden der Message und dem Einlangen dieser am Empfänger darf, wie unter Punkt 3.1.1 auf Seite 49 definiert, nicht mehr als eine Millisekunde betragen. Insofern muss die Übertragungsdauer weniger als eine Millisekunde betragen.

**EVALUIERUNG:** Die Sinnhaftigkeit bezüglich des Einsatzes eines Ethernet LANs wurde im Jahre 2006 durch das Unternehmen RTI<sup>7</sup> getestet und bestätigt.

*„On a 100 MBit Ethernet network, sending 1000 128-Byte packets per second, you can be 99% sure that there will not be a delay of more than 1 millisecond caused by collisions in about a 1140 years.“ ([RTI2006], S. 6)*

In diesem Sinne kann die Anforderung bzw. das Kriterium LAN mit der höchsten Punktzahl bewertet werden, da hiermit die Eignung hinsichtlich des Real-Time Transfers von Daten belegt ist.



Da das verwendete Kommunikationsmedium in allen Fällen das Selbe ist, wird dieses nur einmal betrachtet und nicht gesondert für jedes zu evaluierende System. Lediglich die Ergebnisse werden entsprechend in die einzelnen Evaluierungen übernommen.

### 3.1.3 Anbieterbezogene Kriterien

#### Firmenmerkmale

#### Marktposition

**DEFINITION:** Bei der Positionierung eines Unternehmens am Markt geht es darum, Stärken und Qualitäten des Unternehmens aus Sicht der Zielgruppe klar aufzuzeigen und sich somit positiv von den Mitbewerbern abzugrenzen. (vgl. [LorbeerDesign2007], S 14)

Die Positionierung eines Unternehmens kann über die unterschiedlichsten Parameter erfolgen, klassische Positionierungselemente sind zB Funktion, Preis, Leistungsumfang, Qualität, Erhältlichkeit und Image.

(vgl. [SDI-Research2009])

**ANFORDERUNGEN:** Die Marktposition des Unternehmens soll positiv hinsichtlich der Qualität und des Funktionsumfanges beurteilt werden, wohinge-

<sup>7</sup><http://www.rti.com/>

gen der Preis des Produktes und das Image des Unternehmens geringere Bedeutung zugeschrieben wird.

**EVALUIERUNG:** Die Marktposition eines Herstellers wird anhand klassischer Positionierungselemente und einer Gewichtung dieser bestimmt. Als Positionierungselemente kommen **Funktionsumfang, Preis, Support, Qualität** und **Image** zum Einsatz. Diese werden mit einem Faktor zwischen eins und fünf gewichtet (vgl. Tabelle 3.1).

Positionierungselement	G	B	eff. P
<b>Qualität</b>	5	0	0
<b>Funktionsumfang</b>	4	0	0
<b>Support</b>	3	0	0
<b>Preis</b>	2	0	0
<b>Image</b>	1	0	0
$\Sigma$			<b>0</b>
<b>G</b> ... Gewichtung			
<b>B</b> ... Bewertung			
<b>eff. P</b> ... effektive Punkte			

Tabelle 3.1: Marktpositionierung

Die Punkte für die einzelnen Positionierungselemente werden wie folgt vergeben.

**Qualität** Die Qualität des Produktes wird anhand der im Laufe der Evaluierung erfahrenen Qualität bewertet. Lassen sich sowohl alle administrativen Tätigkeiten als auch alle Funktionen zB Verwendung von Queues, einfach, vollständig und ohne Zusatzaufwand verwenden so kann die Höchstbewertung vergeben werden. Wird entweder die Handhabung zu komplex und undurchsichtig oder lassen sich die beschriebenen Funktionen nicht in vollem Umfang verwenden, so muss eine entsprechend geringere Bewertung mit maximal zwei Punkten vergeben werden. Können Funktionen nicht verwendet werden so hat eine Bewertung mit null zu erfolgen.

**Funktionsumfang** Das System muss in erster Linie Queues unterstützen, sowie die unter Punkt 3.1.1 auf Seite 49 beschriebenen Anforderungen. Werden diese Punkte erfüllt so kann die Höchstbewertung vergeben werden, sollten diese Punkte nicht oder nur teilweise erfüllt so muss eine Bewertung mit null vorgenommen werden.

**Support** Die Bewertung erfolgt analog zur Bewertung des Kriteriums „Support“ (siehe auf der nächsten Seite).

**Preis** Wird das Produkt kostenfrei zur Verfügung gestellt so kann die Höchstbewertung vergeben werden. Liegt der Preis für das Produkt unter € 1.000.- so kann eine Bewertung mit zwei erfolgen, übersteigt der Preis diese Marke oder sind keiner Informationen verfügbar so muss eine Bewertung mit null vorgenommen werden.

**Image** Das Image des Unternehmens wird anhand der Referenzen bewertet.

Kann das Unternehmen auf eine Vielzahl an Referenzen verweisen (> 100) so kann die Höchstbewertung vergeben werden. Werden weniger Referenzen (> 50) nachgewiesen so kann eine Bewertung mit zwei erfolgen, liegt der Wert unterhalb von 50 Referenzen so muss eine Bewertung mit null erfolgen.

Aus diesen Positionierungselementen wird eine gewichtete Summe gebildet und anhand des Ergebnisses werden die Punkte für dieses Kriterium vergeben. Tabelle 3.2 zeigt bei welchem Ergebnis welche Anzahl an Punkten zu vergeben ist.

Ergebnis(-bereich)	Punkte
0 - <2	0
2 - <5	1
5 - <8	2
8 - <10	3
10 - 12	4

Tabelle 3.2: Marktpositionierung - Punkteeinteilung

## Dienstleistungen

### Schulungen

**DEFINITION:** Schulungen sollen dazu dienen, die Handhabung des Produktes zu erlernen und entsprechendes Wissen ins Unternehmen zu transferieren.

**ANFORDERUNGEN:** Schulungen stellen einen optionalen Punkt dar. Prinzipiell sollte es ausreichen die Dokumentation des Systems zur Verfügung zu haben um sich in das System einzuarbeiten. Sollte der Hersteller des Systems Schulungen anbieten, so ist dies entsprechend positiv zu bewerten.

**EVALUIERUNG:** Sollte der Hersteller Schulungen anbieten, so ist die Höchstbewertung zu vergeben. Werden keinerlei Schulungen angeboten, muss eine Bewertung mit null vorgenommen werden. Werden Schulungen nicht direkt vom Hersteller, sondern von Drittfirmen angeboten, so kann eine Bewertung mit zwei erfolgen. Es wird nur das grundsätzliche vorhanden sein des Angebots bewertet und nicht die Qualität der jeweiligen Schulungen.

## Support

**DEFINITION:** Support, bezeichnet die Unterstützung bei jeglichen Problemen, die im Rahmen der Nutzung eines Produktes auftreten können. Es ist hierbei unerheblich, ob es sich um Hard- oder Softwareprobleme handelt.

Dies wird meist durch Support über mehrere Ebenen ermöglicht (First-, Second-, Third-Level-Support), wobei das Wissen zu den technischen Details entsprechend von First- zu Third-Level-Support kontinuierlich ansteigt. (vgl. [SAP2009])

**ANFORDERUNGEN:** Es muss umfassender Support bereitgestellt werden, dieser muss neben allgemeinen Fragestellungen auch auf technische Details bezugnehmen können.

**EVALUIERUNG:** Wird professioneller Support durch den Hersteller angeboten, so ist dies dementsprechend mit der Höchstbewertung zu beurteilen. Wird professioneller Support durch einen Drittanbieter angeboten so kann eine Bewertung mit zwei erfolgen. Werden nur Foren und Mailinglisten als Support Medium herangezogen, hat eine Beurteilung mit eins zu erfolgen. Sollte kein Support vorhanden sein so ist dieser Punkt mit null zu bewerten.

## Bearbeitungsrecht

**DEFINITION:** Das Recht Änderungen an einem bestehenden System durchzuführen. Dies soll es dem Unternehmen, ermöglichen auftretende Fehler selbstständig zu korrigieren bzw. notwendige Verbesserungen durchzuführen. Dieses Recht wird meist nur von Open Source Herstellern, zB Apache, eingeräumt.

*„You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form“ ([Apache2004])*

**ANFORDERUNGEN:** Eigenständige Änderungen müssen gewährt werden, dies muss explizit in der Lizenz festgelegt sein.

**EVALUIERUNG:** Sollte die Lizenz des Produktes eigenständige Änderungen zulassen so kann die Höchstbewertung vergeben werden, sollte dies nicht der Fall sein so hat eine Bewertung mit null zu erfolgen.



Die rechtlichen Details der jeweiligen Lizenzen sind im Zuge einer Entscheidung hinsichtlich einer Lösung von einem Rechtsexperten **gesondert** zu beurteilen.

## 3.2 Kriterienkatalog und Bewertungsliste

Der Kriterienkatalog wurde in Anlehnung an das von Schreiber ([Schreiber2003]) propagierte strukturierte Vorgehen zur Auswahl von Informatikmitteln angelehnt.

In diesem Sinne kann der Kriterienkatalog in zwei große Bereiche unterteilt werden. Zum einen in die Definition der Kriterien, welche die Anforderungen an das System beschreiben und zum anderen die Beschreibung des entsprechenden Vorgehens zur Evaluierung und Bewertung der einzelnen Kriterien. (vgl. [Schreiber2003], S. 141ff, 181ff)

Dieses Vorgehen wurde gewählt, um eine objektive Bewertung der einzelnen Produkte zu gewährleisten. Um sicherzustellen, dass die definierten Kriterien den Anforderungen und Wünschen an das System entsprechen, sollten diese parallel zum Anforderungsprofil, welches integraler Teil des Pflichtenheftes ist, erstellt werden und diese möglichst 1:1 widerspiegeln.

Im weiteren Verlauf sind auch die Bewertungslisten so zu gestalten, dass diese aus den definierten Kriterien abgeleitet werden. Dieses Vorgehen soll sicherstellen, dass im gesamten Verlauf der Evaluierung der Produkte eine konsistente und lösungsorientierte Sichtweise beibehalten wird.

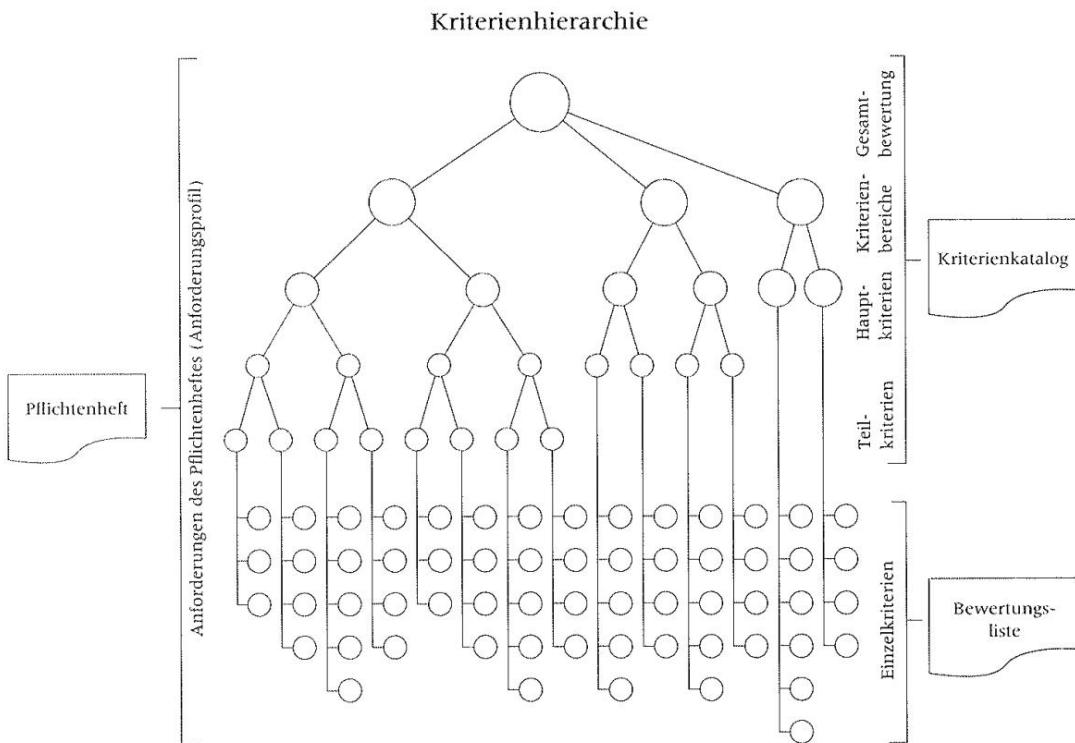


Abbildung 3.3: Zusammenhang Anforderungsprofil - Kriterienkatalog - Bewertungslisten ([Schreiber2003], S.146)

### 3.2.1 Einteilung der Kriterien

Die einzelnen Kriterien sollen soweit möglich von einander unabhängig definierbar und bewertbar sein. Dies ist insofern essentiell, um zu gewährleisten, dass die Gewichtung der Kriterien den realen Einfluss dieser auf eine Entscheidung widerspiegeln.

Kriterien sollten in diesem Sinne strukturiert und hierarchisch aufgebaut werden, um sinnvolle und auf die jeweilige Situation und Anforderung passende Kriterien zu finden. Sollten diese anhand des Pflichtenheftes definiert werden. (vgl. [Schreiber2003], S. 141ff)

Generell teilt Schreiber Kriterien in drei Gruppen ein.

**Muss-Kriterien** Diese entsprechen den im Anforderungsprofil definierten Punkten

**KO-Kriterien** Dies sind Muss-Kriterien, welche in jedem Fall zu 100% erfüllt werden müssen. Ist dies nicht der Fall, so muss eine Lösung unter allen Umständen ausgeschlossen werden.

**Wunsch-Kriterien** Diese stellen Funktionalitäten oder Leistungen zur Verfü-

gung, welche nicht unbedingt zur Erbringung der geforderten Leistung benötigt werden aber weitere Vorteile bringen. In diesem Zusammenhang muss darauf geachtet werden, dass die Anzahl der Wunschkriterien nicht überhand gewinnt und somit den eigentlichen Zweck des Systems verfälscht.

Es muss unter allen Umständen darauf geachtet werden dass eine maximale Übereinstimmung zwischen den Anforderungen und deren Erfüllung erreicht wird.

### 3.2.2 Skalierung der Kriterien

Neben der Einteilung der einzelnen Kriterien, muss auch besonderes Augenmerk darauf gelegt werden, wie die einzelnen Kriterien bewertet werden können. Es muss eine einheitliche Bewertungsgrundlage geschaffen werden.

Dies erfolgt durch die Skalierung der Kriterien. Generell lassen lassen sich drei unterschiedliche Skalierungstypen unterscheiden. (vgl. [Schreiber2003], S.162)

**Nominalskalen** beschreiben simple Ja/Nein Kriterien

**Ordinalskalen** beschreiben die Erfüllung der Erfordernisse in den Relationen größer/kleiner/gleich und legen somit nur eine Rangfolge zwischen den unterschiedlichen Alternativen fest

**Kardinalskalen** werden auch als Intervall-/Verhältnisskalen bezeichnet. Diese beschreiben die Erfüllung der Erfordernisse anhand eines numerischen Ausdrucks welcher direkt mit den Nutzwertunterschieden korreliert.

Im Sinne einer optimalen Abbildung der Realität im Rahmen der Evaluierung, ist es zweckmäßig Kardinalskalen zu verwenden. Dies bedeutet, dass sowohl nominale als auch ordinale Kriterien in Kardinalskalen umgewandelt werden müssen. Hierzu muss eine entsprechende Skalierung gefunden werden, welche nominale/ordinale Kriterien auf eine Kardinalskala abbildet. Diese muss durchgehend für alle Kriterien angewandt werden.

Im Rahmen dieser Arbeit wird die folgende Skalierung angewandt. (vgl. [Schreiber2003], S. 163)

- 4** Sehr gut, die Anforderungen werden zu 100% erfüllt
- 3** Gut, die Anforderungen werden im wesentlichen, dh mit kleineren Abstrichen erfüllt, die Anforderungen werden zu 80% erfüllt
- 2** ausreichend, die Anforderungen werden mit spürbaren Abstrichen noch erfüllt, die Anforderungen werden zu 60% erfüllt
- 1** schwach, die Anforderungen werden mit markanten Abstrichen erfüllt, die Lösung ist ohne merkliche Nachteile nicht einsetzbar, die Anforderungen werden nur zu 40% erfüllt
- 0** unbrauchbar, nicht erfüllt/vorhanden, die Anforderungen werden nur zwischen 0% und 20% erfüllt, die Lösung ist nicht einsetzbar

Nominale Kriterien müssen zwingend mit den Werten 4 bei Erfüllung bzw. Vorhandensein und 0 bei nicht Erfüllung bzw. bei fehlen der jeweiligen Funktion bewertet werden.

### 3.2.3 Gewichtung der Kriterien

Wie bereits unter Punkt 3.2 auf Seite 65 beschrieben, wird der zu erstellende Kriterienkatalog an das von Schreiber ([Schreiber2003]) beschriebene Vorgehen angelehnt.

Die Gewichtung der Kriterien hat zum Ziel den unterschiedlich hohen Einfluss eines Kriteriums bzw. einer Kriterienebene auf eine Kaufentscheidung zu berücksichtigen.

Um dies zu erreichen, wird mittels der Paarvergleichsmethode jedes Kriterium einer Ebene mit jedem anderen der selben Ebene verglichen. Es werden jeweils 10 Punkte vergeben, ist zum Beispiel Kriterium A wesentlich wichtiger als Kriterium B, so kann für A 7 Punkte und dementsprechend für B 3 Punkte vergeben werden. (vgl. [Schreiber2003], S.147-153)

Die Tabelle 3.3 auf der nächsten Seite zeigt die Gewichtung der Kriterien der Ebene eins (E1) des Kriterienkataloges. Dieses Vorgehen wird analog auf die Ebenen zwei (E2) und drei (E3) angewendet.

Kriterien	1	2	3	Punkte (absolut)	Gewicht %
1 Applikationsanforderungen		6	8	14	46,67
2 Systemplattform	4		7	11	36,67
3 Anbieterbezogene Kriterien	2	3		5	16,67
<b>Gesamt</b>				<b>30</b>	<b>100</b>

Tabelle 3.3: Paarvergleichsmethode

### 3.2.4 Berechnung der Punkte

Die einzelnen Kriterien müssen mit Punkten bewertet werden, um eine möglichst hohe Objektivität zu gewährleisten und den relativen Einfluss der jeweiligen Kriterien zu berücksichtigen, werden die Punkte anhand der Formel 3.4 berechnet.

#### Formel 3.4: Punkte

$$\text{Punkte} = g_{rel} * p_{max}$$

$p_{max}$  = maximal zu erreichende Punkte (= 4)

$g_{rel}$  = relatives Gewicht des Kriteriums

Die auf diese Weise berechneten Werte werden sowohl in den Kriterienkatalog (Spalte P) als auch in die Bewertungslisten (siehe Punkt 3.2.9 auf Seite 72) übernommen.

Die effektiv erreichten Punkte (Spalte eff. P) werden analog berechnet.

### 3.2.5 Berechnung der Nutzwerte

Der Nutzwert, bzw. das absolute Gewicht, drückt den Einfluss des jeweiligen Kriteriums bzw. der Kriterienebene auf die Nutzbarkeit des Produktes und in weiterer Folge auf eine Kaufentscheidung aus.

Werden zum Beispiel die absoluten Gewichte der Ebene drei (E3), summiert so ergibt sich daraus das absolute Gewicht der Ebene zwei (E2) (vgl. hierzu Tabelle 3.4 auf Seite 71). Die Berechnung der Nutzwerte erfolgt mittels Formel 3.5 auf der nächsten Seite.

#### Formel 3.5: Nutzwert

$$\text{Nutzwert} = \frac{E1 * E2 * E3 * \dots * En}{100 * 100 * 100 * \dots * 100} * 100$$

Wenn zum Beispiel E1 einen Wert von 46,67 und E2 einen Wert von 60 aufweisen, so ergibt sich für das Kriterium der Ebene E2 ein Nutzwert ( $= \frac{46,67 * 60}{100 * 100} * 100$ ) von 28.

Die Summe aller maximal erreichbaren Nutzwerte je Ebene ( $\sum E1, \sum E2, \dots \sum En$ ) ergibt jeweils 100%.

### 3.2.6 Berechnung der effektiven Nutzwerte

Die effektiven Nutzwerte geben darüber Auskunft, in welchem Verhältnis zum maximal erreichbaren Nutzwert das jeweilige Kriterium bewertet wurde und werden laut Formel 3.6 berechnet.

#### Formel 3.6: effektiver Nutzwert

$$\text{eff. Nutzwert} = \frac{\text{eff. } P}{P} * \text{Nutzwert}$$

Wenn zum Beispiel ein Kriterium einen maximalen Nutzwert von 10 aufweist und im Rahmen der Evaluierung mit 3( $= \text{eff. } P$ ) von 4( $= P$ ) möglichen Punkten bewertet wurde, so ergibt sich daraus ein effektiver Nutzwert ( $= \frac{3}{4} * 10$ ) von 7,5.

Je stärker sich die Summe aller effektiven Nutzwerte je Ebene ( $\sum E1, \sum E2, \dots \sum En$ ) an 100 annähert desto eher entspricht die Lösung bzw. das Produkt den Anforderungen.

### 3.2.7 Kriterienkatalog

Aus den definierten Anforderungen (siehe Punkt 3.1 auf Seite 49) wurde anhand der beschriebenen Vorgehensweise ein zur Beurteilung der zu evaluie-

renden Systeme heranzuziehender Kriterienkatalog erstellt.

Der Kriterienkatalog (siehe Tabelle 3.4) spiegelt sowohl die Anforderungen als auch die Gewichtung der einzelnen Kriterien wider.

Kriterien	Typ	rel. Gewichte [%]			NW	P
		E1	E2	E3		
<b>1 Applikationsanforderungen</b>		46,67			46,67	
<b>1.1 Funktionen/Features</b>			60		28	
1.1.1 Bearbeitungsdauer	m			18	5,04	72
1.1.2 Durchsatz	ko			28	7,84	112
1.1.3 Priorisierung	m			14	3,92	56
1.2.4 Persistierung	ko			30	8,40	120
1.1.5 Logging	w			10	2,81	40
<b>1.2 Software-Merkmale</b>			40		18,67	
1.2.1 Verfügbarkeit	m			30	5,6	120
1.2.2 Skalierbarkeit	m			22	4,11	88
1.2.3 Sprachunabhängigkeit	ko			25	4,67	100
1.2.4 Abhängigkeiten zu Fremdsystemen	w			14	2,61	56
1.2.5 Wartbarkeit	m			9	1,68	36
<b>2 Systemplattform</b>		36,67			36,67	
<b>2.1 Betriebssystem(e)</b>			90		33	
2.1.1 UNIX	ko			100	33	400
<b>2.2 Kommunikationsinfrastruktur</b>			10		3,67	
2.2.1 LAN	m			100	3,67	400
<b>3 Anbieterbezogene Kriterien</b>		16,67			16,67	
<b>3.1 Firmenmerkmale</b>			20		3,33	
3.1.1 Marktposition	w			100	3,33	400
<b>3.2 Dienstleistungen</b>			80		13,33	
3.2.1 Schulungen	w			13,33	1,78	53,33
3.2.2 Support	w			40,00	5,33	160
3.2.3 Bearbeitungsrecht	m			46,67	6,22	186,67
<b>Σ</b>		<b>100</b>			<b>2400</b>	
<b>Typ</b> ... [ko=K.O, m=Muss, w=Wunsch]- Kriterium <b>E1-E<sub>n</sub></b> ... Kriterienebene <b>P</b> ... maximal erreichbare Punkte <b>NW</b> ... Nutzwert						

Tabelle 3.4: Kriterienkatalog

### 3.2.8 KO-Kriterienliste

Alle Kriterien, welche im Kriterienkatalog als KO-Kriterien ausgewiesen wurden, werden in einer eigenen Liste dezidiert zusammengefasst. Diese KO-Kriterienliste dient im ersten Schritt der Evaluierung dazu, zu überprüfen, ob das Produkt grundlegend geeignet ist. Sollte ein Produkt nicht alle KO-Kriterien

erfüllen, so hat es für die weiteren Evaluierungsschritte auszuscheiden. Tabelle 72 zeigt die aus dem Kriterienkatalog resultierende KO-Kriterienliste. (vgl. [Schreiber2003], S.156-157)

Kriterien	erfüllt
<b>1 Applikationsanforderungen</b>	
<b>1.1 Funktionen/Features</b>	
1.1.2 Durchsatz	
1.2.4 Persistierung	
<b>1.2 Software-Merkmale</b>	
1.2.3 Sprachunabhängigkeit	
<b>2 Systemplattform</b>	
<b>2.1 Betriebssystem(e)</b>	
2.1.1 UNIX	
Erfüllt alle KO-Kriterien?	

Tabelle 3.5: KO-Kriterienliste

### 3.2.9 Bewertungsliste

Die Bewertungslisten werden im Zuge der Evaluierung herangezogen, um die einzelnen Produkte in einem einheitlichen Rahmen zu bewerten. Die Bewertungslisten stellen die unterste Ebene der Kriterienhierarchie, dh die bewertbaren Einzelkriterien, samt deren maximal erreichbaren Punkten bzw. Nutzwerten dar. Meist ist es zielführend einen entsprechenden Rahmen zB Minimal- und Maximalwerte für das jeweilige Kriterium festzuhalten. (vgl. [Schreiber2003], S.159)

Im Zuge der Bewertung der einzelnen Produkte ist es sinnvoll, zu jeder vorgenommenen Bewertung einen Kommentar bzw. eine Begründung zu verfassen. Dies dient dazu, die Transparenz der Bewertungen zusätzlich zu erhöhen. Die entsprechende Kommentar Spalte wurde hier aus Gründen der Übersichtlichkeit außen vor gelassen. In den jeweiligen Evaluierungen bzw. im Anhang ist diese jedoch aufgeführt.

Die Spalten Kriterien, Typ, G und P entsprechen jenen aus dem Kriterienkatalog, wie diese ermittelt werden können, kann unter dem Punkt 3.2 auf Seite 65ff nachgelesen werden. Die Spalte N erfasst die Bewertung des jeweiligen Kriteriums. Die effektiven Punkte (eff. P) ergeben sich in Folge aus dem Gewicht (G) und der Bewertung (N) analog zur Berechnung der Spalte P, siehe dazu Punkt 3.2.4 auf Seite 69.

Die effektiven Nutzwerte (eff. NW) werden wie unter Punkt 3.2.6 auf Seite 70 beschrieben berechnet.

Als Bewertungsliste wird die Tabelle 3.6 auf der nächsten Seite herangezogen.

### 3.2.10 Weitere Faktoren

Neben den definierten Kriterien sind diverse weiterführende Faktoren zu berücksichtigen und gesondert zu betrachten.

## Eignung

Schreiber stellt vier unterschiedliche Varianten der Eignung eines Produktes fest (siehe Abbildung 3.4). (vgl. [Schreiber2003], S.192-193)

Diese Varianten lassen sich für jedes definierte Kriterium, als auch in einer Gesamtbetrachtung auf das Produkt an sich, anwenden.

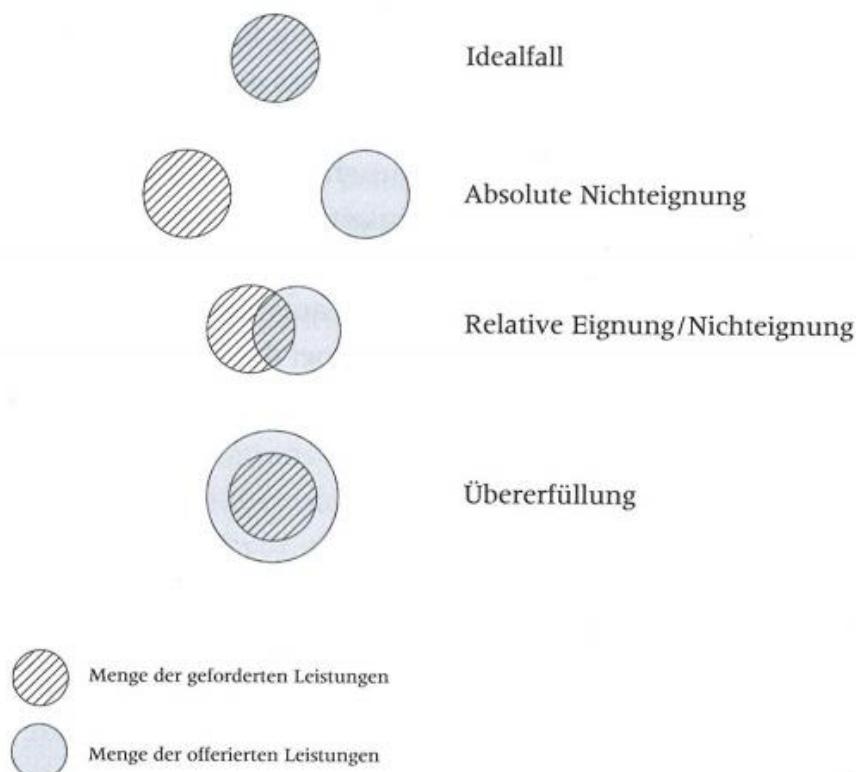


Abbildung 3.4: geforderte vs. offerierte Leistungen ([Schreiber2003], S.192)

Kriterien	Typ	G	P	Messgröße [min   max]	N	eff. P	eff. NW
<b>1 Applikationsanforderungen</b>							
<b>1.1 Funktionen/Features</b>							
1.1.1 Bearbeitungsdauer	m	18	72	min: -   max: 2ms	0	0	0
1.1.2 Durchsatz	ko	28	112	min: 500   max: -	0	0	0
1.1.3 Priorisierung	m	14	56	j/n	0	0	0
1.2.4 Persistierung	ko	30	120	j/n	0	0	0
1.1.5 Logging	w	10	40	j/n	0	0	0
<b>1.2 Software-Merkmale</b>							
1.2.1 Verfügbarkeit	ko	30	120	% lt. Herstellerangaben	0	0	0
1.2.2 Skalierbarkeit	m	22	88	Schwankungen von 1.1.1, 1.1.2	0	0	0
1.2.3 Sprachunabhängigkeit	ko	25	100	min: Java, Perl, plsql	0	0	0
1.2.4 Abhängigkeiten zu Fremdsystemen	w	14	56	min:0   max: 3	0	0	0
1.2.5 Wartbarkeit	m	9	36	min: -   max: 5h	0	0	0
<b>2 Systemplattform</b>							
<b>2.1 Betriebssystem(e)</b>							
2.1.1 UNIX	ko	100	400	min: AIX, SuSe   max: -	0	0	0
<b>2.2 Kommunikationsinfrastruktur</b>							
2.2.1 LAN	m	100	400	-	0	0	0
<b>3 Anbieterbezogene Kriterien</b>							
<b>3.1 Firmenmerkmale</b>							
3.1.1 Marktposition	w	100	400	-	0	0	0
<b>3.2 Dienstleistungen</b>							
3.2.1 Schulungen	w	13,33	53,33	j/n	0	0	0
3.2.2 Support	w	40	160	j/n	0	0	0
3.2.3 Bearbeitungsrecht	m	46,67	186,67	j/n	0	0	0
					<b>2400</b>		<b>0</b>
<b>G</b> ...Gewicht <b>eff. P</b> ...effektiv erreichte Punkte <b>eff. NW</b> ...effektiver Nutzwert							
<b>P</b> ...Soll-Punkte <b>N</b> ...Bewertung entspr. Skalierung (0-4)							

Tabelle 3.6: Template - Bewertungsliste

**Idealfall** Optimale Deckung zwischen den geforderten und den offerierten Leistungen. In diesem Fall kann die maximal zu erreichende Punktzahl vergeben werden.

**Absolute Nichteignung** Die Grenzen der akzeptablen Zielerfüllung wurden unterschritten bzw. geforderte Leistungen sind nicht vorhanden. Für diesen Fall muss eine Bewertung mit null vorgenommen werden.

**Relative Eignung/Nichteignung** Die offerierte Leistung liegt innerhalb der definierten Grenzen für das jeweilige Kriterium, erfüllt dies aber nicht zur Gänze. Die zu vergebenden Punkte schwanken im mittleren Bereich, zB zwischen eins und drei.

**Übererfüllung** Die offerierte Leistung übersteigt die geforderte Leistung, es ist zu bewerten inwiefern dies einen Einfluss auf die Gesamtbewertung des Systems hat. Steigen zum Beispiel die Kosten durch diese Übererfüllung markant an, so ist dies entsprechend negativ zu bewerten. Kann kein negativer Einfluss festgestellt werden, so kann für eine Übererfüllung gleich einem Idealfall die maximal zu erreichende Punktzahl vergeben werden.

## Risiken

Ähnlich der Skalierung der Kriterien (siehe Punkt 3.2.2 auf Seite 67) müssen auch die Risiken in einer vereinheitlichten Form erfasst werden. Die Bewertung der Risiken stellen keine konkret messbaren Faktoren dar sondern soll helfen ein Produkt und dessen Auswirkungen einzuschätzen.

Schreiber propagiert eine Einteilung der Risiken nach Risikoklassen und Risikoverursachern, wobei zwei Faktoren zu beachten sind. Zum einen die Wahrscheinlichkeit (W) mit welcher das jeweilige Risiko eintritt und zum anderen der daraus resultierende Schaden (S). Aus diesen beiden Faktoren lässt sich ein Risikofaktor (R) bestimmen, welcher eine gesamtheitliche Aussage über das bestehende Risiko zulässig macht, dieser lässt sich laut Formel 3.7 auf der nächsten Seite berechnen. (vgl. [Schreiber2003], S.213-217)

**Formel 3.7: Risiko**

$$R = (W + 1) * (S + 1) - 1$$

Die Skalen zur Beurteilung der einzelnen Risiken sind folgend definiert.

<b>Wahrscheinlichkeit (W)</b>		<b>Schaden (S)</b>	
<b>0</b>	Sehr gering, kein Risiko erkennbar	<b>0</b>	keiner
<b>1</b>	gering	<b>1</b>	gering, nur geringer Schaden bzw. geringer Aufwand zur Beseitigung
<b>2</b>	mittel	<b>2</b>	mittel, spürbarer Schaden bzw. verschmerzbarer Aufwand zur Beseitigung
<b>3</b>	hoch	<b>3</b>	hoch, hoher Schaden bzw. beachtlicher Aufwand zur Beseitigung. Es besteht eventuell eine Gefährdung der Projektes
<b>4</b>	sehr hoch	<b>4</b>	sehr hoch, auch mit hohem Aufwand nicht zu beseitigen. Dies resultiert meist in einem Abbruch des Projektes

Tabelle 3.7: Skalen zur Risikobewertung

Für die Risikobewertung wird die Tabelle 3.8 auf der nächsten Seite als Template herangezogen. Da die Auswirkungen dh der zu erwartenden Schaden (S) für alle zu evaluierenden Produkte der Selbe ist, wird diese Spalte bereits vorausgefüllt. Die Wahrscheinlichkeit (W) variiert pro Produkt, wodurch sich ein individueller Risikofaktor (R) für jedes Produkt ergibt.

Aus den einzelnen Risikofaktoren (R) lässt sich ein Gesamtrisikofaktor ( $R_g$ ) als Summe aller Einzelrisikofaktoren berechnen (siehe Formel 3.8), anhand dessen sich die einzelnen Produkte untereinander vergleichen lassen.

**Formel 3.8: Gesamtrisiko**

$$R_g = \sum R$$

Der Maximalwert für  $R_g$  liegt in diesem Fall bei 147. Um für eine bessere Vergleichbarkeit zu sorgen, wird der Gesamtrisikofaktor in eine prozentuale Dar-

Risiko	Bewertung			Kommentar
	W	S	R	
<b>technische Risiken</b>				
- Verfügbarkeit	0	3		Kann erst auf lange Sicht getestet werden
- Skalierbarkeit	0	3		Die Skalierbarkeit kann eventuell ab einem nicht festgelegten Grenzwert stark schwanken
- Durchsatz	0	4		Ein Minimaldurchsatz von 500 Nachrichten muss gewährleistet werden
- Wartbarkeit	0	2		Es könnten Probleme mit der Handhabung erst im Laufenden Betrieb festgestellt werden
<b>personelle Risiken</b>				
- fehlende Erfahrung	0	1		Die betroffenen Entwickler haben noch keinerlei Erfahrung mit diesem System
<b>Kosten-/Nutzenrisiken</b>				
- Integration	0	4		Die Integration in ein bestehendes System, kann unerwartete Auswirkungen haben
<b>Partnerrisiken</b>				
- Softwarehersteller	0	3		Hersteller kann vom Markt verschwinden bzw. das Produkt einstellen
- Plattformhersteller	0	3		große Wahrscheinlichkeit das Plattformhersteller langfristig am Markt bestehen bleiben
<b>Gesamtrisikofaktor (<math>R_g</math>)</b>		<b>23</b>		
<b>Gesamtrisikofaktor in % (<math>R_{gp}</math>)</b>		<b>0</b>		

Tabelle 3.8: Template - Risikobewertung

stellung transformiert. Hierzu wird der Minimalwert (= 23) auf den Wert 0 berichtet und der Maximalwert (= 147) auf den Wert 124.

Formel 3.9 zeigt wie die der Gesamtrisikofaktor in eine prozentuale Darstellung umgerechnet wird.

#### Formel 3.9: Gesamtrisiko (prozentuell)

$$R_{gp} = \frac{100 * (R_g - 23)}{124}$$

Die berechneten Werte für  $R_{gp}$  werden im Anschluss kaufmännisch gerundet. Erreicht ein Produkt für  $R_g$  einen Wert von 80 so ergibt sich daraus für  $R_{gp}$  ein Wert von 46%.

Liegt der Wert für  $R_{gp}$  über 60% so sollte das Produkt nicht in die engere Auswahl eingeschlossen werden, da die Risiken zu hoch werden.

## Kosten

Um ein IT System möglichst objektiv hinsichtlich der Kosten beurteilen zu können, werden die so genannten Total Costs of Ownership (TCO) herangezogen. Diese integrieren sowohl die Anschaffungskosten als auch alle weiteren während der gesamten Lebenszeit des Produktes anfallenden Kosten. Das Consulting Unternehmen Gartner hat bereits Ende der Achtziger versucht TCO umfassend formal zu definieren.

*„TCO [Total Cost of Ownership] is a holistic assessment of IT costs over time. The term holistic assessment implies an all-encompassing collection of the costs associated with IT investments, including capital investment, licence fees, leasing costs and service fees, as well as direct (budgeted) and indirect (unbudgeted) labour expenses“ ([Kirwin1987])*

Da diese Kosten im Vorraus berechnet werden, können diese Zahlen nur als Schätzwert und nicht als exakte Berechnung betrachtet werden. Wird dies vernachlässigt, kann es zu einer Scheingenaugkeit kommen. Die Berechnung der TCOs wird jedoch mit steigender Erfahrung im Unternehmen immer exakter.

Es werden immer noch viele nennenswerte Faktoren, wie zum Beispiel die Raummiete für einen Serverraum, oder ähnliches oftmals nicht in entsprechender Art und Weise berücksichtigt.

Die TCOs lassen sich generell in zwei große Gruppen einteilen (siehe Tabelle 3.9 auf der nächsten Seite), zum einen in die Investitionskosten und zum anderen in die Betriebskosten, im folgenden werden diese kurz dargestellt. (vgl. [Piedad2001], S.5-12)

Die Erfassung aller Kosten, die für ein System über einen bestimmten Zeitraum anfallen, sorgt für eine Kostentransparenz und muss dem entsprechend im Entscheidungsprozess berücksichtigt werden. (vgl. [Schreiber2003], S.206)

Tabelle 3.10 auf Seite 80 zeigt wie die Kosten übersichtlich, aufgeschlüsselt nach Investitions-/Betriebskosten sowie interne/externe Kosten, erfasst werden können.

Investitionskosten	
<b>Produktsuche</b>	Ermittlung welche Produkte am Markt verfügbar sind
<b>Bewerten der Anbieter</b>	Das beste Angebot ermitteln, hierzu dient unter anderem der Kriterienkatalog und die darauf aufbauenden Evaluierungen
<b>Anschaffung</b>	Kaufen des gewählten Produktes
<b>Installieren</b>	Installieren des Systems und integrieren in die bestehende Systemlandschaft
<b>Anpassen</b>	Anpassungen und Erweiterungen durchführen
<b>Trainings und Schulungen</b>	Einschulen der Mitarbeiter auf das neue System
Betriebskosten	
<b>System Management</b>	Alle Tätigkeiten die zur Sicherstellung der einwandfreien Funktion des Systems notwendig sind, sowie Backup und Recovery Tätigkeiten
<b>Hard-/Softwarewartung</b>	Präventive Wartung der Hard- und Software sowie die schnelle Korrektur auftretender Fehler
<b>Support</b>	Alle Tätigkeiten die darauf abzielen die Benutzer bei der Verwendung des Systems zu unterstützen, zB Helpdesk, Trainings und Schulungen.
<b>Umwelteinflüsse</b>	Zum Beispiel Stromkosten oder Kosten für die Klimatisierung des Serverraumes

Tabelle 3.9: Beispiele zu Investitions- und Betriebskosten

Die internen Kosten wurden durch das Unternehmen festgelegt und bleiben für alle Produkte konstant. In diesem Sinne sind somit lediglich die externen Kosten ausschlaggebend, um jedoch die internen Kosten nicht zu vernachlässigen, werden diese aufgeführt.

	<b>Kosten</b>	<b>Kommentar</b>	
<b>Investitionskosten</b>			
<b>extern</b>			
Kaufpreis	0	Lizenz	einmalig
Schulungen	0	Grundlegende Einschulungen in das System	
<b>intern</b>			
Integration	90000	Anpassen der aktuellen Systeme 1500h á 60 €	einmalig
Test(s)	30000	Testen des neuen Systems nach der Integration 500h á 60 €	
<b>Investitionskosten gesamt</b>	<b>120 000</b>		
<b>Betriebskosten</b>			
<b>extern</b>			
Support	0		laufend
<b>intern</b>			
Problem & Change Management	1200	20h Jahr á 60 €	laufend
Administration & Wartung	3000	50h Jahr á 60 €	
<b>Betriebskosten gesamt</b>	<b>4 200</b>		
<b>Gesamtkosten [ Jahr]</b>	<b>124 200</b>		

Tabelle 3.10: Total Costs of Ownership

## 3.3 Evaluierung

Wie bereits unter Punkt 3 auf Seite 47, beschrieben soll ein einheitlicher Evaluierungsprozess eine objektive Beurteilung der Produkte ermöglichen. Die folgenden Punkte *Marktanalyse*( auf dieser Seite), *Grobevaluierung*( auf der nächsten Seite) und *Detailevaluierung*( auf Seite 100) sowie die *Gegenüberstellung der Produkte*( auf Seite 113) wenden das beschriebene Vorgehen an konkreten Produkten an.

### 3.3.1 Marktanalyse

*„Da sich die Anwendungsgebiete für Software erweitern und der Bedarf nach verteilten Lösungen steigt, gewinnt der Einsatz von Middleware perspektivisch mehr an Bedeutung“ ([Reinecke2004])*

In diesem Sinne stieg auch die Anzahl der angeboten Lösungen im Bereich der Middleware im Laufe der Jahre stark an. Nicht nur proprietäre bzw. kostenpflichtige Lösungen, sondern auch vermehrt Open Source und kostenlose Lösungen stehen zur Auswahl.

Im folgenden sollen jene Produkte kurz beschrieben werden, welche für eine Evaluierung ausgewählt wurden. Die Auswahl der Produkte wurde durch

das Unternehmen vorgenommen und basiert auf den versprochenen Features und Leistungen der jeweiligen Produkte.

Neben den gewählten Produkten gibt es eine Vielzahl an weiteren Produkten und Lösungen. Entsprechende Aufstellungen finden sich zum Beispiel unter [http://en.wikipedia.org/wiki/Java\\_Message\\_Service#Provider\\_implementations](http://en.wikipedia.org/wiki/Java_Message_Service#Provider_implementations) und [http://de.wikipedia.org/wiki/Java\\_Message\\_Service\\_Provider](http://de.wikipedia.org/wiki/Java_Message_Service_Provider).

Tabelle 3.11 zeigt welche Produkte im Rahmen dieser Diplomarbeit evaluiert wurden.

Open Source / kostenlos	Proprietär / kostenpflichtig
Apache ActiveMQ <sup>a</sup>	IBM WebsphereMQ <sup>b</sup>
Sun OpenMQ <sup>c</sup>	Oracle Advanced Queuing <sup>d</sup>
JBoss Messaging <sup>e</sup>	Progress SonicMQ <sup>f</sup>
	Fiorano FioranoMQ <sup>g</sup>

<sup>a</sup><http://activemq.apache.org/>

<sup>b</sup><http://www-01.ibm.com/software/integration/wmq/>

<sup>c</sup><https://mq.dev.java.net/>

<sup>d</sup>[http://download.oracle.com/docs/cd/B19306\\_01/appdev.102/b14294/queuing.htm](http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14294/queuing.htm)

<sup>e</sup><http://www.jboss.org/jbossmessaging/>

<sup>f</sup><http://www.sonicsoftware.com/products/sonicmq/index.ssp>

<sup>g</sup>[http://www.fiorano.com/products/fmq/products\\_fioranofmq.php](http://www.fiorano.com/products/fmq/products_fioranofmq.php)

Tabelle 3.11: Middleware Lösungen / Produkte

### 3.3.2 Grobevaluierung

Die Grobevaluierung der ausgewählten Produkte dient in erster Linie dazu, einen objektiven Überblick zu generieren und weiters dazu zu überprüfen ob die Produkte grundlegend geeignet sind. (vgl. [Schreiber2003], S.157, 221)

Um die grundlegende Eignung eines Produktes nachzuweisen bzw. zu widerlegen, kommt die KO-Kriterienliste (siehe Punkt 3.2.8 auf Seite 71) zum Einsatz. Werden alle Punkte erfüllt, so kann das Produkt einer Detailevaluierung unterzogen werden. Werden jedoch nicht alle KO-Kriterien erfüllt so muss das Produkt aus dem weiteren Evaluierungs- und Entscheidungsprozess ausgeschlossen werden.

## Apache ActiveMQ

Apache ActiveMQ bietet einen vollwertigen JMS Provider, der neben der einfachen Installation und Administration (siehe *Installation und Administration* auf Seite 82) auch eine hervorragende Leistung in allen geforderten Bereichen aufweist.

Als Beispiel sei hier der Durchsatz erwähnt, welcher bei allen Tests jenseits der geforderten 1000 Nachrichten in der Sekunde lag (vgl. Abbildung 3.5).

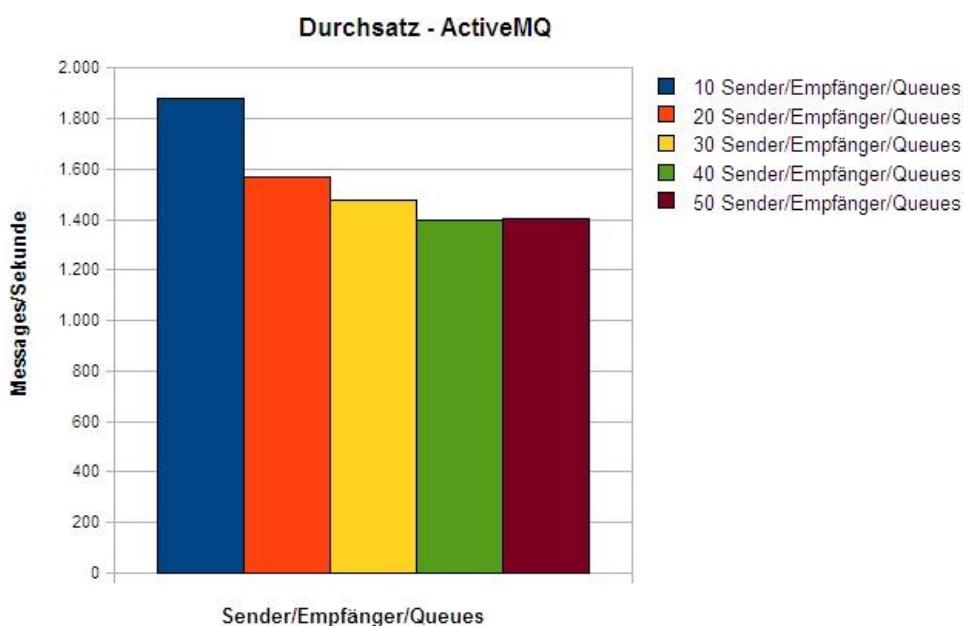


Abbildung 3.5: Apache ActiveMQ - Durchsatz

**Installation und Administration** Tabelle 3.12 auf der nächsten Seite zeigt die von Apache verifizierten System Anforderungen.

Die Installation und das Starten von Apache ActiveMQ lässt sich in wenigen Minuten bewerkstelligen. Nachdem die Systemvariablen (JAVA\_HOME, ANT\_HOME und PATH) entsprechend adaptiert wurden genügt es, über die Konsole das Startskript (bin/activemq) auszuführen, um den JMS Provider zu starten. Wird die Prozessumgebung (zB Konsole) beendet, so wird auch ActiveMQ beendet. Um dies zu verhindern, kann ActiveMQ auch im Hintergrund ausgeführt werden. Dazu muss ein „&“ an den Programmaufruf angefügt werden (`./activemq &`).

Component	Requirements
<b>Operating Systems</b>	
UNIX	Ubuntu Linux Powerdog Linux MacOS AIX HP-UX Solaris or any Unix platform that supports Java
Windows	Windows XP SP2 Windows 2000
<b>Software</b>	
Java Development Kit	>1.5.x
<b>Hardware</b>	
CPU	no information provided
RAM	no information provided
Disk space	~40MB

Tabelle 3.12: System Requirements - Apache ActiveMQ

1. Set **JAVA\_HOME** to point to your JDK installation directory
2. Set **ANT\_HOME** to point to your ANT installation directory
3. Add **JAVA\_HOME** and **ANT\_HOME** to **PATH**
4. go to the directory where you have extracted ActiveMQ
5. cd bin
6. **./activemq**

Die Administration von ActiveMQ kann vollständig über das Webinterface (vgl. Abbildung 3.6 auf der nächsten Seite), welches unter <http://localhost:8161/admin> erreichbar ist, vorgenommen werden.

Neben den administrativen Aufgaben (zB anlegen von neuen Queues), können die einzelnen Queues und die beinhalteten Nachrichten auch ins Detail überwacht und analysiert werden. Durch den intuitiven Aufbau des Webinterfaces, lassen sich sehr schnell alle wichtigen Tätigkeiten erlernen und ausführen.

The screenshot shows the Apache ActiveMQ web interface. At the top, there's a navigation bar with links for Home, Queues, Topics, Subscribers, and Send. Below the navigation is a search bar labeled 'Queue Name' with a 'Create' button. The main area is titled 'Queues' and contains a table with two rows of data. The columns are: Name, Number Of Pending Messages, Number Of Consumers, Messages Sent, Messages Received, Views, and Operations. The first row has 'qname-0' in the Name column, 16 in Number Of Pending Messages, 1 in Number Of Consumers, 1087 in Messages Sent, 1071 in Messages Received, and 'Browse' and 'Send To Purge' buttons in the Views and Operations columns respectively. The second row has 'qname-1' in the Name column, 1 in Number Of Pending Messages, 1 in Number Of Consumers, 1103 in Messages Sent, 1102 in Messages Received, and 'Browse' and 'Send To Purge' buttons in the Views and Operations columns respectively.

Name	Number Of Pending Messages	Number Of Consumers	Messages Sent	Messages Received	Views	Operations
qname-0	16	1	1087	1071	Browse	Send To Purge Delete
qname-1	1	1	1103	1102	Browse	Send To Purge Delete

Abbildung 3.6: Apache ActiveMQ - Webinterface

**Fazit** ActiveMQ kann durchwegs positiv beurteilt werden und wird einer Detailevaluierung unterzogen.

## Sun OpenMQ

Bei OpenMQ handelt es sich um eine frei verfügbare Version der Sun Message Queue, welche als proprietäres Produkt angeboten wird.

OpenMQ bietet einen vollwertigen JMS Provider. Als generelles Fazit lässt sich sagen, dass OpenMQ sowohl in der Performance als auch in der Handhabung sehr gut bis ausgezeichnet abschneidet. Die Performance im Bezug auf den Durchsatz lag im Mittel über 1400 Nachrichten in der Sekunde (vgl. Abbildung 3.7 auf der nächsten Seite) und somit deutlich über dem geforderten Minimalwert.

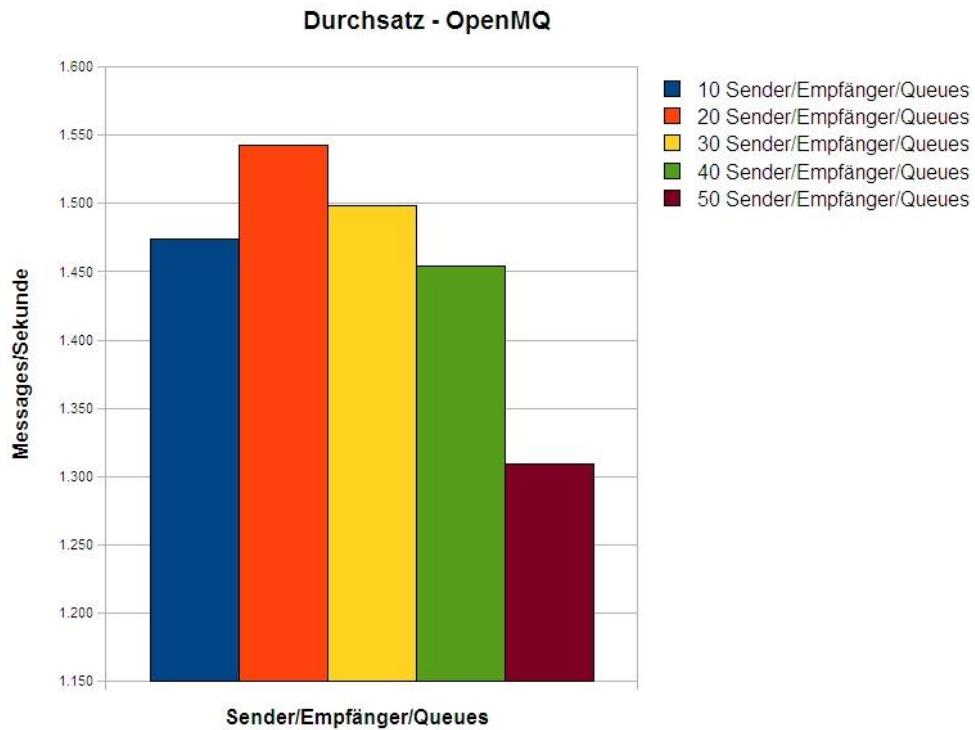


Abbildung 3.7: Sun OpenMQ- Durchsatz

**Installation und Administration** Tabelle 3.13 zeigt die von Sun verifizierten System Anforderungen.

Component	Requirements
<b>Operating Systems</b>	
Solaris	Solaris 9, 10
Linux	Red Hat Enterprise Linux Advanced Server 3.0, 4.0, 5.0 Red Hat Enterprise Linux Enterprise Server 3.0, 4.0, 5.0
Windows	Windows Vista Windows XP Professional SP2 Windows 2000 Advanced Server SP4 Windows Server 2003 Standard and Enterprise Editions SP2
IBM AIX	AIX 6.1
<b>Software</b>	
Java Runtime Environment	>1.5.0_15
Java Development Kit	>1.5.0_15
<b>Hardware</b>	
CPU	>Intel Pentium 2
RAM	>256MB
Disk space	installation (.zip) file ~150MB temporary working/installation directory ~220MB Installed product (omitting shared components and local message store) ~22MB

Tabelle 3.13: System Requirements - Sun OpenMQ

OpenMQ konnte im Laufe der Evaluierung nicht über den empfohlenen

grafischen Installer am Testsystem installiert werden. Die Installation brach mit der folgenden Meldung ab.

**Listing 3.1: Error Sun Installer**

```
1 ./installer: line 54: dirname: command not found
2 Exception in thread "main" java.lang.NoClassDefFoundError: \n
   ↗MkTemp
3 Caused by: java.lang.ClassNotFoundException: MkTemp
4  at java.net.URLClassLoader\$1.run(URLClassLoader.java:200)
5  at java.security.AccessController.doPrivileged(Native Method)
6  at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
7  at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
8  at sun.misc.Launcher\$AppClassLoader.loadClass(Launcher.java\n
   ↗:301)
9  at java.lang.ClassLoader.loadClass(ClassLoader.java:252)
10 at java.lang.ClassLoader.loadClassInternal(ClassLoader.java\n
    ↗:320)
11 Could not find the main class: MkTemp.
12 Program will exit. Unable to create temporary file, exiting
```

Offiziell gibt es von Sun keine für SuSe verifizierte Version, es wird jedoch eine generische Unix Version angeboten, welche manuell installiert werden konnte. Hierzu wurde wie folgt vorgegangen.

1. Unzip the binary ([http://download.java.net/mq/open-mq/4.3/b07/mq4\\_3-binary-Unix-20081108.jar](http://download.java.net/mq/open-mq/4.3/b07/mq4_3-binary-Unix-20081108.jar)) bundle to a folder of your choice (further referenced as \$TMP).
2. Edit the file **\$TMP/mq/etc/imqenv.conf** to set **IMQ\_DEFAULT\_JAVAHOME** to a JDK1.5.0. or above
3. Change directory to **\$TMP/mq/bin**
4. Start the broker: **./imqbrokerd -tty**

OpenMQ bietet eine eigene Management Console (vgl. Abbildung 3.8 auf der nächsten Seite), welche sowohl Monitoring als auch administrative Aufgaben unterstützt. Die übersichtliche Gestaltung des User Interfaces erlaubt eine schnelle Einarbeitung und ermöglicht somit OpenMQ schnell und unkompliziert an die geforderten Bedingungen anzupassen.

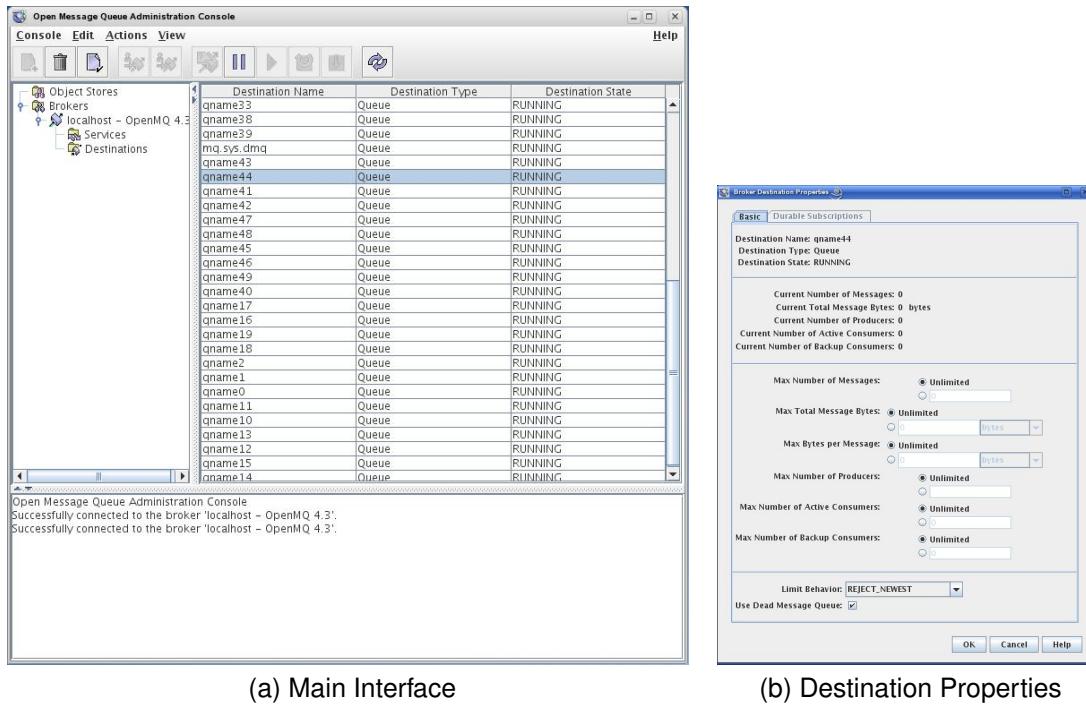


Abbildung 3.8: Sun OpenMQ Administration Interface

**Fazit** OpenMQ kann durchwegs positiv beurteilt werden und wird einer Detailevaluierung unterzogen.

## JBoss Messaging

JBoss gehört zum bekannten Softwarehersteller Red Hat. Die neueste Version des JBoss Messagings wird als Standard Messaging Provider für den JBoss Application Server Version 5 verwendet. Sowohl der JBoss Application Server<sup>8</sup> als auch JBoss Messaging<sup>9</sup> stehen zum freien Download zur Verfügung.

**Installation und Administration** Tabelle 3.14 auf der nächsten Seite zeigt die von JBoss verifizierten System Anforderungen.

Die Administration von JBoss Messaging kann ähnlich wie bei Apache ActiveMQ über ein Web Interface (siehe Abbildung 3.9 auf der nächsten Seite) vorgenommen werden. Dieses ist jedoch wesentlich umfangreicher und unübersichtlicher. Dies leitet sich aus dem Umstand ab, dass das Web Interface

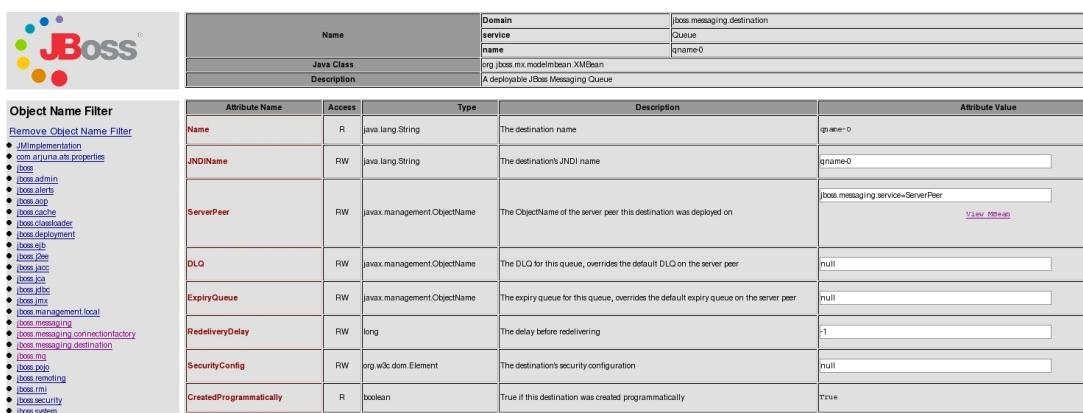
<sup>8</sup><http://www.jboss.org/jbossas/downloads/>

<sup>9</sup><http://jboss.org/jbossmessaging/>

Component	Requirements
<b>Operating Systems</b>	
UNIX	any Unix platform that supports Java
Windows	any Windows platform that supports Java
<b>Software</b>	
Java Development Kit	>1.5.x >1.6.x
<b>Hardware</b>	
CPU	no information provided
RAM	no information provided
Disk space	extracted zip file size ~130MB

Tabelle 3.14: System Requirements - JBoss Messaging

zur allgemeinen Administration des JBoss Application Servers gedacht ist und somit nicht auf die Aufgaben des Messagings optimiert wurde.



The screenshot shows the JBoss Seam Administration interface. On the left, there is a sidebar with a tree view of object names, including categories like 'Remove Object Name Filter', 'JNDIName', 'ServerPeer', 'DLO', 'ExpiryQueue', 'RedeliveryDelay', 'SecurityConfig', and 'CreatedProgrammatically'. The main panel displays a table with the following columns: Name, Domain, Type, Description, and Attribute Value. The table contains the following data:

Name	Domain	Type	Description	Attribute Value
service	jboss.messaging.destination	Queue		
name	qname-0			
Java Class	org.jboss.mq.modelimplbean.XMBean			
Description	A deployable JBoss Messaging Queue			
Object Name Filter				
Remove Object Name Filter				
● <a href="#">jboss-implementation</a>				
● <a href="#">com.arunava.ws.property</a>				
● <a href="#">boss</a>				
● <a href="#">boss-admin</a>				
● <a href="#">boss-alerts</a>				
● <a href="#">boss-beans</a>				
● <a href="#">boss-cache</a>				
● <a href="#">boss-classloader</a>				
● <a href="#">boss-deployment</a>				
● <a href="#">boss-ejb</a>				
● <a href="#">boss-ejb3</a>				
● <a href="#">boss-jca</a>				
● <a href="#">boss-jbc</a>				
● <a href="#">boss-jms</a>				
● <a href="#">boss-jmx</a>				
● <a href="#">boss-management-local</a>				
● <a href="#">boss-messaging</a>				
● <a href="#">boss-messaging-connectionfactory</a>				
● <a href="#">boss-messaging-destination</a>				
● <a href="#">boss-mq</a>				
● <a href="#">boss-pool</a>				
● <a href="#">boss-remoting</a>				
● <a href="#">boss-ri</a>				
● <a href="#">boss-security</a>				
● <a href="#">boss-system</a>				

Abbildung 3.9: JBoss Messaging - Administration

**Fazit** Leider kam im Laufe der Tests zu Tage, dass nicht alle Nachrichten, welche an den JBoss Messaging Provider gesendet wurden auch korrekt zugestellt wurden. Es kam zu der Situation, dass Nachrichten korrekt vom JBoss Messaging Provider angenommen aber nicht an den Empfänger weitergeleitet wurden (siehe Abbildung 3.10).

Ein entsprechender Bug Report wurde an die Entwickler von JBoss Messaging übermittelt (siehe <sup>10</sup>).

<b>MessageCount</b>	R	int	The number of messages in the queue	3
<b>DeliveringCount</b>	R	int	The number of messages currently being delivered	3

Abbildung 3.10: JBoss Messaging - Fehlverhalten

<sup>10</sup><https://jira.jboss.org/jira/browse/JBMESSAGING-1684>



Aufgrund des beschriebenen Fehlverhaltens von JBoss Messaging konnten die Tests nicht erfolgreich abgeschlossen werden. Da somit die Minimalanforderungen nicht erfüllt werden muss JBoss Messaging aus der weiteren Betrachtung ausscheiden.

## IBM WebsphereMQ

IBM WebsphereMQ ist das Nachfolgeprodukt der Websphere MQSeries und aktuell in der Version 7 verfügbar.

**Installation und Administration** Tabelle 3.15 zeigt die von IBM verifizierten System-Anforderungen. Eine detaillierte Aufstellung aller Systemanforderungen findet sich unter <http://www-01.ibm.com/software/integration/wmq/requirements/#WebSphereMQV70>.

Component	Requirements
<b>Operating Systems</b>	
AIX	AIX 5.3 plus TL04 and appropriate Firmware, 6.1
HP-UX	HP-UX 11i V2, V3
Linux	Red Hat Enterprise Linux v4.0, v5.0 SuSe Linux Enterprise Server v9, v10
Solaris	Solaris v9, v10
Windows	Windows XP SP2 Windows Server 2003 SP1 Windows Server 2008 Windows Vista (Business, Enterprise or Ultimate)
<b>Software</b>	
Java Development Kit	>1.5.x
<b>Hardware</b>	
CPU	no information provided
RAM	no information provided
Disk space	extracted installation zip file ~550MB

Tabelle 3.15: System Requirements - IBM Websphere

Die Installation erfolgt über die Kommandozeile und erfordert grundlegende Linux Kenntnisse im Bereich der Verwendung von RPM Packages. Eine kurze Anleitung/Beschreibung des Installationsvorganges findet sich unter [ibm.com](http://www.ibm.com/developerworks/websphere/library/techarticles/0705_salkosuo/0705_salkosuo.html)<sup>11</sup>.

<sup>11</sup>[http://www.ibm.com/developerworks/websphere/library/techarticles/0705\\_salkosuo/0705\\_salkosuo.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0705_salkosuo/0705_salkosuo.html)

Die wesentlichen Schritte werden in Listing 3.2 auf der nächsten Seite dargestellt.

**Listing 3.2: Installation IBM WebsphereMQ**

```
1 #Installation procedure
2 # - First extract the archive to any folder of your choice
3 # - run mqlicense.sh to read and accept the licence
4 # - install needed packages [minimum]
5
6 rpm -ivh MQSeriesRuntime-7.0.0-0.i386.rpm
7 rpm -ivh MQSeriesServer-7.0.0-0.i386.rpm
8
9 # after these steps a user account called 'mqm' exists which is \
   →used for administrative tasks - you may need to update the \
   →rights granted to user 'mqm'
10
11 # log in as 'mqm'
12 # - change directory to /opt/mqm/bin
13 cd /opt/mqm/bin
14
15 # - create a Queue Manager named 'MQServer'
16 crtmqm MQServer
17
18 # - start the Queue Manager
19 strmqm MQServer
20
21 # - create a MQServer.conf file where you define the queues - \
   →the file could look like this
22 # DEFINE QLOCAL('qname0') + REPLACE
23 # DEFINE QLOCAL('qname1') + REPLACE
24
25 # use the MQServer.conf file to create the Queues - the output \
   →will be written to MQServer-conf.log
26 runmqsc MQServer < MQServer.conf > MQServer-conf.log
27
28 # - start the command server
29 strmqcsv MQServer &
30
31 # - start server listener - by default port 1414 is used
32 runmqqlsr -m MQServer -t TCP &
33
34 # verify that everything is working
35 netstat -an | grep 1414
```

Die Administration von IBM WebsphereMQ kann über den mitgelieferten MQ Explorer erfolgen. Dieser basiert auf Eclipse und stellt alle benötigten Funktionen bereit. Abbildung 3.11 auf der nächsten Seite zeigt die Standardansicht des MQ Explorers.

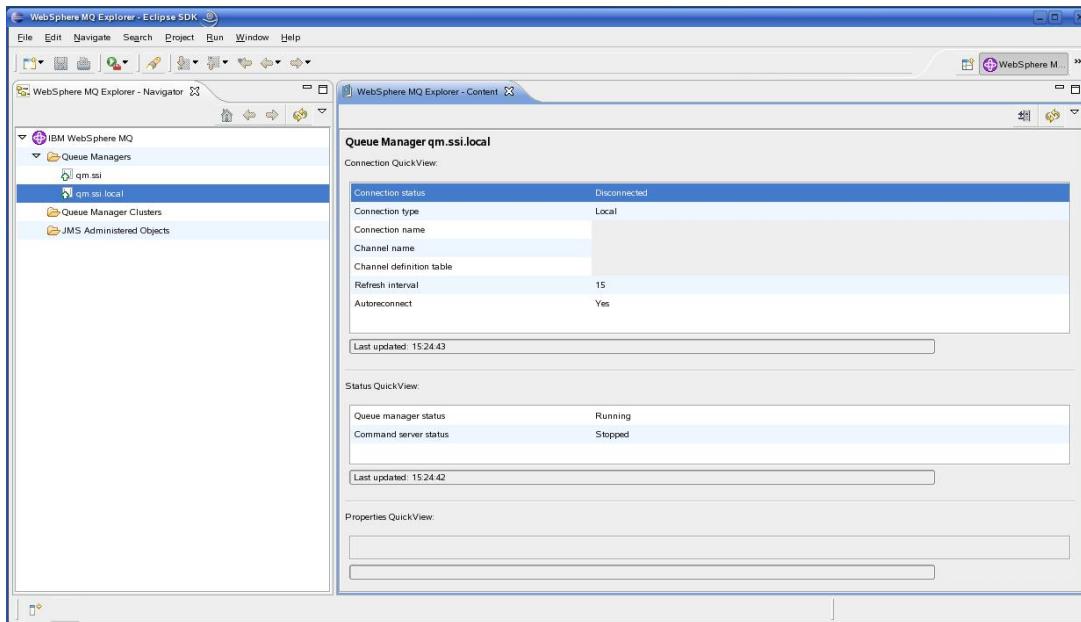


Abbildung 3.11: IBM WebsphereMQ - Administration

**Fazit** Im Laufe der Tests zeigte sich, dass IBM WebsphereMQ nicht die geforderte Leistung erbringt. Die Anzahl der Nachrichten die pro Sekunde übertragen werden konnten lag zwischen 1-2, was deutlich unter den geforderten 500 Nachrichten pro Sekunde liegt.

Dieses Verhalten wurde auch in anderen Benchmarks bestätigt, so wurde zum Beispiel durch die Crimson Consulting Group im Jahr 2003 ([Crimson2003]) festgestellt, dass IBM Websphere bei persistentem Messaging nur einen Durchsatz von rund 55 Nachrichten pro Sekunde schafft. Die Crimson Consulting Group verwendete WebsphereMQ in der Version 5.3.

Die im Rahmen dieser Arbeit durchgeführten Tests zeichnen ein ähnliches Bild für die aktuelle WebsphereMQ Version.



Da der Durchsatz nicht die geforderten Minimalkriterien erfüllt muss IBM WebsphereMQ aus der weiteren Betrachtung ausgeschlossen werden.

## Oracle Advanced Queuing

**Installation und Administration** Tabelle 3.16 auf der nächsten Seite zeigt die von Oracle verifizierten System-Anforderungen.

Component	Requirements
<b>Operating Systems</b>	
UNIX	Apple Mac OS X Server HP-UX HP Tru64 UNIX HP OpenVMS IBM AIX5L IBM z/OS Linux: x86, x86-64 Sun Solaris: x86, x86-64
Windows	Microsoft Windows: x86, x86-64
<b>Software</b>	
Depends on the platform used, a Unix host must have several Unix packages installed. For example: <b>make, gcc, glibc</b> for detailed information consult the corresponding Readme files. <a href="http://www.oracle.com/pls/db102/portal.all_books">http://www.oracle.com/pls/db102/portal.all_books</a>	
<b>Hardware</b>	
CPU	no information provided
RAM	>1024MB
Disk space	~1.5GB - 3.5GB

Tabelle 3.16: System Requirements - Oracle Advanced Queuing

Oracle Advanced Queuing (AQ) bietet einen in die Datenbank integrierten JMS Provider. Dieser wird standardmäßig mit jeder Version einer Oracle Datenbank kostenlos mitgeliefert und ist nach erfolgreicher Konfiguration (vgl. Listing 3.3) sofort einsatzbereit.

**Listing 3.3: Installation/Konfiguration - Oracle AQ**

```

1
2  create role ssi_aq_adm_role; /* create AQ Administrator Role */
3  grant connect, resource, aq_administrator_role to <
   -ssi_aq_adm_role; /* grant rights */
4  create role ssi_aq_user_role; /* create AQ User Role */
5  grant create session, aq_user_role to ssi_aq_user_role; /* grant<
   - rights */
6  create user ssi_aquser identified by ssi_aquser; /* create AQ <
   -User */
7  create user ssi_aqadm identified by ssi_aqadm; /* create AQ <
   -Admin */
8
9  grant ssi_aq_adm_role to ssi_aqadm; /* grant rights */
10 grant ssi_aq_user_role to ssi_aquser; /* grant rights */
11
12 exec DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
13     privilege => 'ENQUEUE_ANY',
14     grantee => 'ssi_aqadm',
15     admin_option => FALSE);
16
17 exec DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
18     privilege => 'DEQUEUE_ANY',
19     grantee => 'ssi_aqadm',
20     admin_option => FALSE);

```

```
21
22  exec DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
23      privilege => 'MANAGE_ANY',
24      grantee => 'ssi_aqadm',
25      admin_option => FALSE);
26
27  exec DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
28      privilege => 'ENQUEUE_ANY',
29      grantee => 'ssi_aquser',
30      admin_option => FALSE);
31
32  exec DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
33      privilege => 'DEQUEUE_ANY',
34      grantee => 'ssi_aquser',
35      admin_option => FALSE);
36
37  /* set number of processes that should be created for dealing \
   -with messaging issues */
38 ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 10; commit;
39 ALTER SYSTEM SET AQ_TM_PROCESSES=10; commit;
40
41 /* logged in as ssi_aqadm */
42
43 /* create the table where the messages will be persisted */
44 exec DBMS_AQADM.CREATE_QUEUE_TABLE (
45     QUEUE_TABLE => 'SYSTEM.QTable_qname_0',
46     QUEUE_PAYLOAD_TYPE => 'SYS.AQ$_JMS_TEXT_MESSAGE',
47     MULTIPLE_CONSUMERS => FALSE);
48
49 /* create the message queue itself, and tell the system which \
   -table to use */
50 exec DBMS_AQADM.CREATE_QUEUE (
51     QUEUE_NAME => 'qname_0',
52     QUEUE_TABLE => 'SYSTEM.QTable_qname_0');
53
54 /* start the message queue */
55 exec DBMS_AQADM.START_QUEUE (QUEUE_NAME => 'SYSTEM.qname_0');
```

**Fazit** Im Rahmen der Test zeigte sich jedoch, dass der Durchsatz von Oracle AQ weit unter den minimal geforderten 500 Nachrichten pro Sekunde liegt. Die Zustellung von 10.000 Nachrichten benötigte rund 84 Minuten, was einem Durchsatz von rund 2 Nachrichten pro Sekunde entspricht.

Tests diverser Computer- und Beratungsunternehmen zeigen auf wesentlich stärkeren Testsystemen eine proportional gesehen ähnliche Performance. So zeigt zum Beispiel Dell in seinem Testszenario einen durchschnittlichen Durchsatz zwischen <200 und <550 Nachrichten pro Sekunde. (vgl. [Dell2007], S.87)

Zu ähnlichen Ergebnissen kommt auch ein Test von SUN, welcher die

Performance von Oracle AQ auf unterschiedlichen Serversystemen betrachtet. Hierbei zeigt sich deutlich, dass der Durchsatz stark von der Leistungsfähigkeit des Servers abhängig ist. Der kleinste Durchsatz liegt in diesem Fall bei 16 Nachrichten pro Sekunde und der höchste bei 302 Nachrichten pro Sekunde.

Ab der Version 10g Rel.2 bietet Oracle die Möglichkeit des „Buffered Messaging“. In diesem Fall werden die Messages nicht in die Datenbank persistiert, sondern nur im Speicher gehalten. Sollte der Speicher vollständig belegt sein, werden die Messages in die Datenbank geschrieben. Buffered Messaging ist wesentlich schneller als persistentes Messaging, hat jedoch den Nachteil, dass Messages verloren gehen können (zB wenn die Datenbank neu gestartet wird). Weitere Informationen zu Oracle Advanced Queuing in Verwendung mit JMS finden sich unter [http://download-west.oracle.com/docs/cd/B19306\\_01/server.102/b14257/jm\\_create.htm#CIAEDDAE](http://download-west.oracle.com/docs/cd/B19306_01/server.102/b14257/jm_create.htm#CIAEDDAE)



Da der minimal geforderte Durchsatz nicht erreicht wurde, wird Oracle AQ im Weiteren nicht mehr berücksichtigt.

## Progress SonicMQ

Progress SonicMQ ist ein proprietäres Produkt, welches sich durch die einfache Handhabung und die herausragende Performance (vgl. Abbildung 3.12) auszeichnet. Weiters werden von Progress umfassende Dokumentationen und Beispiele bereitgestellt, welche die Einarbeitungszeit auf ein Minimum reduzieren.

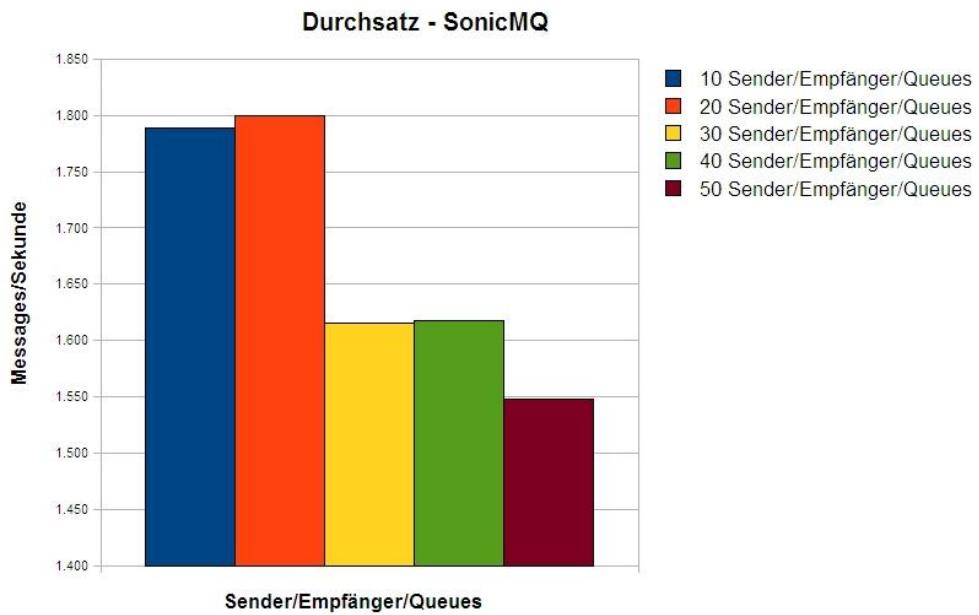


Abbildung 3.12: Progress SonicMQ - Durchsatz

**Installation und Administration** Tabelle 3.17 zeigt die von Progress verifizierten System Anforderungen.

Component	Requirements
<b>Operating Systems</b>	
UNIX	Sun Solaris 9, 10 Red Hat 4 Update 6, 5 Update 2 SuSe 9 SP 3, 10 SP 2 AIX 5L V5.2, 5L V5.3 HP-UX 11i V2, 11i V3
Windows	MS Vista Business SP1 MS XP Pro SP 3 Win Server 2003 SP2
<b>Software</b>	
JVM	>1.4.2
<b>Hardware</b>	
CPU	no information provided
RAM	no information provided
Disk space	no information provided

Tabelle 3.17: System Requirements - Progress SonicMQ

Die Installation von SonicMQ lässt sich über einen grafischen Wizard erledigen (siehe Abbildung 3.13 auf der nächsten Seite). Um die Installation zu starten, muss das Archiv (SonicMQ7.6.tar) in einen beliebigen Ordner extrahiert und das Setup Skript (setup.sh) ausgeführt werden.

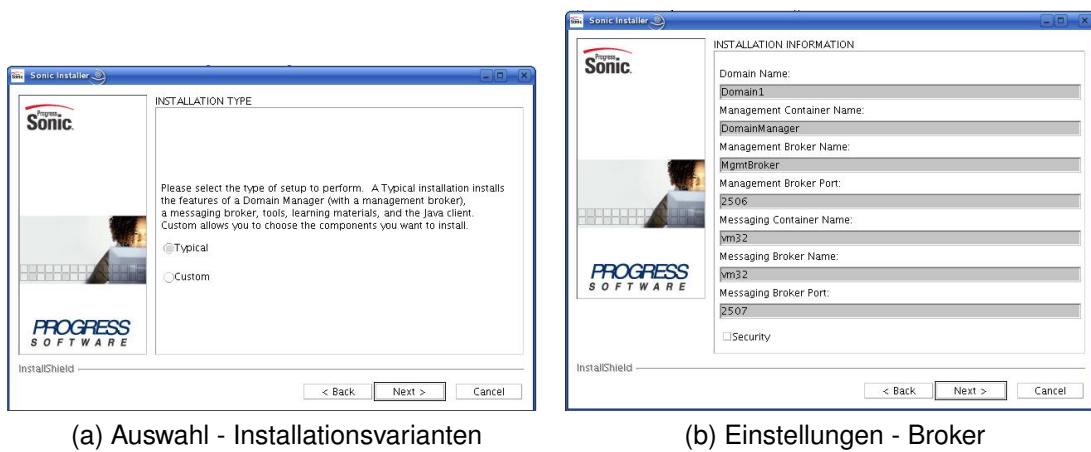


Abbildung 3.13: Progress SonicMQ - Installation

Der SonicMQ JMS Provider lässt sich nach erfolgreicher Installation über das Start Skript (startcontainer.sh), welches sich im gewählten Installationspfad (zB /opt/Sonic/bin) befindet, starten.

SonicMQ bietet ein leistungsstarkes Interfaces zur Administration und zum Monitoring (siehe Abbildung 3.14), welches durch Aufrufen des entsprechenden Skripts (zB startmc.sh) gestartet werden kann.

Neben den administrativen Aufgaben verfügt das Administrations Interface auch über einen eigenen JMS Test Client (siehe Abbildung 3.14b), welcher dazu eingesetzt werden kann die Funktionalität der JMS Destinations (Queues/Topics) zu testen.

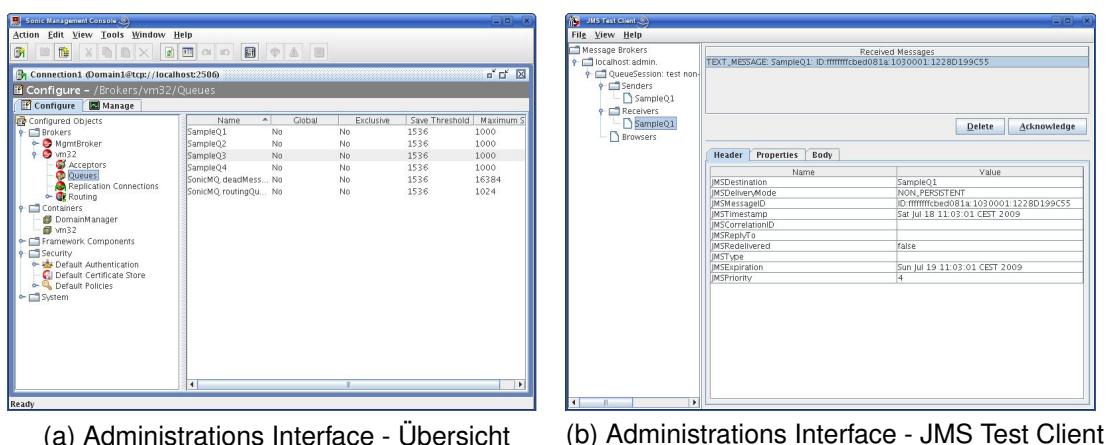


Abbildung 3.14: Progress SonicMQ - Administration

**Fazit** SonicMQ kann durchwegs positiv beurteilt werden und wird einer Detailevaluierung unterzogen.

## Fiorano FioranoMQ

FioranoMQ ist einer der leistungsstärksten JMS Provider am Markt. Dies bestätigte sich im Laufe der Tests (vgl. Abbildung 3.15). Der gemessene Durchsatz unterlag nur geringen Schwankungen und blieb zu jedem Zeitpunkt über 2000 Nachrichten in der Sekunde.

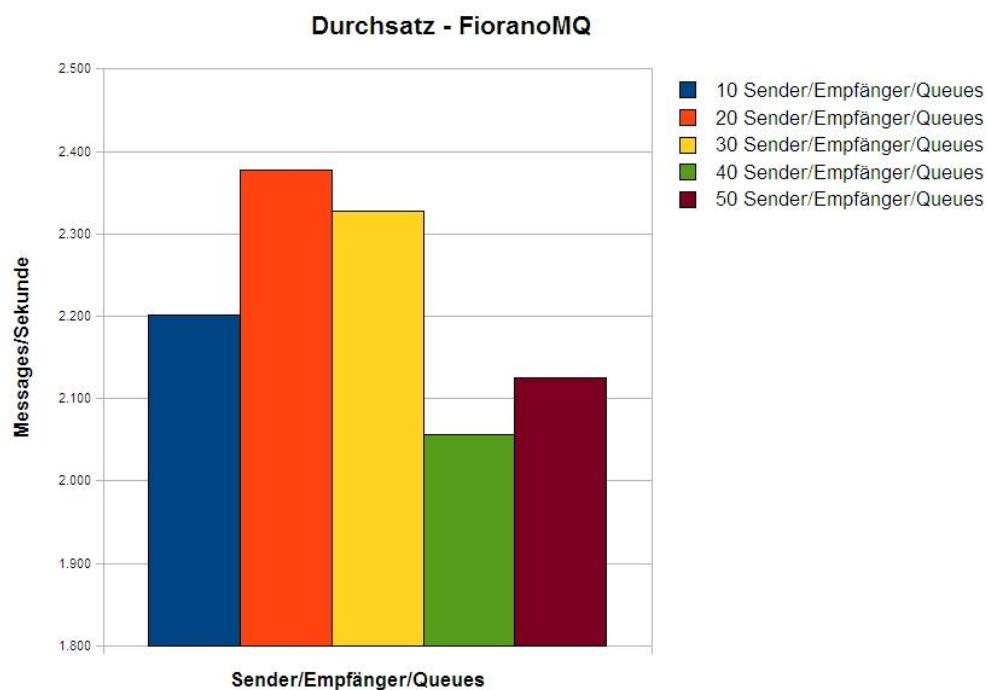


Abbildung 3.15: Fiorano FioranoMQ - Durchsatz

**Installation und Administration** Tabelle 3.18 auf der nächsten Seite zeigt die von Fiorano verifizierten System Anforderungen. Diese sind unter folgender Adresse einsehbar: [http://www.fiorano.com/products/fmq/fmq\\_sysreq.php](http://www.fiorano.com/products/fmq/fmq_sysreq.php)

FioranoMQ lässt sich über einen grafischen Wizard installieren (siehe Abbildung 3.16 auf der nächsten Seite).

Component	Requirements
<b>Operating Systems</b>	
UNIX	Linux Solaris Mac OS X HP-UX IBM AIX or any other platform that supports Java
Windows	Windows NT Windows 2000 Windows XP
<b>Software</b>	
no information provided	
<b>Hardware</b>	
CPU	>400MHz - >1GHz recommended
RAM	>256MB - 1GB recommended
Disk space	~250MB - 300MB

Tabelle 3.18: System Requirements - Fiorano FioranoMQ

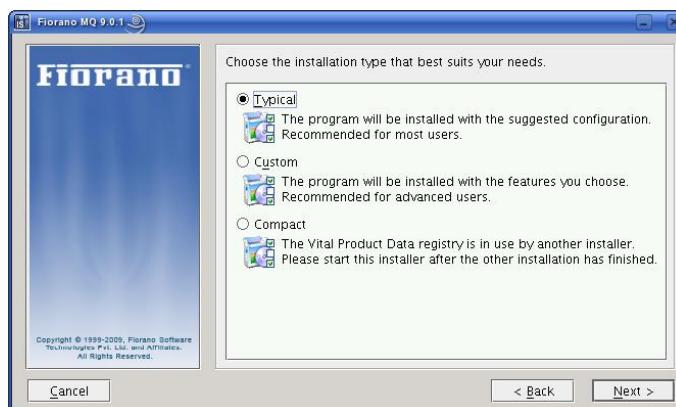


Abbildung 3.16: Fiorano FioranoMQ - Installation

Nach erfolgreicher Installation kann FioranoMQ mittels des fmq.sh Skriptes gestartet werden. Das Start Skript befindet sich im Installationspfad unter /fmq/bin.

FioranoMQ wird mit einer vollwertigen Management Konsole (Fiorano Studio, siehe Abbildung 3.17 auf der nächsten Seite) ausgeliefert, welches alle notwendigen Funktionen zur Administration und zum Monitoring bereitstellt.



Abbildung 3.17: Fiorano FioranoMQ - Administration

**Fazit** FioranoMQ kann durchwegs positiv beurteilt werden und wird einer Detailevaluierung unterzogen.

## Ergebnis der Grobevaluierung

Tabelle 3.19 zeigt die Ergebnisse der Grobevaluierung.

Kriterien	Apache	Sun	JBOSS	IBM	Oracle	Progress	Fiorano
<b>1 Applikationsanforderungen</b>							
<b>1.1 Funktionen/Features</b>							
1.1.2 Durchsatz	ja	ja	nein	nein	nein	ja	ja
1.2.4 Persistierung	ja	ja	ja	ja	ja	ja	ja
<b>1.2 Software-Merkmale</b>							
1.2.3 Sprachunabhängigkeit	ja	ja	ja	ja	ja	ja	ja
<b>2 Systemplattform</b>							
<b>2.1 Betriebssystem(e)</b>							
2.1.1 UNIX	ja	ja	ja	ja	ja	ja	ja
<b>Erfüllt alle KO-Kriterien?</b>	ja	ja	nein	nein	nein	ja	ja

Tabelle 3.19: Grobevaluierung

Alle Produkte die im Laufe der Grobevaluierung durchgehend positiv bewertet wurden, dh alle KO Kriterien erfüllen, werden im nächsten Schritt einer Detai-

levaluierung unterzogen. Alle Produkte, auf die dies nicht zutrifft, werden ab diesem Zeitpunkt aus der weiteren Evaluierung ausgeschlossen.

Die folgenden Produkte werden im Weiteren einer Detailevaluierung unterzogen.

- Apache ActiveMQ
- Sun OpenMQ
- Progress SonicMQ
- Fiorano FioranoMQ

### 3.3.3 Detailevaluierung

#### Apache ActiveMQ

**Bewertungsliste** Wie aus der Bewertungsliste (siehe Tabelle 3.20 auf der nächsten Seite, Kommentare siehe Anhang A.5.1 auf Seite 28) ersichtlich ist, schneidet Apache ActiveMQ durchwegs sehr gut ab. Sowohl hinsichtlich der geforderten Performance als auch der nicht funktionalen Kriterien lässt sich ActiveMQ als sehr gut bezeichnen.

**Risiken** Die Risiken (vgl. Tabelle 3.21 auf Seite 102) durch den Einsatz von ActiveMQ bewegen sich in einem tolerierbaren Bereich.

**Kosten** Die Kosten (vgl. Tabelle 3.22 auf Seite 103) für die Anschaffung von ActiveMQ belaufen sich auf null, da es sich um ein frei verfügbares System handelt. Ebenso ist kostenfreier Support über Foren und Mailinglisten verfügbar, es kann jedoch über Drittfirmen (zB <http://www.springsource.com/>) professioneller Support zugekauft werden. Die Kosten für zugekauften Support müssten in diesem Fall hinzugezogen werden.

Kriterien	Typ	G	P	Messgröße [min   max]	N	eff. P	eff. NW
<b>1 Applikationsanforderungen</b>							
<b>1.1 Funktionen/Features</b>							
1.1.1 Bearbeitungsdauer	m	18	72	min: -   max: 2ms	4	72	5,04
1.1.2 Durchsatz	ko	28	112	min: 500   max: -	4	112	7,84
1.1.3 Priorisierung	m	14	56	j/n	2	28	1,96
1.2.4 Persistierung	ko	30	120	j/n	4	120	8,4
1.1.5 Logging	w	10	40	j/n	4	40	2,8
<b>1.2 Software-Merkmale</b>							
1.2.1 Verfügbarkeit	ko	30	120	% lt. Herstellerangaben	2	60	2,8
1.2.2 Skalierbarkeit	m	22	88	Schwankungen von 1.1.1, 1.1.2	4	88	4,11
1.2.3 Sprachunabhängigkeit	ko	25	100	min: Java, Perl, plsql	3	75	3,5
1.2.4 Abhängigkeiten zu Fremdsystemen	w	14	56	min:0   max: 3	3	42	1,96
1.2.5 Wartbarkeit	m	9	36	min: -   max: 5h	4	36	1,68
<b>2 Systemplattform</b>							
<b>2.1 Betriebssystem(e)</b>							
2.1.1 UNIX	ko	100	400	min: AIX, SuSe   max: -	4	400	33
<b>2.2 Kommunikationsinfrastruktur</b>							
2.2.1 LAN	m	100	400	-	4	400	3,67
<b>3 Anbieterbezogene Kriterien</b>							
<b>3.1 Firmenmerkmale</b>							
3.1.1 Marktposition	w	100	400	-	3	300	2,5
<b>3.2 Dienstleistungen</b>							
3.2.1 Schulungen	w	13,33	53,33	j/n	2	26,66	0,89
3.2.2 Support	w	40	160	j/n	2	80	2,67
3.2.3 Bearbeitungsrecht	m	46,67	186,67	j/n	4	186,67	6,22
				<b>2400</b>		<b>2066,34</b>	<b>89,04</b>
$\Sigma$							

**G**...Gewicht  
**eff. P**...erfolgt erreichte Punkte  
**eff. NW**...effektiver Nutzwert  
**P**...Soll-Punkte  
**N**...Bewertung entspr. Skalierung (0-4)

Tabelle 3.20: Bewertungsliste Apache ActiveMQ

Risiko	Bewertung				Kommentar
	W	S	R		
<b>technische Risiken</b>					
Verfügbarkeit	2	3	11	11	Die Verfügbarkeit des Systems kann erst auf lange Sicht getestet werden. Da das Produkt jedoch bei diversen Unternehmen und Bildungseinrichtungen erfolgreich im Dauerbetrieb im Einsatz ist kann davon ausgegangen werden das in diesem Punkt keinerlei Probleme auftreten sollten.
Skalierbarkeit	2	3	11	11	Im Laufe der Test zeigte sich das ActiveMQ mit steigender Anzahl an Clients (Sender/Empfänger) einen beinahe konstanten Durchsatz schaffte und somit hervorragend skaliert.
Durchsatz	2	4	14	14	Der geforderte Minimaldurchsatz von 500 Nachrichten in der Sekunde wurde in allen Tests bei weiten übertrafen. Es könnte jedoch zu Problemen kommen in Bereichen die jenseits von 50 Sender/Empfänger Paaren liegen da dies nicht in den Tests überprüft wurde.
Wartbarkeit	3	2	11	11	Durch das auf das wesentliche fokussierte Management Interface könnte der Fall eintreten das bestimmte Sonderfunktionen nicht oder nur kompliziert ausführbar sind.
<b>personelle Risiken</b>					
fehlende Erfahrung	2	1	5	5	Da sich ActiveMQ stark an den JMS Standard orientiert wird die Einarbeitungszeit auf ein Minimum reduziert. Grundlegende Kenntnisse in der Verwendung von Java sind von Vorteil.
<b>Kosten-/Nutzenrisiken</b>					
Integration	2	4	14	14	Die Integration in das bestehende System kann zu unvorhersehbarem Verhalten des Systems führen. Dies könnte auftreten da es nicht bekannte Abhängigkeiten zwischen dem System und dem aktuellen Message Broker (BOSS) gibt.
<b>Partnerrisiken</b>					
Softwarehersteller	1	3	7	7	Apache ist ein bekannter und etablierter Hersteller von Softwareprodukten. Die Wahrscheinlichkeit das entweder der Hersteller (Apache) vom Markt verschwindet oder das Produkt als solches eingestellt wird ist als sehr gering einzuschätzen.
Plattformhersteller	1	3	7	7	Sowohl IBM als auch SuSe sind etablierte Hersteller von Betriebssystemen, die Wahrscheinlichkeit das diese Hersteller vom Markt verschwinden ist als sehr gering einzustufen.
<b>Gesamtrisikofaktor</b>	<b>80</b>				
<b>Gesamtrisikofaktor in % (R<sub>gp</sub>)</b>	<b>46</b>				

Tabelle 3.21: Risikobewertung Apache ActiveMQ

	Kosten	Kommentar	
<b>Investitionskosten</b>			
<b>extern</b>			
Kaufpreis	0	Lizenz: kostenfrei	einmalig
Schulungen	0	Grundlegende Einschulungen in das System: kostenfreie Tutorials	
<b>intern</b>			
Integration	90000	Anpassen der aktuellen Systeme 1500h á 60 €	einmalig
Test(s)	30000	Testen des neuen Systems nach der Integration 500h á 60 €	
<b>Investitionskosten gesamt</b>	<b>120 000</b>		
<b>Betriebskosten</b>			
<b>extern</b>			
Support	0	kostenfrei verfügbar	laufend
<b>intern</b>			
Problem & Change Management	1200	20h Jahr á 60 €	laufend
Administration & Wartung	3000	50h Jahr á 60 €	
<b>Betriebskosten gesamt</b>	<b>4 200</b>		
<b>Gesamtkosten [ Jahr]</b>	<b>124 200</b>		

Tabelle 3.22: Total Costs of Ownership - Apache ActiveMQ

## Sun OpenMQ

**Bewertungsliste** OpenMQ schneidet in allen Punkten der Evaluierung hervorragend ab (vgl. Tabelle 3.23 auf der nächsten Seite). Sowohl hinsichtlich der Performance als auch der nicht funktionalen Features bietet OpenMQ alle benötigten und geforderten Funktionen. Neben der einfachen Handhabung im Rahmen der Installation liefert OpenMQ ein hervorragendes Interface zur Administration.

**Risiken** Die Risiken (vgl. Tabelle 3.24 auf Seite 105), die durch den Einsatz von OpenMQ entstehen bewegen sich in einem akzeptablen Bereich.

**Kosten** Da durch die Anschaffung von OpenMQ keinerlei Kosten entstehen und es kostenfreien Support durch Foren und Mailinglisten gibt, belaufen sich die externen Anschaffungskosten auf null (siehe Tabelle 3.25 auf Seite 106). Sun bietet neben den kostenfreien Support auch kostenpflichtigen Support für seine Produkte. Die entsprechenden Kosten müssten in diesem Fall hinzugezogen werden.

Kriterien		Typ	G	P	Messgröße [min   max]	N	eff. P	eff. NW
<b>1 Applikationsanforderungen</b>								
<b>1.1 Funktionen/Features</b>								
1.1.1 Bearbeitungszeitdauer	m	18	72		min: -   max: 2ms	4	72	5,04
1.1.2 Durchsatz	ko	28	112		min: 500   max: -	4	112	7,84
1.1.3 Priorisierung	m	14	56		j/n	4	56	3,92
1.2.4 Persistierung	ko	30	120		j/n	4	120	8,4
1.1.5 Logging	w	10	40		j/n	4	40	2,8
<b>1.2 Software-Merkmale</b>								
1.2.1 Verfügbarkeit	ko	30	120	% lt. Herstellerangaben		4	120	5,6
1.2.2 Skalierbarkeit	m	22	88	Schwankungen von 1.1.1, 1.1.2		4	88	4,11
1.2.3 Sprachunabhängigkeit	ko	25	100	min: Java, Perl, plsql		3	75	3,5
1.2.4 Abhängigkeiten zu Fremdsystemen	w	14	56	min:0   max: 3		3	42	1,96
1.2.5 Wartbarkeit	m	9	36	min: -   max: 5h		4	36	1,68
<b>2 Systemplattform</b>								
<b>2.1 Betriebssystem(e)</b>								
2.1.1 UNIX	ko	100	400	min: AIX, SuSe   max: -		4	400	33
<b>2.2 Kommunikationsinfrastruktur</b>								
2.2.1 LAN	m	100	400	-		4	400	3,67
<b>3 Anbieterbezogene Kriterien</b>								
<b>3.1 Firmenmerkmale</b>								
3.1.1 Marktposition	w	100	400	-		4	400	3,33
<b>3.2 Dienstleistungen</b>								
3.2.1 Schulungen	w	13,33	53,33	j/n		4	53,33	1,78
3.2.2 Support	w	40	160	j/n		4	160	5,33
3.2.3 Bearbeitungsrecht	m	46,67	186,67	j/n		4	186,67	6,22
					<b>2400</b>		<b>2361</b>	<b>98,19</b>
<b>Σ</b>								

G...Gewicht  
eff. P...effektiv erreichte Punkte  
eff. NW...effektiver Nutzwert  
P...Soll-Punkte  
N...Bewertung entspr. Skalierung (0-4)

Tabelle 3.23: Bewertungsliste Sun OpenMQ

Risiko	Bewertung				Kommentar
	W	S	R		
<b>technische Risiken</b>					
Verfügbarkeit	1	3	7		Die Verfügbarkeit des Systems kann erst auf lange Sicht getestet werden. Da das Produkt für Flughafen- und Schienenverkehrssysteme (siehe <a href="https://mq.dev.java.net/customers.html">https://mq.dev.java.net/customers.html</a> ) eingesetzt wird, kann davon ausgegangen werden dass in diesem Punkt keine Probleme auftreten sollten.
Skalierbarkeit	1	3	7		Im Laufe der Test zeigte sich das OpenMQ mit steigender Anzahl an Clients (Sender/Empfänger) einen beinahe konstanten Durchsatz schaffte und somit hervorragend skaliert.
Durchsatz	2	4	14		Der geforderte Minimaldurchsatz von 500 Nachrichten in der Sekunde wurde in allen Tests bei weitem übertroffen. Es könnte jedoch zu Problemen kommen in Bereichen die jenseits von 50 Sender/Empfänger Paaren liegen da dies nicht in den Tests überprüft wurde.
Wartbarkeit	1	2	5		Die mitgelieferte Management Console bietet jegliche Funktionen um alle anfälligen administrativen Aufgaben zu erledigen.
<b>personelle Risiken</b>					
fehlende Erfahrung	2	1	5		Da sich OpenMQ stark am JMS Standard orientiert wird die Einarbeitungszeit auf ein Minimum reduziert. Grundlegende Kenntnisse in der Verwendung von Java sind von Vorteil.
<b>Kosten-/Nutzenrisiken</b>					
Integration	2	4	14		Die Integration in das bestehende System kann zu unvorhersehbarem Verhalten des Systems führen. Dies könnte auftreten da es nicht bekannte Abhängigkeiten zwischen dem System und dem aktuellen Message Broker (BOSS) gibt.
<b>Partnerrisiken</b>					
Softwarehersteller	1	3	7		Sun ist seit Jahren ein starker Partner sowohl im Bereich der Software als auch der Hardware. Gegen Ende April dieses Jahres wurde Sun von Oracle übernommen (siehe <a href="http://www.oracle.com/us/corporate/press/018363">http://www.oracle.com/us/corporate/press/018363</a> ).
Plattformhersteller	1	3	7		Sowohl IBM als auch SuSe sind etablierte Hersteller von Betriebssystemen, die Wahrscheinlichkeit das diese Hersteller vom Markt verschwinden ist als sehr gering einzustufen.
<b>Gesamtrisikofaktor</b>	<b>66</b>				
<b>Gesamtrisikofaktor in % (R<sub>gp</sub>)</b>	<b>35</b>				

Tabelle 3.24: Risikobewertung Sun OpenMQ

	<b>Kosten</b>	<b>Kommentar</b>	
<b>Investitionskosten</b>			
<b>extern</b>			
Kaufpreis	0	Lizenz: kostenfrei verfügbar	einmalig
Schulungen	0	Grundlegende Einschulungen in das System: kostenfreie Tutorials	
<b>intern</b>			
Integration	90000	Anpassen der aktuellen Systeme 1500h á 60 €	einmalig
Test(s)	30000	Testen des neuen Systems nach der Integration 500h á 60 €	
<b>Investitionskosten gesamt</b>	<b>120 000</b>		
<b>Betriebskosten</b>			
<b>extern</b>			
Support	0	kostenfrei verfügbar	laufend
Problem & Change Management	1200	20h Jahr á 60 €	
Administration & Wartung	3000	100h Jahr á 60 €	
<b>Betriebskosten gesamt</b>	<b>4 200</b>		
<b>Gesamtkosten [ Jahr]</b>	<b>124 200</b>		

Tabelle 3.25: Total Costs of Ownership - Sun OpenMQ

## Progress SonicMQ

**Bewertungsliste** Progress SonicMQ kann durch alle Kriterienbereiche hindurch als hervorragend bezeichnet werden (siehe Tabelle 3.26 auf der nächsten Seite).

**Risiken** Die Risiken (vgl. Tabelle 3.27 auf Seite 109), die durch den Einsatz von SonicMQ entstehen bewegen sich in einen akzeptablen Bereich.

**Kosten** Es wurden durch das Unternehmen Progress keine Angaben zu den Kosten der Anschaffung und des Supports getätigt. Aus diesem Grund kann im Bereich der Kosten keine Gegenüberstellung zu den konkurrierenden Produkten erfolgen.

Kriterien	Typ	G	P	Messgröße [min   max]	N	eff. P	eff. NW
<b>1 Applikationsanforderungen</b>							
<b>1.1 Funktionen/Features</b>							
1.1.1 Bearbeitungsdauer	m	18	72	min: -   max: 2ms	4	72	5,04
1.1.2 Durchsatz	ko	28	112	min: 500   max: -	4	112	7,84
1.1.3 Priorisierung	m	14	56	j/n	4	56	3,92
1.2.4 Persistierung	ko	30	120	j/n	4	120	8,4
1.1.5 Logging	w	10	40	j/n	4	40	2,8
<b>1.2 Software-Merkmale</b>							
1.2.1 Verfügbarkeit	ko	30	120	% lt. Herstellerangaben	3	90	4,2
1.2.2 Skalierbarkeit	m	22	88	Schwankungen von 1.1.1, 1.1.2	4	88	4,11
1.2.3 Sprachunabhängigkeit	ko	25	100	min: Java, Perl, plsql	3	75	3,5
1.2.4 Abhängigkeiten zu Fremdsystemen	w	14	56	min:0   max: 3	3	42	1,96
1.2.5 Wartbarkeit	m	9	36	min: -   max: 5h	4	36	1,68
<b>2 Systemplattform</b>							
<b>2.1 Betriebssystem(e)</b>							
2.1.1 UNIX	ko	100	400	min: AIX, SuSe   max: -	4	400	33
<b>2.2 Kommunikationsinfrastruktur</b>							
2.2.1 LAN	m	100	400	-	4	400	3,67
<b>3 Anbieterbezogene Kriterien</b>							
<b>3.1 Firmenmerkmale</b>							
3.1.1 Marktposition	w	100	400	-	4	400	3,33
<b>3.2 Dienstleistungen</b>							
3.2.1 Schulungen	w	13,33	53,33	j/n	4	53,33	1,78
3.2.2 Support	w	40	160	j/n	4	160	5,33
3.2.3 Bearbeitungsrecht	m	46,67	186,67	j/n	0	0	0
				<b>2400</b>			
					<b>Σ</b>		
						<b>2144,32</b>	<b>90,57</b>

**G**...Gewicht  
**eff. P**...erfolgt erreichte Punkte  
**eff. NW**...effektiver Nutzwert  
**P**...Soll-Punkte  
**N**...Bewertung entspr. Skalierung (0-4)

Tabelle 3.26: Bewertungsliste Progress SonicMQ

Risiko	Bewertung				Kommentar
	W	S	R		
<b>technische Risiken</b>					
Verfügbarkeit	1	3	7		Die Verfügbarkeit des Systems kann erst auf lange Sicht getestet werden. Da die Produkte von Progress laut deren Auskunft (siehe <a href="http://www.sonicsoftware.com/customers/index.ssp">http://www.sonicsoftware.com/customers/index.ssp</a> ) weltweit bei mehr als 60.000 Unternehmen im Einsatz sind, kann davon ausgegangen werden das in diesem Punkt keinerlei Probleme auftreten sollten.
Skalierbarkeit	1	3	7		Im Laufe der Test zeigte sich das SonicMQ mit steigender Anzahl an Clients (Sender/Empfänger) einen beinahe konstanten Durchsatz schaffte und somit hervorragend skaliert.
Durchsatz	1	4	9		Der geforderte Minimaldurchsatz von 500 Nachrichten in der Sekunde wurde in allen Tests bei weiten übertrroffen. Es könnte jedoch zu Problemen kommen in Bereichen die jenseits von 50 Sender/Empfänger Paaren liegen da dies nicht in den Tests überprüft wurde.
Wartbarkeit	1	2	5		Die mitgelieferte Management Console bietet jegliche Funktionen um alle anfälligen administrativen Aufgaben zu erledigen.
<b>personelle Risiken</b>					
fehlende Erfahrung	2	1	5		Da sich SonicMQ stark an den JMS Standard orientiert wird die Einarbeitungszeit auf ein Minimum reduziert. Grundlegende Kenntnisse in der Verwendung von Java sind von Vorteil.
<b>Kosten-/Nutzenrisiken</b>					
Integration	2	4	14		Die Integration in das bestehende System kann zu unvorhersehbarem Verhalten des Systems führen. Dies könnte auftreten da es nicht bekannte Abhängigkeiten zwischen dem System und dem aktuellen Message Broker (BOSS) gibt.
<b>Partnerrisiken</b>					
Softwarehersteller	1	3	7		Progress ist ein etablierter Hersteller von Software Produkten mit einer Vielzahl an Kunden in den unterschiedlichsten Sparten. Die Wahrscheinlichkeit das Progress vom Markt verschwindet ist als sehr gering einzustufen.
Plattformhersteller	1	3	7		Sowohl IBM als auch SuSe sind etablierte Hersteller von Betriebssystemen, die Wahrscheinlichkeit das diese Hersteller vom Markt verschwinden ist als sehr gering einzustufen.
<b>Gesamtrisikofaktor</b>	<b>61</b>				
<b>Gesamtrisikofaktor in % (<math>R_{gp}</math>)</b>	<b>31</b>				

Tabelle 3.27: Risikobewertung Progress SonicMQ

## Fiorano FioranoMQ

**Bewertungsliste** FioranoMQ zeigte im Laufe der Evaluierung deutlich die beste Performance aller getesteten Produkte. Weiters zeichnet sich FioranoMQ durch die einfache Administration mittels des mitgelieferten Administrations Interfaces (Fiorano Studio) aus. Generell lässt sich somit FioranoMQ als hervorragend bezeichnen (vgl. Tabelle 3.28 auf der nächsten Seite).

**Risiken** Die Risiken (vgl. Tabelle 3.29 auf Seite 112), welche durch den Einsatz von FioranoMQ entstehen lassen sich durchwegs mit den der anderen Produkte vergleichen und bewegen sich somit in einem tolerierbaren Bereich.

**Kosten** Es wurden durch das Unternehmen Fiorano keine Angaben zu den Kosten der Anschaffung und des Supports getätigt. Aus diesem Grund kann im Bereich der Kosten keine Gegenüberstellung zu den konkurrierenden Produkten erfolgen.

	Kriterien	Typ	G	P	Messgröße [min   max]	N	eff. P	eff. NW
<b>1 Applikationsanforderungen</b>								
<b>1.1 Funktionen/Features</b>								
1.1.1 Bearbeitungsdauer	m	18	72		min: -   max: 2ms	4	72	5,04
1.1.2 Durchsatz	ko	28	112		min: 500   max: -	4	112	7,84
1.1.3 Priorisierung	m	14	56		j/n	4	56	3,92
1.2.4 Persistierung	ko	30	120		j/n	4	120	8,4
1.1.5 Logging	w	10	40		j/n	4	40	2,8
<b>1.2 Software-Merkmale</b>								
1.2.1 Verfügbarkeit	ko	30	120	% lt. Herstellerangaben		4	120	5,6
1.2.2 Skalierbarkeit	m	22	88	Schwankungen von 1.1.1, 1.1.2		4	88	4,11
1.2.3 Sprachunabhängigkeit	ko	25	100	min: Java, Perl, plsql		3	75	3,5
1.2.4 Abhängigkeiten zu Fremdsystemen	w	14	56	min:0   max: 3		3	42	1,96
1.2.5 Wartbarkeit	m	9	36	min: -   max: 5h		4	36	1,68
<b>2 Systemplattform</b>								
<b>2.1 Betriebssystem(e)</b>								
2.1.1 UNIX	ko	100	400	min: AIX, SuSe   max: -		4	400	33
<b>2.2 Kommunikationsinfrastruktur</b>								
2.2.1 LAN	m	100	400	-		4	400	3,67
<b>3 Anbieterbezogene Kriterien</b>								
<b>3.1 Firmenmerkmale</b>								
3.1.1 Marktposition	w	100	400	-		4	400	3,33
<b>3.2 Dienstleistungen</b>								
3.2.1 Schulungen	w	13,33	53,33	j/n		4	53,33	1,78
3.2.2 Support	w	40	160	j/n		4	160	5,33
3.2.3 Bearbeitungsrecht	m	46,67	186,67	j/n		0	0	0
				<b>2400</b>			<b>2174,32</b>	<b>91,97</b>

G...Gewicht  
 eff. P...erfolgt erreichte Punkte  
 eff. NW...effektiver Nutzwert  
 P...Soll-Punkte  
 N...Bewertung entspr. Skalierung (0-4)

Tabelle 3.28: Bewertungsliste Fiorano FioranoMQ

Risiko	Bewertung				Kommentar
	W	S	R		
<b>technische Risiken</b>					
Verfügbarkeit	1	3	7		Die Verfügbarkeit des Systems kann erst auf lange Sicht getestet werden. Da die Produkte jedoch bei einer Vielzahl an Unternehmen (siehe <a href="http://www.fiorano.com/resources/successstories/successstories.php">http://www.fiorano.com/resources/successstories/successstories.php</a> ) erfolgreich im Einsatz sind, kann davon ausgegangen werden das es in diesem Punkt zu keinerlei Problemen kommen sollte.
Skalierbarkeit	1	3	7		Im Laufe der Test zeigte sich das FioranoMQ mit steigender Anzahl an Clients (Sender/Empfänger) einen beinahe konstanten Durchsatz schaffte und somit hervorragend skaliert.
Durchsatz	1	4	9		Der geforderte Minimaldurchsatz von 500 Nachrichten in der Sekunde wurde in allen Tests bei weiten übertrffen. Der Minimal festgestellte Wert lag über 2.000 Nachrichten in der Sekune. Es könnte jedoch zu Problemen kommen in Bereichen die jenseits von 50 Sender/Empfänger Paaren liegen da dies in den Tests nicht überprüft wurde.
Wartbarkeit	1	2	5		Die mitgelieferte Management Console bietet jegliche Funktionen um alle anfälligen administrativen Aufgaben zu erledigen.
<b>personelle Risiken</b>					
fehlende Erfahrung	2	1	5		Da sich FioranoMQ stark an den JMS Standard orientiert wird die Einarbeitungszeit auf ein Minimum reduziert. Grundlegende Kenntnisse in der Verwendung von Java sind von Vorteil.
<b>Kosten-/Nutzenrisiken</b>					
Integration	2	4	14		Die Integration in das bestehende System kann zu unvorhersehbarem Verhalten des Systems führen. Dies könnte auftreten da es nicht bekannte Abhängigkeiten zwischen dem System und dem aktuellen Message Broker (BOSS) gibt.
<b>Partnerrisiken</b>					
Softwarehersteller	1	3	7		Fiorano ist ein etablierter und erfolgreicher Hersteller von Software Lösungen im Bereich der Middleware und kann eine Vielzahl an entsprechenden Referenzen aufweisen. Die Wahrscheinlichkeit das Fiorano vom Markt verschwindet ist als sehr gering einzuschätzen.
Plattformhersteller	1	3	7		Sowohl IBM als auch SuSe sind etablierte Hersteller von Betriebssystemen, die Wahrscheinlichkeit das diese Hersteller vom Markt verschwinden ist als sehr gering einzustufen.
<b>Gesamtrisikofaktor</b>	<b>61</b>				
<b>Gesamtrisikofaktor in % (<math>R_{gp}</math>)</b>	<b>31</b>				

Tabelle 3.29: Risikobewertung Fiorano FioranoMQ

### 3.3.4 Gegenüberstellung der Produkte

Alle Produkte, die einer Detailevaluierung unterzogen worden sind, können als grundlegend sehr gut geeignet bezeichnet werden und liefern in vielen Punkten ähnliche bis identische Ergebnisse. In Tabelle 3.30 auf der nächsten Seite werden die einzelnen Evaluierungsergebnisse zusammengefasst und einander gegenübergestellt.

Anhand des effektiven Nutzwertes kann OpenMQ von Sun als optimale Lösung hervorgehoben werden.

#### 3.3.4.1 Bewertungen

Im Spinnendiagramm (siehe Abbildung 3.18, (vgl. [Schreiber2003], S.227)) wird der effektiv erreichte Nutzwert der einzelnen Kriterien im Verhältnis zum theoretischen Maximum dargestellt, und dient dazu einen schnellen Überblick über die Produkte und deren Abschneiden zu erhalten. Da in diesem Fall alle Produkte sehr nahe beieinander liegen ist ein großer Teil der resultierenden Kurven deckungsgleich.

Nutzwertprofil

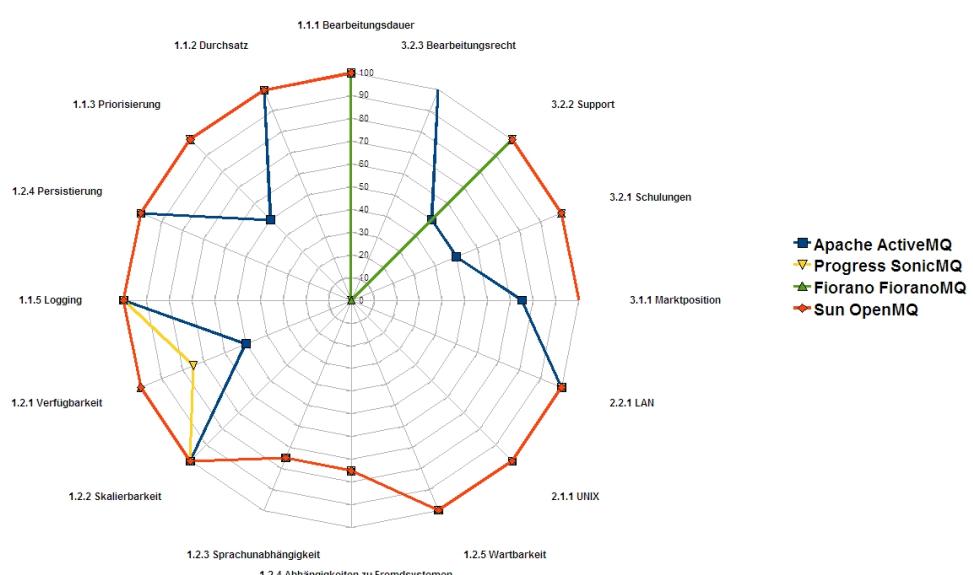


Abbildung 3.18: Nutzwertprofil - Effektive Nutzwerte in Prozent des maximalen Nutzwertes

	Open Source				Proprietär			
	Apache ActiveMQ		Sun OpenMQ		Progress SonicMQ		Fiorano FioranoMQ	
	P	eff.NW	P	eff.NW	P	eff.NW	P	eff.NW
<b>1 Applikationsanforderungen</b>								
<b>1.1 Funktionen/Features</b>								
1.1.1 Bearbeitungszeitdauer	72	5,04	72	5,04	72	5,04	72	5,04
1.1.2 Durchsatz	112	7,84	112	7,84	112	7,84	112	7,84
1.1.3 Priorisierung	28	1,96	56	3,92	56	3,92	56	3,92
1.2.4 Persistierung	120	8,4	120	8,4	120	8,4	120	8,4
1.1.5 Logging	40	2,8	40	2,8	40	2,8	40	2,8
<b>1.2 Software-Merkmale</b>								
1.2.1 Verfügbarkeit	60	2,8	120	5,6	90	4,2	120	5,6
1.2.2 Skalierbarkeit	88	4,11	88	4,11	88	4,11	88	4,11
1.2.3 Sprachunabhängigkeit	75	3,5	75	3,5	75	3,5	75	3,5
1.2.4 Abhängigkeiten zu Fremdsystemen	42	1,96	42	1,96	42	1,96	42	1,96
1.2.5 Wartbarkeit	36	1,68	36	1,68	36	1,68	36	1,68
<b>2 Systemplattform</b>								
<b>2.1 Betriebssystem(e)</b>								
2.1.1 UNIX	400	33	400	33	400	33	400	33
<b>2.2 Kommunikationsinfrastruktur</b>								
2.2.1 LAN	400	3,67	400	3,67	400	3,67	400	3,67
<b>3 Anbieterbezogene Kriterien</b>								
<b>3.1 Firmenmerkmale</b>								
3.1.1 Marktposition	300	2,5	400	3,33	400	3,33	400	3,33
<b>3.2 Dienstleistungen</b>								
3.2.1 Schulungen	26,66	0,89	53,33	1,78	53,33	1,78	53,33	1,78
3.2.2 Support	80	2,67	160	5,33	160	5,33	160	5,33
3.2.3 Bearbeitungsrecht	186,68	6,22	186,68	6,22	0	0	0	0
$\sum P$	<b>2066,34</b>		<b>2361</b>		<b>2144,32</b>		<b>2174,32</b>	
$\sum eff.NW$		<b>89,04</b>		<b>98,19</b>		<b>90,57</b>		<b>91,97</b>

P ... erreichte Punkte  
eff. NW ... effektiver Nutzwert

Tabelle 3.30: Gegenüberstellung

### 3.3.4.2 Risiken

Tabelle 3.31 zeigt die Ergebnisse der Risikobewertungen. Abbildung 3.19 stellt die Risiken nochmals grafisch dar, um eine schnelle Übersicht über die Risiken der jeweiligen Produkte zu gewährleisten.

Wie 3.2.10 auf Seite 77, beschrieben müssten alle Produkte welche einen Gesamtrisikofaktor [%] von mehr als 60% haben aus der weiteren Bedachtnahme ausscheiden. Da aber alle Produkte weit unterhalb dieses Grenzwertes liegen, können alle Produkte berücksichtigt werden.

	Apache ActiveMQ	Sun OpenMQ	Progress SonicMQ	Fiorano FioranoMQ
<b>technische Risiken</b>				
Verfügbarkeit	11	7	7	7
Skalierbarkeit	11	7	7	7
Durchsatz	14	14	9	9
Wartbarkeit	11	5	5	5
<b>personelle Risiken</b>				
fehlende Erfahrung	5	5	5	5
<b>Kosten-/Nutzenrisiken</b>				
Integration	14	14	14	14
<b>Partnerrisiken</b>				
Softwarehersteller	7	7	7	7
Plattformhersteller	7	7	7	7
<b>Gesamtrisiko</b>	<b>80</b>	<b>66</b>	<b>61</b>	<b>61</b>
<b>Gesamtrisiko [%]</b>	<b>46</b>	<b>35</b>	<b>31</b>	<b>31</b>

Tabelle 3.31: Risikobewertungen

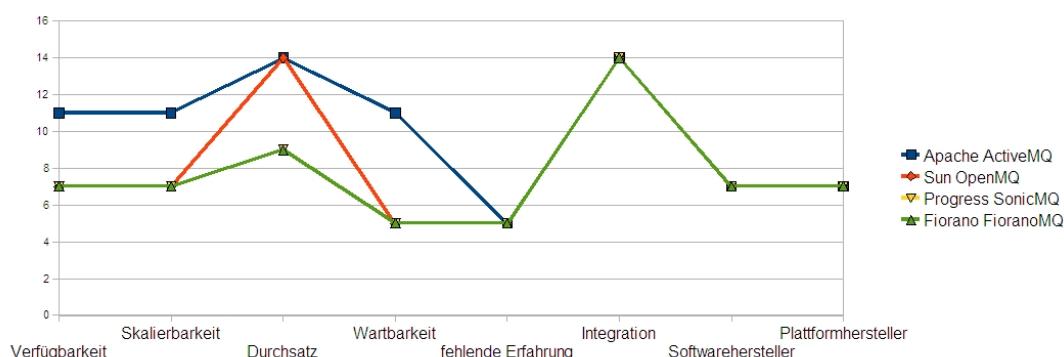


Abbildung 3.19: Risikoprofile

### 3.3.4.3 Kosten

Sowohl Apache ActiveMQ als auch Sun OpenMQ sind vollkommen kostenfrei erhältlich. In diesem Sinne belaufen sich die Gesamtkosten für diese Produkte auf die internen Kosten.

Da weder durch das Unternehmen Progress noch durch Fiorano Angaben zu den Lizenz- bzw. Supportkosten gemacht wurden, kann kein weiterer Vergleich der Kosten an dieser Stelle stattfinden.

Sollte die Entscheidung zu einem dieser beiden Produkte tendieren, müssen die Kosten nochmals mit dem jeweiligen Unternehmen abgestimmt werden.

### 3.3.5 Produktempfehlung

Anhand der Gegenüberstellung der Produkte kann eine Empfehlung für das Produkt OpenMQ von Sun ausgesprochen werden. OpenMQ liefert den höchsten effektiven Nutzwert, dh die Anforderungen am Besten erfüllt. Ebenso bewegen sich die Risiken welche durch den Einsatz von OpenMQ entstehen würden in einem akzeptablen Bereich.

Abbildung 3.20 auf der nächsten Seite zeigt in einer kumulierte Darstellung der erreichten effektiven Nutzwerte pro Kriterium. Diese Darstellung ermöglicht einen schnellen Vergleich der Produkte und zeigt deutlich in welchen Kriterien sich die einzelnen Produkte voneinander unterscheiden.

Es zeigt sich das alle Produkte nahe bei einander liegen, und sich nur geringfügig unterscheiden. Unterschiede lassen sich in den Kriterien Priorisierung und Verfügbarkeit sowie in den anbieterbezogenen Kriterien feststellen.

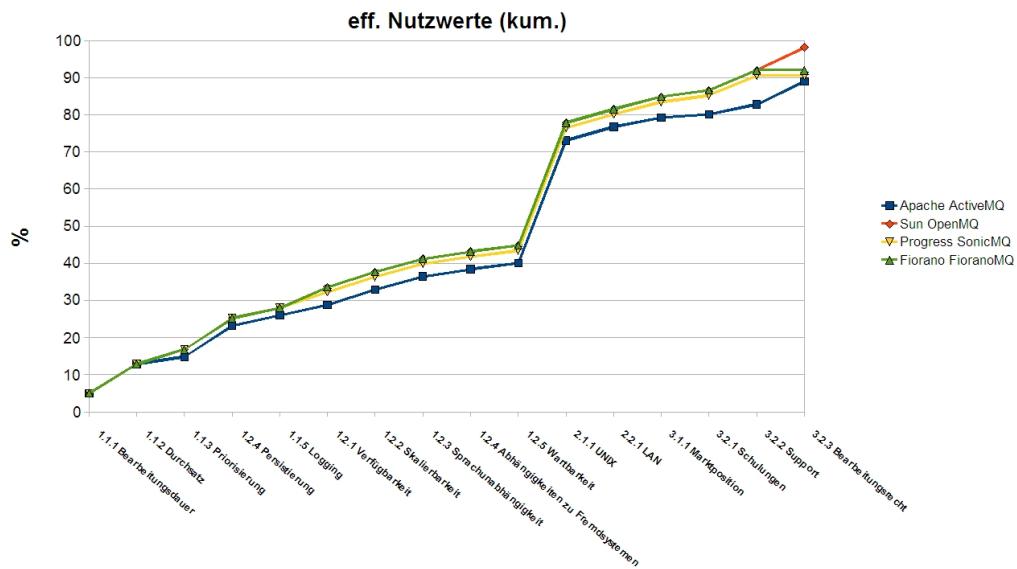


Abbildung 3.20: Effektive Nutzwerte (kumuliert)

Apache ActiveMQ unterstützt nativ keinerlei Priorisierung der Nachrichten und wurde in den anbieterbieterbezogenen Kriterien schlechter als die Konkurrenz beurteilt. Sun OpenMQ, Progress SonicMQ und Fiorano FioranoMQ liegen bis auf das Kriterium „Bearbeitungsrecht“ extrem nahe beieinander.

Da es sich sowohl bei Progress als auch Fiorano um proprietäre Hersteller handelt darf der Source Code des Produktes weder eingesehen noch eigenständig geändert werden. Dahingegen kann der Source Code von Sun OpenMQ in vollem Umfang eingesehen und geändert werden. Dies stellt somit den ausschlaggebenden Faktor in Richtung Sun OpenMQ dar.

# 4

## Kapitel 4

### Conclusio

Die Arbeitshypothese (siehe auf Seite i) kann in vollem Umfang bestätigt werden. Anhand eines strukturierten Evaluierungsprozesses wurden die am Markt verfügbaren Produkte einer Grob- sowie einer Detailevaluierung unterzogen. Im Laufe der Evaluierungen ergaben sich vier potentielle Kandidaten für einen Einsatz im Unternehmen. Diese waren Apache ActiveMQ, Sun OpenMQ, Progress SonicMQ sowie Fiorano FioranoMQ.

Grundsätzlich würden sich alle vier Produkte eignen, Sun OpenMQ lieferte in diesem Zusammenhang den größten effektiven Nutzwert aller Produkte und steht zudem kostenfrei zur Verfügung. Da es eine ausreichend große Community hinter OpenMQ gibt, sowie die Möglichkeit professionellen Support durch Sun zu zukaufen, stellt auch dieser Punkt kein Hindernis dar. Somit lässt sich eine klare Empfehlung für Sun OpenMQ aussprechen.

Im Laufe der Evaluierung zeigte sich, dass namhafte Hersteller (zB IBM, Oracle) wesentlich schlechter beurteilt werden mussten. Dies betraf nicht nur die unproportional aufwendige und unübersichtliche Handhabung, sondern auch die festgestellte Performance. Beide Produkte (IBM WebsphereMQ, Oracle Advanced Queuing) erreichten bei weitem nicht die geforderten Minimalwerte.

Die Vermutung liegt nahe, dass auf leistungsstärkeren Systemen der Unterschied zwischen diesen und den Konkurrenzprodukten weniger deutlich ausfällt. Verschiedenste Benchmarks zeigen jedoch immer noch gravierende Unterschiede auf. ([Crimson2003, Dell2007])

Generell lässt sich anmerken, dass sowohl Open Source als auch kommerzielle Produkte die benötigten Leistungen erbringen. Die Entscheidung hinsichtlich eines Produktes kann in diesem Sinne von der benötigten Tiefe des Supports abhängig gemacht werden. Sollte als Support eine aktive Community ausreichen, so lässt sich an dieser Stelle ein klare Empfehlung hinsichtlich Open Source Produkten aussprechen.

# Literaturverzeichnis

- [Dangelmaier2002] WILHELM DANGELMAIER, HAGEN LESSING, ULRICH PAPE, MICHAEL RÜTHER: *Klassifikation von EAI-Systemen*. HMD 2002, Heft 225 Seite: 61-71, <http://hmd.dpunkt.de/225/06.html>, /LITERATUR/[DANGELMAIER2002]
- [Dell2007] RAMKUMAR RAJAGOPAL, MAHMOUD AHMADIAN: *Using Oracle Streams Advanced Queuing in Application Messaging Infrastructures*. Dell Power Solutions 2007, Mai Ausgabe Seite: 84-88, <http://www.dell.com/downloads/global/power/ps2q07-20070232-Rajagopal.pdf>, /LITERATUR/[DELL2007]
- [Hohpe2005] GREGOR HOPHE, BOBBY WOOLF: *Enterprise Integration Patterns*. Addison-Wesley 2005, 5te Ausgabe
- [Manowarda1999] MARLIES MANOWARDA: *Dokumentation der SSI Schäfer Messaging Middleware - BOSS*. **firmenintern** 1999, 1te Ausgabe
- [Piedad2001] FLOYD PIEDAD, MICHAEL HAWKINS: *High Availability - Design, Techniques and Processes*. Prentice Hall 2001
- [Reussner2009] RALF REUSSNER, WILHELM HASSELBRING: *Handbuch der Software-Architektur*. DPunkt Verlag 2009, 2te Ausgabe
- [Kersken2008] SASCHA KERSKEN: *IT-Handbuch für Fachinformatiker*. Galileo-Press 2008, 4te (Online) Ausgabe, [http://openbook.galileocomputing.de/it\\_handbuch/](http://openbook.galileocomputing.de/it_handbuch/), /LITERATUR/[KERSKEN2008]
- [Schreiber2003] JOSEF SCHREIBER: *Beschaffung von Informatikmitteln*. Haupt 2003, 4te Ausgabe
- [Ullenboom2009] CHRISTIAN ULLENBOOM: *Openbook: Java ist auch eine Insel*. Galileo-Press 2009, 8te (Online) Ausgabe, <http://openbook.galileocomputing.de/javainsel8/>, /LITERATUR/[ULLENBOOM2009]

# Internet-Ressourcen

[Apache2004] APACHE: *Apache Software Lizenz*. 2004, Version 2, <http://www.apache.org/licenses/LICENSE-2.0.txt>, VISITED: 08-05-2009, /LITERATUR/[APACHE2004]

[BrianGoetz2008] BRIAN GOETZ, ROBERT ECKSTEIN: *An Introduction to Real-Time Java Technology: Part 1, The Real-Time Specification for Java*. 2008, [http://java.sun.com/developer/technicalArticles/Programming/rt\\_pt1/](http://java.sun.com/developer/technicalArticles/Programming/rt_pt1/), VISITED: 27-01-2009, /LITERATUR/[BRIANGOETZ2008]

[Crimson2003] CRIMSON CONSULTING GROUP: *High-Performance JMS Messaging - A Benchmark Comparison of Sun Java System Message Queue and IBM Websphere MQ*. 2003, [http://wwws.sun.com/software/products/message\\_queue/wp\\_JMSperformance.pdf](http://wwws.sun.com/software/products/message_queue/wp_JMSperformance.pdf), VISITED: 28-05-2009, /LITERATUR/[CRIMSON2003]

[ITWISSEN2009] WWW.ITWISSEN.INFO: *Persistenz/persistence* 2009, <http://www.itwissen.info/definition/lexikon/Persistenz-persistence.html>, VISITED: 19-02-2009, /LITERATUR/[ITWISSEN2009]

[ITWISSEN2009a] WWW.ITWISSEN.INFO: *Middleware* 2009, <http://www.itwissen.info/definition/lexikon/Middleware-middleware.html>, VISITED: 24-02-2009, /LITERATUR/[ITWISSEN2009A]

[Kirwin1987] BILL KIRWIN: *Total Costs of Ownership* 1987-2009, <http://amt.gartner.com/TCO/MoreAboutTCO.htm>, VISITED: 10-02-2009, /LITERATUR/[KIRWIN1987]

[LorbeerDesign2007] LORBEER DESIGN: *Das neue Unternehmen am Markt positionieren.* 2007, [http://www.lorbeerdesign.de/admin/files/422/vortrag\\_LD\\_071106\\_klein2.pdf](http://www.lorbeerdesign.de/admin/files/422/vortrag_LD_071106_klein2.pdf), VISITED: 08-06-2009, /LITERATUR/[LORBEERDESIGN2007]

[Microsoft2009] PETER HOUSTON: *Integrating Applications with Message Queuing Middleware* 2009, <http://technet.microsoft.com/en-us/library/cc723259.aspx>, VISITED: 30-03-2009, /LITERATUR/[MICROSOFT2009]

[Microsoft2009a] MSDN: *Entwerfen von verteilten Anwendung mit Visual Studio .NET - Testen der Verfügbarkeit* 2009, <http://msdn>.

microsoft.com/de-de/library/aa292184, VISITED: 15-04-2009, /LITERATUR/[MICROSOFT2009A]

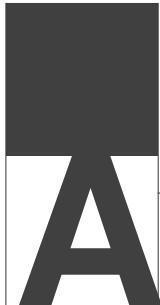
[Middleware2009] WWW.MIDDLEWARE.ORG: *Message Queuing Middleware* 2009, <http://www.middleware.org/general/mqm.html>, VISITED: 30-03-2009, /LITERATUR/[MIDDLEWARE2009]

[Reinecke2004] GOETZ REINECKE: *Notwendigkeit von Middleware* 2004, [http://www.activevb.de/tutorials/tut\\_middleware/middleware2.html](http://www.activevb.de/tutorials/tut_middleware/middleware2.html), VISITED: 24-02-2009, /LITERATUR/[REINECKE2004]

[RTI2006] WWW.RTI.COM: *Can Ethernet be Real-Time?* 2006, <http://www.rti.com/mk/Ethernet.html>, VISITED: 12-01-2009, /LITERATUR/[RTI2006]

[SAP2009] WWW.SAP.COM: *SAP 101 Terms Glossary - Support* 2006, [http://www.sap.com/usa/solutions/Glossary/p\\_to\\_t.epx](http://www.sap.com/usa/solutions/Glossary/p_to_t.epx), VISITED: 16-04-2009, /LITERATUR/[SAP2009]

[SDI-Research2009] WWW.SDI-RESEARCH.AT: *Marktpositionierung* 2006, <http://www.sdi-research.at/strategie/marktpositionierung.html>, VISITED: 26-05-2009, /LITERATUR/[SDI-RESEARCH2009]



Anhang A

## Anhang

### A.1 Kontakte

<p><b>FH JOANNEUM</b> INFORMATIONSMANAGEMENT</p> <p>Alte Poststraße 147, 8020 Graz</p>	<p><b>SSI SCHÄFER</b></p> <p>Fischeraustraße 27, 8051 Graz</p>
<p>Tel: ..... +43 (0) 316 / 5453-8500 Fax: ..... +43 (0) 316 / 5453-8501 Web: ..... <a href="http://fh-joanneum.at/ima">http://fh-joanneum.at/ima</a> E-Mail: ..... <a href="mailto:info@fh-joanneum.at">info@fh-joanneum.at</a></p>	<p>Tel: ..... +43 (0) 316 / 6096-0 Fax: ..... +43 (0) 316 / 6096-455 Web: ..... <a href="http://www.ssi-schaefer.at">http://www.ssi-schaefer.at</a> E-Mail: ... <a href="mailto:sales@ssi-schaefer-peem.com">sales@ssi-schaefer-peem.com</a></p>
<p><b>Betreuer:</b> <b>FH-Prof. Dipl.-Ing. Peter Salhofer</b> Tel: ..... +43 (0) 316 / 5453-8518 E-Mail: ... <a href="mailto:peter.salhofer@fh-joanneum.at">peter.salhofer@fh-joanneum.at</a></p>	<p><b>Betreuer:</b> <b>DI(FH) Dieter Zeiml</b> Tel: ..... +43 (0) 316 6096-368 E-Mail: ... <a href="mailto:d.zeiml@ssi-schaefer-peem.com">d.zeiml@ssi-schaefer-peem.com</a></p>

<p><b>Alexander Josef Marktl</b></p> <p>Augasse 44/1, 8051 Graz</p>
<p>Tel (Privat): ..... +43 (0) 660 / 6871 329 Tel (Büro): ..... +43 (0) 316 / 6096-544 E-Mail (1 - FH): ..... <a href="mailto:AlexanderJosef.Marktl.ima05@fh-joanneum.at">AlexanderJosef.Marktl.ima05@fh-joanneum.at</a> E-Mail (2 - SSI Schäfer PEEM): ..... <a href="mailto:a.marktl@ssi-schaefer-peem.com">a.marktl@ssi-schaefer-peem.com</a> E-Mail (3 - Privat): ..... <a href="mailto:alexander.marktl@gmx.net">alexander.marktl@gmx.net</a></p>

## A.2 Testumgebung

Die Evaluierungen der einzelnen Produkte wurden auf einem Dell Latitude D830, Intel® Core™ 2 Duo T7500 durchgeführt. Als Betriebssystem kam SuSe Linux 10.1 in einer Virtual Machine zum Einsatz. Die Virtual Machine konnte über einen Prozessorkern sowie 512MB RAM verfügen.

Bei den Test wurden sowohl die Sender/Empfänger als auch die Messaging Middleware auf ein und dem selben System ausgeführt, dieser Aufbau wurde gewählt da er der realen Situation entspricht, um gravierende Umbrüche zum bestehenden System so weit als möglich zu vermeiden, wurde nur auf Point-to-Point Szenarien eingegangen.

Die Testprogramme wurden durchgehend in Java™ verfasst. Sender und Empfänger wurden jeweils so verfasst das diese asynchron arbeiten, dh der Sender schickt die Nachrichten an eine Queue, der Empfänger registriert einen Listener an der jeweiligen Queue und wird entsprechend benachrichtigt wenn neue Nachrichten vorhanden sind.

## A.3 UML

Abbildung A.1 auf der nächsten Seite zeigt das Klassendiagramm der Test Applikation, die entsprechenden Source Listings finden sich unter A.4 auf Seite 4.

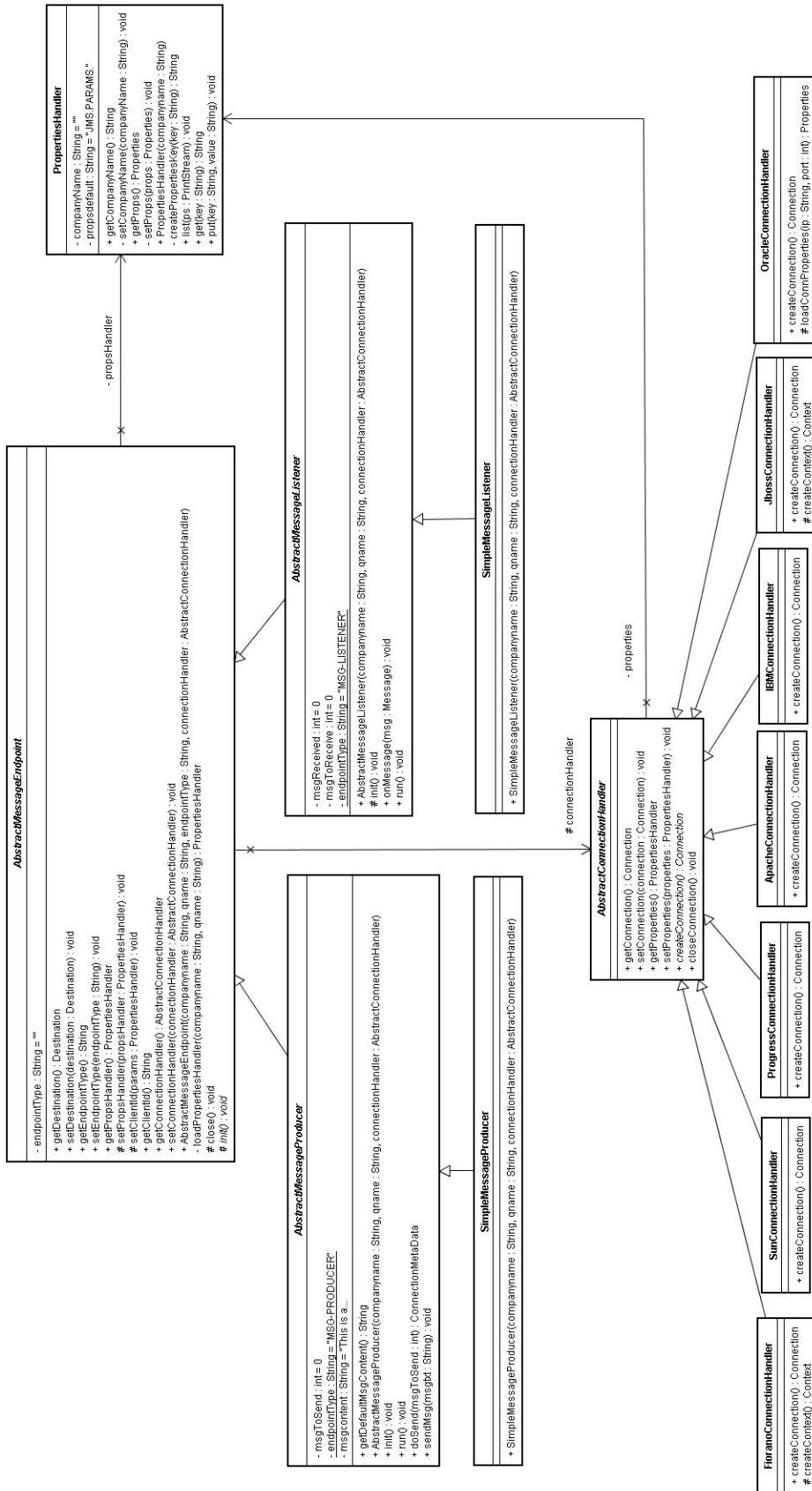


Abbildung A.1: UML der Test Applikation

## A.4 Listings

Hier werden aus Gründen der Nachvollziehbarkeit die einzelnen Java Klassen aufgeführt welche im Rahmen der Evaluierungen verwendet wurden.

### A.4.1 Provider Properties

In der Konfigurationsdatei werden die benötigten Einstellungen für die einzelnen JMS Provider festgelegt. Dies bringt den Vorteil mit sich das Konfigurationsänderungen keinerlei Änderungen an der Source Codes mit sich zieht. Die Konfigurationsdatei wird von einem Properties Handler (siehe Listing A.6 auf Seite 14) eingelesen und weiterverarbeitet.

#### Listing A.1: JMS Provider Properties

```
1 # Apache ActiveMQ
2 JMS.PARAMS.APACHE.PRODUCT=ActiveMQ
3 JMS.PARAMS.APACHE.IP=localhost
4 JMS.PARAMS.APACHE.PORT=61616
5 JMS.PARAMS.APACHE.MSGAMOUNT=10000
6
7 #Sun OpenMQ
8 JMS.PARAMS.SUN.PRODUCT=OpenMQ
9 JMS.PARAMS.SUN.IP=localhost
10 JMS.PARAMS.SUN.PORT=7676
11 JMS.PARAMS.SUN.MSGAMOUNT=10000
12
13 #JBoss Messaging
14 JMS.PARAMS.JBOSS.PRODUCT=Messaging
15 JMS.PARAMS.JBOSS.IP=localhost
16 JMS.PARAMS.JBOSS.PORT=4457
17 JMS.PARAMS.JBOSS.JNDI_PORT=1099
18 JMS.PARAMS.JBOSS.CONNECTION_FACTORY=ConnectionFactory
19 JMS.PARAMS.JBOSS.INITIAL_CONTEXT_FACTORY=org.jnp.interfaces.\
    -NamingContextFactory
20 JMS.PARAMS.JBOSS.URL_PKG_PREFIXES=org.jboss.naming:org.jnp.\
    -interfaces
21 JMS.PARAMS.JBOSS.MSGAMOUNT=10000
22
23 #Progress SonicMQ
24 JMS.PARAMS.PROGRESS.PRODUCT=SonicMQ
25 JMS.PARAMS.PROGRESS.IP=localhost
26 JMS.PARAMS.PROGRESS.PORT=2506
27 JMS.PARAMS.PROGRESS.MSGAMOUNT=10000
28
29 #Fiorano FioranoMQ
30 JMS.PARAMS.FIORANO.PRODUCT=FioranoMQ
31 JMS.PARAMS.FIORANO.IP=localhost
32 JMS.PARAMS.FIORANO.PORT=1856
```

```

33 JMS.PARAMS.FIORANO.BACKUPPORT=1956
34 JMS.PARAMS.FIORANO.MSGAMOUNT=10000
35 JMS.PARAMS.FIORANO.SECURITY_PRINCIPAL=anonymous
36 JMS.PARAMS.FIORANO.SECURITY_CREDENTIALS=anonymous
37 JMS.PARAMS.FIORANO.INITIAL_CONTEXT_FACTORY=fiorano.jms.runtime.\
    -naming.FioranoInitialContextFactory
38
39 #Oracle Advanced Queuing
40 JMS.PARAMS.ORACLE.PRODUCT=Advanced Queuing
41 JMS.PARAMS.ORACLE.IP=localhost
42 JMS.PARAMS.ORACLE.PORT=1521
43 JMS.PARAMS.ORACLE.MSGAMOUNT=10000
44 JMS.PARAMS.ORACLE.DB_AQ_ADMIN_NAME=ssi_aqadm
45 JMS.PARAMS.ORACLE.DB_AQ_ADMIN_PWD=ssi_aqadm
46 JMS.PARAMS.ORACLE.DB_AQ_USER_NAME=ssi_aquser
47 JMS.PARAMS.ORACLE.DB_AQ_USER_PWD=ssi_aquser
48 JMS.PARAMS.ORACLE.SID=PEEM
49
50 #IBM WebsphereMQ
51 JMS.PARAMS.IBM.PRODUCT=WebsphereMQ
52 JMS.PARAMS.IBM.IP=localhost
53 JMS.PARAMS.IBM.PORT=1414
54 JMS.PARAMS.IBM.MSGAMOUNT=10000
55 JMS.PARAMS.IBM.QUEUEMANAGER=qm.ssi.local
56 JMS.PARAMS.IBM.TRANSPORTTYPE=1
57 # TRANSPORTTYPE = 1 = JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
58 JMS.PARAMS.IBM.CHANNEL=SYSTEM.DEF.SVRCONN
59 JMS.PARAMS.IBM.SECURITY_PRINCIPAL=mqm
60 JMS.PARAMS.IBM.SECURITY_CREDENTIALS=P@ssw0rd

```

---

## A.4.2 Abstrakte Klassen

Die abstrakten Klassen definieren gemeinsame Eigenschaften und Funktionen für Message Endpoints (Listing A.2) und dessen abgeleiteten Objekte Message Producer (Listing A.3 auf Seite 8) und Message Listener (Listing A.4 auf Seite 10).

### Listing A.2: JMS Abstract Message Endpoint

```

1 package ssi.jms.simple.general;
2
3 import javax.jms.Destination;
4 import javax.jms.JMSEException;
5 import javax.jms.MessageConsumer;
6 import javax.jms.MessageProducer;
7 import javax.jms.Session;
8
9 /**
10 * AbstractMessageEndpoint defines a unified way to create
11 * MessageProducers and MessageConsumers

```

```

12  *
13  * @author marktl
14  *
15  */
16 public abstract class AbstractMessageEndpoint extends Thread {
17  /*
18   * private/protected member variables
19   */
20  private PropertiesHandler propsHandler = null;
21  private String endpointType = "";
22  protected AbstractConnectionHandler connectionHandler = null;
23  protected Session session = null;
24  protected Destination destination = null;
25  protected MessageProducer producer = null;
26  protected MessageConsumer consumer = null;
27
28  /*
29   * Getters and Setters
30   */
31  public Destination getDestination() {
32      return destination;
33  }
34
35  public void setDestination(Destination destination) {
36      this.destination = destination;
37  }
38
39  public String getEndpointType() {
40      return endpointType;
41  }
42
43  public void setEndpointType(String endpointType) {
44      this.endpointType = endpointType;
45  }
46
47  public PropertiesHandler getPropsHandler() {
48      return propsHandler;
49  }
50
51  protected void setPropsHandler(PropertiesHandler propsHandler) \
52      → {
53      this.propsHandler = propsHandler;
54  }
55
56  protected void setClientId(PropertiesHandler params) {
57      String company = params.getCompanyName();
58      String product = params.get("PRODUCT");
59      String qname = params.get("QNAME");
60      this.endpointType = company + "-" + product + "-" + \
61          → endpointType + "-"
62          + qname;
63  }
64
65  public String getClientId() {
66      return endpointType;
67  }

```

```

66
67     public AbstractConnectionHandler getConnectionHandler() {
68         return connectionHandler;
69     }
70
71     public void setConnectionHandler(AbstractConnectionHandler <
72         connectionHandler) {
73         this.connectionHandler = connectionHandler;
74     }
75
76     /**
77     * Constructor
78     * @param companyname - Name of the JMS Provider Vendor
79     * @param qname - Name of the Queue (Destination) which shall <
80     *               be used
81     * @param endpointType - describe the endpoint type, may be <
82     *                      →MSG-Producer or MSG-Listener
83     * @param connectionHandler - define the connection handler to<
84     *                           → be used {@link ssi.jms.simple.general.<
85     *                           →AbstractConnectionHandler}
86     */
87     public AbstractMessageEndpoint(String companyname, String <
88         qname,
89         String endpointType, AbstractConnectionHandler <
90             connectionHandler) {
91         setEndpointType(endpointType);
92         setPropsHandler(loadPropertiesHandler(companyname, qname));
93         setConnectionHandler(connectionHandler);
94         getConnectionHandler().setProperties(getPropsHandler());
95         setClientId(getPropsHandler());
96         init();
97     }
98
99     /**
100    * Load the Properties handler which reads the properties (eg <
101      →connection data)
102    * from the configuration file
103    * @param companyname - Name of the JMS Provider Vendor, used <
104      →to look up the correct configuration
105    * @param qname - Name of the Queue (Destination) which shall <
106      →be used
107    * @return PropertiesHandler {@link ssi.jms.simple.general.<
108      →PropertiesHandler}
109    */
110    private PropertiesHandler loadPropertiesHandler(String <
111        companyname,
112        String qname) {
113        PropertiesHandler tmpPropsHandler = new PropertiesHandler(<
114            →companyname);
115        tmpPropsHandler.put("QNAME", qname);
116        return tmpPropsHandler;
117    }
118
119    /**
120    * unified method for closing connections
121    */

```

```

109  protected void close() {
110    try {
111      if (producer != null) {
112        producer.close();
113      }
114      if (consumer != null) {
115        consumer.close();
116      }
117      if (session != null) {
118        session.close();
119      }
120    } catch (JMSEException e) {
121      e.printStackTrace();
122    }
123    connectionHandler.closeConnection();
124  }
125
126  /* abstract methods that MUST be implemented */
127  protected abstract void init();
128
129  /**
130   * inner class for logging start and finishing time
131   */
132  protected class ConnectionMetaData {
133    protected long starttime = 0;
134    protected long finishtime = 0;
135    protected long duration = 0;
136  }
137 }

```

---

### Listing A.3: JMS Abstract Message Producer

```

1  package ssi.jms.simple.general;
2
3  import javax.jms.Connection;
4  import javax.jms.JMSEException;
5  import javax.jms.Session;
6  import javax.jms.TextMessage;
7
8  /**
9   * AbstractMessageP a MessageListener that can be used with any
10  * JMS Provider
11  * @author marktl
12  *
13  */
14  public abstract class AbstractMessageProducer extends \
15    -AbstractMessageEndpoint {
16  /**
17   * private member variables
18   */
19  private int msgToSend = 0;
20  private static String endpointType = "MSG-PRODUCER";
21  /*This is the default content for the test messages - \
22   *approximately 200Bytes*/

```

```

22     private String msgcontent = "This is a Test Message!This is a \
23         →Test Message!This is a Test Message!This is a Test Message\
24         →!This is a Test Message!This is a Test Message!This is a \
25         →Test Message!This is a Test Message!This is a Test Message\
26         →!";
27
28     public String getDefaultMsgContent() {
29         return msgcontent;
30     }
31
32     /**
33      * Constructor
34      *
35      * @param companyname - Name of the JMS Provider Vendor
36      * @param qname - Name of the Queue (Destination) which shall \
37      →be used
38      * @param connectionHandler - handler to establish the \
39      →connection to the JMS Provider
40      */
41     public AbstractMessageProducer(String companyname, String \
42         →qname,
43         AbstractConnectionHandler connectionHandler) {
44         super(companyname, qname, endpointType, connectionHandler);
45         msgToSend = Integer.valueOf((getPropsHandler()).get(" \
46             →MSGAMOUNT"));
47     }
48
49     /**
50      * initialize the Message Producer
51      */
52     @Override
53     public void init() {
54         String qname = (getPropsHandler()).get("QNAME");
55         try {
56             Connection connection = getConnectionHandler(). \
57                 →createConnection();
58             connection.start();
59             session = connection.createSession(false, Session. \
60                 →AUTO_ACKNOWLEDGE);
61             setDestination(session.createQueue(qname));
62             producer = session.createProducer(getDestination());
63         } catch (JMSEException e) {
64             e.printStackTrace();
65         }
66     }
67
68     /**
69      * The Message Producer itself is a Thread which is used for
70      * sending a specified amount (msgToSend) of messages to the
71      * given Destination
72      */
73     @Override
74     public void run() {
75         ConnectionMetaData connectionmeta = doSend(msgToSend);
76
77         System.out.println(getClientId() + " finished in "

```

```

68         + connectionmeta.duration + " Milliseconds" + "[" +
69         + connectionmeta.starttime + "|" + connectionmeta.↖
70             →finishtime
71         + "]");;
72     }
73 /**
74 * send the specified amount of messages (msgToSend) to the
75 * given Destination
76 *
77 * @param msgToSend - amount of messages to send
78 * @return
79 */
80 public ConnectionMetaData doSend(int msgToSend) {
81     ConnectionMetaData connmetadata = new ConnectionMetaData();
82
83     try {
84         connmetadata starttime = System.currentTimeMillis();
85         for (int i = 1; i <= msgToSend; i++) {
86             sendMsg(getDefaultMsgContent());
87         }
88         connmetadata.finishtime = System.currentTimeMillis();
89         connmetadata.duration = connmetadata.finishtime
90             - connmetadata.starttime;
91     } catch (Exception e) {
92         e.printStackTrace();
93     } finally {
94         close();
95     }
96     return connmetadata;
97 }
98 /**
99 * Send an actual message to the given Destination
100 *
101 * @param msgtxt - Message Body/Payload
102 * @throws Exception
103 */
104 public void sendMsg(String msgtxt) throws Exception {
105     TextMessage simpleMessage = session.createTextMessage();
106     simpleMessage.setText(msgtxt);
107     producer.send(simpleMessage);
108 }
109 }

```

---

#### Listing A.4: JMS Abstract Message Listener

```

1 package ssi.jms.simple.general;
2
3 import javax.jms.Connection;
4 import javax.jms.JMSEException;
5 import javax.jms.Message;
6 import javax.jms.MessageListener;
7 import javax.jms.Session;
8 import javax.jms.TextMessage;

```

```

9
10 /**
11  * AbstractMessageListener a MessageListener that can be used \
12  *      -with any
13  * JMS Provider
14  *
15  *
16 */
17 public abstract class AbstractMessageListener extends \
18     -AbstractMessageEndpoint
19     implements MessageListener {
20
21 /**
22  * private member variables
23 */
24 private int msgReceived = 0;
25 private int msgToReceive = 0;
26 private static String endpointType = "MSG-LISTENER";
27
28 /**
29  * Constructor
30  *
31  * @param companyname - Name of the JMS Provider Vendor
32  * @param qname - Name of the Queue (Destination) which shall \
33  *      -be used
34  * @param connectionHandler - handler to establish the \
35  *      -Connection to the JMS Provider
36 */
37 public AbstractMessageListener(String companyname, String \
38     -qname,
39     AbstractConnectionHandler connectionHandler) {
40     super(companyname, qname, endpointType, connectionHandler);
41     msgToReceive = Integer.valueOf((getPropsHandler()).get("\
42         -MSGAMOUNT"));
43 }
44
45 /**
46  * initialize the Message Listener
47 */
48 @Override
49 protected void init() {
50     String qname = (super.getPropsHandler()).get("QNAME");
51     try {
52         Connection connection = connectionHandler.createConnection\
53             -();
54         connection.start();
55         session = connection.createSession(false, Session.\
56             -AUTO_ACKNOWLEDGE);
57         setDestination(session.createQueue(qname));
58         consumer = session.createConsumer(getDestination());
59         consumer.setMessageListener(this);
60     } catch (JMSException e) {
61         e.printStackTrace();
62     }
63 }

```

```

57
58  /**
59   * define what happens when a message arrives.
60   * In this case only a counter is increased up to the
61   * defined number of messages (msgToReceive).
62   */
63 @Override
64 public void onMessage(Message msg) {
65     msgReceived = msgReceived + 1;
66     TextMessage txtMsg = (TextMessage) msg;
67     if (msgReceived == msgToReceive) {
68         System.out.println(getClientId() + " Received all msg @ ["
69             + System.currentTimeMillis() + "]");
70         msgReceived = 0;
71     } else if (msgReceived > msgToReceive) {
72         msgReceived = 0;
73     }
74 }
75
76 /**
77  * The Message Listener it self is a thread which can be set \
78  * to wait.
79  */
80 @Override
81 public void run() {
82     /*
83      * do nothing here just wait for new messages - because of \
84      * this the
85      * thread can be set to wait.
86     */
87     try {
88         synchronized (this) {
89             this.wait();
90         }
91     } catch (InterruptedException e) {
92         e.printStackTrace();
93     } finally {
94         close();
95     }
96 }

```

---

### A.4.3 Handler Klassen

Connection Handler (Listing A.5 auf der nächsten Seite) werden dazu verwendet die Verbindung zu einem JMS Provider herzustellen und diese in Folge zu überwachen und zu trennen.

Properties Handler (Listing A.6 auf Seite 14) werden dazu verwendet die Parameter aus der Konfigurationsdatei (siehe Listing A.1 auf Seite 4) zu laden und die entsprechenden Parameter sowohl an den verwendeten Connection

Handler (siehe Listing A.5) als auch an den Message Endpoint (siehe Listing A.2 auf Seite 5) weiterzuleiten.

#### Listing A.5: JMS Abstract Connection Handler

```
1 package ssi.jms.simple.general;
2
3 import javax.jms.Connection;
4 import javax.jms.JMSEException;
5
6 /**
7  * AbstractConnectionHandler used to define the way a connection \
8  * → between the
9  * messaging provider and a client is established
10 *
11 * @author marktl
12 */
13 public abstract class AbstractConnectionHandler {
14     /*
15      * Private member variables
16      */
17     private Connection connection = null;
18     private PropertiesHandler properties = null;
19
20     /*
21      * Getters and Setters
22      */
23     public Connection getConnection() {
24         return connection;
25     }
26
27     public void setConnection(Connection connection) {
28         this.connection = connection;
29     }
30
31     public PropertiesHandler getProperties() {
32         return properties;
33     }
34
35     public void setProperties(PropertiesHandler properties) {
36         this.properties = properties;
37     }
38
39     /**
40      * The method createConnection defines how the connection \
41      * →between server and
42      * client is established
43      *
44      * @return Connection {@link javax.jms.Connection}
45      */
46     public abstract Connection createConnection();
47
48     /**
49      * default method for closing a connection
50      */
51 }
```

```

49     public void closeConnection() {
50         try {
51             if (connection != null) {
52                 connection.close();
53             }
54         } catch (JMSEException jmse) {
55             System.out.println("Could not close connection " + \
56                                 connection
57                         + " exception was " + jmse);
58         }
59     }
60 }
```

---

### Listing A.6: JMS Properties Handler

```

1 package ssi.jms.simple.general;
2
3 import java.io.FileInputStream;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6 import java.io.PrintStream;
7 import java.util.Properties;
8
9 /**
10  * PropertiesHandler reads the properties from the configuration
11  * file and passes it to the connection handlers and the
12  * message endpoints
13  *
14  * @author marktl
15  *
16  */
17 public class PropertiesHandler {
18     /*
19      * private member variables
20     */
21     private Properties props = null;
22     private String companyName = "";
23     private String propsdefault = "JMS.PARAMS.";
24
25     /*
26      * Getters and Setters
27     */
28     public String getCompanyName() {
29         return companyName;
30     }
31
32     private void setCompanyName(String companyName) {
33         this.companyName = companyName;
34     }
35
36     public Properties getProps() {
37         return props;
38     }
39 }
```

```

40     private void setProps(Properties props) {
41         this.props = props;
42     }
43
44     /**
45      * Constructor
46      *
47      * @param companyname - Name of the JMS Provider Vendor
48      */
49     public PropertiesHandler(String companyname) {
50         Properties tmp = new Properties();
51         try {
52             String path = System.getProperty("user.dir")
53                 + System.getProperty("file.separator") + "config"
54                 + System.getProperty("file.separator")
55                 + "params.properties";
56             tmp.load(new FileInputStream(path));
57         } catch (FileNotFoundException e) {
58             e.printStackTrace();
59         } catch (IOException e) {
60             e.printStackTrace();
61         }
62         setProps(tmp);
63         setCompanyName(companyname);
64     }
65
66     /**
67      * create a valid key for the properties
68      *
69      * @param key - detail data for the key
70      * @return String - valid key
71      */
72     private String createPropertiesKey(String key) {
73         return propsdefault + getCompanyName() + "." + key;
74     }
75
76     /**
77      * print the properties to the given stream
78      *
79      * @param ps - PrintStream used for displaying the properties
80      */
81     public void list(PrintStream ps) {
82         (getProps()).list(ps);
83     }
84
85     /**
86      * read value from the properties file
87      *
88      * @param key - key which should be looked up
89      *
90      * @return String - value
91      */
92     public String get(String key) {
93         return (String) (getProps()).get(createPropertiesKey(key));
94     }
95

```

```

96  /**
97   * add data (key/value pair) to the properties
98   *
99   * @param key
100  * @param value
101 */
102 public void put(String key, String value) {
103     getProps().put(createPropertiesKey(key), value);
104 }
105
106 }
```

---

## A.4.4 Implementationen Message Endpoint

Message Endpoints können entweder als Message Listener (Listing A.7) oder als Message Producer (Listing A.8 auf der nächsten Seite) ausgeführt werden.

**Listing A.7: JMS Message Producer**

```

1 package ssi.jms.simple.general;
2
3 /**
4  * A simple implementation of an Message Producer
5  *
6  * @author marktl
7 */
8 public class SimpleMessageProducer extends \
    AbstractMessageProducer {
9
10 /**
11  * Constructor
12  *
13  * @param companyname
14  *          - Name of the JMS Provider Vendor
15  * @param qname
16  *          - Name of the Queue (Destination) which shall be \
    used
17  * @param connectionHandler
18  *          - handler for establishing the connection to the \
    JMS Provider
19 */
20 public SimpleMessageProducer(String companyname, String qname,
21     AbstractConnectionHandler connectionHandler) {
22     super(companyname, qname, connectionHandler);
23     System.out
24         .println("Started MessageProducer [" + getClientId() \
    + "] on Queue ["
25         + (super.getPropsHandler()).get("QNAME") + "]");
26     }
27 }
28 }
```

---

#### Listing A.8: JMS Message Listener

```
1 package ssi.jms.simple.general;
2
3 /**
4  * A simple implementation of an JMS message listener
5  *
6  * @author marktl
7  */
8 public class SimpleMessageListener extends \
9      AbstractMessageListener {
10
11    /**
12     * Constructor
13     *
14     * @param companyname
15     *          - Name of the JMS Provider Vendor
16     * @param qname
17     *          - Name of the Queue (Destination) which shall be \
18     *          → used
19     * @param connectionHandler
20     *          - handler for establishing the connection to the \
21     *          → JMS Provider
22     */
23
24  public SimpleMessageListener(String companyname, String qname,
25      AbstractConnectionHandler connectionHandler) {
26    super(companyname, qname, connectionHandler);
27    System.out
28        .println("Started MessageListener [" + getClientId()
29        + "] on Queue ["
30        + (super.getPropsHandler()).get("QNAME") + "]");
31  }
32
33 }
```

## A.4.5 Start Klassen

Die Klassen StartMultipleMessageListeners (Listing A.9) und StartMultipleMessageProducers (Listing A.10 auf Seite 19) wurden verwendet um die einzelnen Testserien zu starten. Die Anzahl der zu startenden Instanzen kann über eine private Variable definiert werden.

#### Listing A.9: Start multiple Message Listeners

```
1 package ssi.jms.simple.general.starttests;
2
3 import java.lang.reflect.Constructor;
4
5 import ssi.jms.simple.general.AbstractConnectionHandler;
6 import ssi.jms.simple.general.AbstractMessageListener;
```

```

7  import ssi.jms.simple.general.SimpleMessageListener;
8
9 /**
10 * start any given number of message listeners
11 *
12 * @author marktl
13 */
14 public class StartMultipleMessageListeners {
15
16 /*
17 * private member variables
18 */
19 private static int recvrsToCreate = 1;
20 private static String companyname = "SUN";
21 private static String packageUrl = "ssi.jms.simple.p2p";
22 private static String defaultClassName = "ConnectionHandler";
23
24 /*
25 * valid values for companyname are: - APACHE - SUN - JBOSS - \
26 * PROGRESS -
27 * FIORANO - ORACLE - IBM
28 */
29
30 @SuppressWarnings("unchecked")
31 public static void main(String[] args) {
32     for (int i = 0; i < recvrsToCreate; i++) {
33         String qname = "qname_" + i;
34         AbstractMessageListener listener = null;
35         AbstractConnectionHandler connectionHandler = null;
36         try {
37             Class<AbstractConnectionHandler> clazz = (Class< \
38                     AbstractConnectionHandler>) Class
39                     .forName(getClassUrl());
40             Constructor<AbstractConnectionHandler> constructor = \
41                     clazz
42                     .getConstructor();
43             connectionHandler = constructor.newInstance();
44             listener = new SimpleMessageListener(companyname, qname,
45                     connectionHandler);
46         } catch (Exception e) {
47             e.printStackTrace();
48         }
49         listener.start();
50     }
51 /**
52 * create the class url for reflection (class loading)
53 *
54 * @return String - url
55 */
56 private static String getClassUrl() {
57     return packageUrl + "." + companyname.toLowerCase() + "."
58         + getClassName();
59 }
```

```

60  /**
61   * recreate class name - include the company name starting \
62   *      →with upper case
63   * lowering the rest
64   *
65   * @return String - class name
66   */
67  private static String getClassName() {
68      String tmp = "";
69      tmp = companynname.substring(0, 1)
70          + (companynname.substring(1, companynname.length()))
71          .toLowerCase() + defaultClassName;
72      return tmp;
73  }

```

---

### Listing A.10: Start multiple Message Producers

```

1  package ssi.jms.simple.general.starttests;
2
3  import java.lang.reflect.Constructor;
4  import java.util.HashMap;
5
6  import ssi.jms.simple.general.AbstractConnectionHandler;
7  import ssi.jms.simple.general.AbstractMessageProducer;
8  import ssi.jms.simple.general.SimpleMessageProducer;
9
10 /**
11  * start any given number of message producers
12  *
13  * @author marktl
14  */
15 public class StartMultipleMessageProducers {
16
17  /*
18   * private member variables
19   */
20  private static int producersToCreate = 1;
21  private static String companynname = "SUN";
22  private static String packageUrl = "ssi.jms.simple.p2p";
23  private static String defaultClassName = "ConnectionHandler";
24  private static HashMap<String, AbstractMessageProducer> \
25  →producers = null;
26
27  /*
28  * valid values for companynname are: - APACHE - SUN - JBOSS - \
29  * →PROGRESS -
30  * FIORANO - ORACLE - IBM
31  */
32
33  @SuppressWarnings("unchecked")
34  public static void main(String[] args) {
35      producers = new HashMap<String, AbstractMessageProducer>();
36      for (int i = 0; i < producersToCreate; i++) {
37          String qname = "qname_" + i;

```

```

36     AbstractMessageProducer producer = null;
37     AbstractConnectionHandler connectionHandler = null;
38     try {
39         Class<AbstractConnectionHandler> clazz = (Class<\  

40             →AbstractConnectionHandler>) Class  

41             .forName(getClassUrl());
42         Constructor<AbstractConnectionHandler> constructor = \  

43             →clazz  

44             .getConstructor();
45         connectionHandler = constructor.newInstance();
46         producer = new SimpleMessageProducer(companyname, qname,  

47             connectionHandler);
48     } catch (Exception e) {  

49         e.printStackTrace();
50     }
51     addSender(qname, producer);
52 }
53
54 /**
55 * create the class url for reflection (class loading)
56 *
57 * @return String - url
58 */
59 private static String getClassUrl() {
60     return packageUrl + "." + companyname.toLowerCase() + "."
61         + getClassName();
62 }
63
64 /**
65 * recreate class name - include the company name starting \  

66     →with upper case
67 * lowering the rest
68 *
69 * @return String - class name
70 */
71 private static String getClassName() {
72     String tmp = "";
73     tmp = companyname.substring(0, 1)
74         + (companyname.substring(1, companyname.length()))
75             .toLowerCase() + defaultClassName;
76     return tmp;
77 }
78
79 /**
80 * add a message producer to the list of managed message \  

81     →producers
82 *
83 * @param qname
84 *      - Name of the Queue (Destination) which shall be \  

85     → used
86 * @param producer
87 *      - an instance of Message producer
88 */
89 private static void addSender(String qname, \  


```

```

    ↗AbstractMessageProducer producer) {
87     producers.put(qname, producer);
88 }
89
90 /**
91  * send messages from all managed message producers
92 */
93 private static void sendAll() {
94     System.out.println("Started transaction @ ["
95         + System.currentTimeMillis() + "]");
96     for (String qname : producers.keySet()) {
97         AbstractMessageProducer msgProducer = producers.get(qname) ↗
98             ;
99         msgProducer.start();
100    }
101 }

```

---

## A.4.6 Implementationen Connection Handler

Wie aus der UML Darstellung (siehe Abbildung A.1 auf Seite 3) hervorgeht werden in allen Fällen die selben Message Producer und Message Listener verwendet welche sich lediglich darin unterscheiden welches ConnectionHandler Objekt sie übergeben bekommen. Dadurch wird ein hoher Grad an Flexibilität erreicht und die Wiederverwendbarkeit des Source Codes maximiert.

**Listing A.11: Apache ActiveMQ Connection Handler**

```

1 package ssi.jms.simple.p2p.apache;
2
3 import javax.jms.Connection;
4 import javax.jms.JMSException;
5
6 import org.apache.activemq.ActiveMQConnectionFactory;
7
8 import ssi.jms.simple.general.AbstractConnectionHandler;
9
10 /**
11  * Connection Handler that establishes a connection to Apache ↗
12  * →ActiveMQ
13  *
14  * @author marktl
15 */
16 public class ApacheConnectionHandler extends ↗
17     AbstractConnectionHandler {
18
19     @Override
20     public Connection createConnection() {
21         String ip = (getProperties()).get("IP");
22         int port = Integer.valueOf((getProperties()).get("PORT"));

```

```

21
22     String connectstring = "tcp://" + ip + ":" + port;
23     ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory\  

24         (
25             connectstring);
26     Connection conn = null;
27
28     try {
29         conn = cf.createConnection();
30     } catch (JMSEException jmse) {
31         jmse.printStackTrace();
32     }
33     setConnection(conn);
34     return conn;
35 }
36 }
```

---

#### Listing A.12: Fiorano FioranoMQ Connection Handler

```

1 package ssi.jms.simple.p2p.fiorano;
2
3 import java.util.Properties;
4
5 import javax.jms.Connection;
6 import javax.jms.QueueConnectionFactory;
7 import javax.naming.Context;
8 import javax.naming.InitialContext;
9
10 import ssi.jms.simple.general.AbstractConnectionHandler;
11
12 /**
13  * Connection Handler that establishes a connection to Fiorano \
14  * →FioranoMQ
15  *
16  * @author marktl
17  */
18 public class FioranoConnectionHandler extends \
19     →AbstractConnectionHandler {
20
21     private InitialContext ctx;
22
23     @Override
24     public Connection createConnection() {
25         try {
26             ctx = (InitialContext) createContext();
27         } catch (Exception e) {
28             e.printStackTrace();
29         }
30
31         QueueConnectionFactory qcf = null;
32         Connection conn = null;
33         try {
34             qcf = (QueueConnectionFactory) ctx.lookup("PRIMARYQCF");
35             conn = qcf.createConnection();
36 }
```

```

34         } catch (Exception e) {
35             e.printStackTrace();
36         }
37         return conn;
38     }
39
40     /**
41      * create a context that will be used during connection \n
42      * establishment
43      *
44      * @return Context
45      * @throws Exception
46      */
47     protected Context createContext() throws Exception {
48         String ip = (getProperties()).get("IP");
49         int port = Integer.valueOf((getProperties()).get("PORT"));
50         int backupPort = Integer.valueOf((getProperties()).get("\n"
51             "BACKUPPORT"));
52         String host = "http://" + ip + ":" + port;
53         String backuphost = "http://" + ip + ":" + backupPort;
54
55         Properties env = new Properties();
56         env.put(Context.SECURITY_PRINCIPAL, (getProperties())
57             .get("SECURITY_PRINCIPAL"));
58         env.put(Context.SECURITY_CREDENTIALS, (getProperties())
59             .get("SECURITY_CREDENTIALS"));
60         env.put(Context.PROVIDER_URL, host);
61         env.put(Context.INITIAL_CONTEXT_FACTORY, (getProperties())
62             .get("INITIAL_CONTEXT_FACTORY"));
63         env.put("BackupConnectURLs", backuphost);
64
65         return new InitialContext(env);
66     }
67 }

```

---

### Listing A.13: IBM WebsphereMQ Connection Handler

```

1 package ssi.jms.simple.p2p.ibm;
2
3 import javax.jms.Connection;
4 import javax.jms.JMSException;
5
6 import ssi.jms.simple.general.AbstractConnectionHandler;
7
8 import com.ibm.mq.jms.MQQueueConnectionFactory;
9
10 /**
11  * Connection Handler that establishes a connection to IBM \n
12  * WebsphereMQ
13  *
14  * @author markt1
15  */
16 public class IbmConnectionHandler extends \n
17     AbstractConnectionHandler {

```

```

17    @Override
18    public Connection createConnection() {
19        javax.jms.ConnectionFactory factory = null;
20        Connection conn = null;
21        String ip = (getProperties()).get("IP");
22        int port = Integer.valueOf((getProperties()).get("PORT"));
23        String principal = (getProperties()).get("SECURITY_PRINCIPAL" +
24            "→");
25        String credentials = (getProperties()).get("→
26            SECURITY_CREDENTIALS");
27        String channel = (getProperties()).get("CHANNEL");
28        int transportType = Integer.valueOf((getProperties())
29            .get("TRANSPORTTYPE"));
30        String qm = (getProperties()).get("QUEUEMANAGER");
31
32        factory = new MQQueueConnectionFactory();
33        ((MQQueueConnectionFactory) factory).setHostName(ip);
34        try {
35            ((MQQueueConnectionFactory) factory).setPort(port);
36
37            ((MQQueueConnectionFactory) factory)
38                .setTransportType(transportType);
39            ((MQQueueConnectionFactory) factory).setQueueManager(qm);
40            ((MQQueueConnectionFactory) factory).setChannel(channel);
41            conn = ((MQQueueConnectionFactory) factory).→
42                →createConnection(
43                    principal, credentials);
44        } catch (JMSEException e) {
45            e.printStackTrace();
46        }
47        return conn;
48    }
49}

```

---

#### Listing A.14: JBoss Messaging Connection Handler

```

1 package ssi.jms.simple.p2p.jboss;
2
3 import java.util.Properties;
4
5 import javax.jms.Connection;
6 import javax.jms.ConnectionFactory;
7 import javax.naming.Context;
8 import javax.naming.InitialContext;
9
10 import ssi.jms.simple.general.AbstractConnectionHandler;
11
12 /**
13  * Connection Handler that establishes a connection to JBoss →
14  * Messaging
15  * @author marktl
16  */

```

```

17 public class JbossConnectionHandler extends \
   -AbstractConnectionHandler {
18     private InitialContext ctx;
19
20     @Override
21     public Connection createConnection() {
22         String factory = (getProperties()).get("CONNECTION_FACTORY") \
           →;
23         Connection conn = null;
24         try {
25             ctx = (InitialContext) createContext();
26             ConnectionFactory cf = (ConnectionFactory) ctx.lookup( \
               →factory);
27             conn = cf.createConnection();
28         } catch (Exception e) {
29             e.printStackTrace();
30         }
31         setConnection(conn);
32         return conn;
33     }
34
35     /**
36     * create a context which will be used during connection \
       →establishment
37     *
38     * @return Context
39     * @throws Exception
40     */
41     protected Context createContext() throws Exception {
42         String ip = (getProperties()).get("IP");
43         int jndiport = Integer.valueOf((getProperties()).get(" \
           →JNDI_PORT"));
44         String host = "jnp://" + ip + ":" + jndiport;
45
46         Properties props = new Properties();
47         props.put(Context.INITIAL_CONTEXT_FACTORY, (getProperties()) \
           .get("INITIAL_CONTEXT_FACTORY"));
48         props.put(Context.PROVIDER_URL, host);
49         props.put(Context.URL_PKG_PREFIXES, (getProperties()) \
           .get("URL_PKG_PREFIXES"));
50
51         return new InitialContext(props);
52     }
53 }

```

---

#### Listing A.15: Oracle Advanced Queuing Connection Handler

```

1 package ssi.jms.simple.p2p.oracle;
2
3 import java.util.Properties;
4
5 import javax.jms.Connection;
6 import javax.jms.QueueConnectionFactory;
7
8 import oracle.jms.AQjmsFactory;

```

```

9  import ssi.jms.simple.general.AbstractConnectionHandler;
10
11 /**
12  * Connection Handler that establishes a connection to Oracle \
13  * Advanced Queuing
14  *
15  */
16 public class OracleConnectionHandler extends \
17     AbstractConnectionHandler {
18
19     @Override
20     public Connection createConnection() {
21         QueueConnectionFactory qcf = null;
22         Connection conn = null;
23         try {
24             String ip = (getProperties()).get("IP");
25             int port = Integer.valueOf((getProperties()).get("PORT"));
26             String sid = (getProperties()).get("SID");
27             String aq_user_name = (getProperties()).get("\
28                 DB_AQ_USER_NAME");
29             String aq_user_pwd = (getProperties()).get("DB_AQ_USER_PWD\
30                 ");
31
32             Properties usr = new Properties();
33             String host = "jdbc:oracle:thin:@" + ip + ":" + port + ":"\
34                 + sid;
35             usr.put(aq_user_name, aq_user_pwd);
36
37             qcf = AQjmsFactory.getQueueConnectionFactory(host, usr);
38
39             conn = qcf.createQueueConnection(aq_user_name, aq_user_pwd\
40                 );
41         } catch (Exception e) {
42             e.printStackTrace();
43         }
44         return conn;
45     }
46 }

```

---

#### Listing A.16: Progress SonicMQ Connection Handler

```

1 package ssi.jms.simple.p2p.progress;
2
3 import javax.jms.Connection;
4 import javax.jms.JMSEException;
5
6 import ssi.jms.simple.general.AbstractConnectionHandler;
7
8 /**
9  * Connection Handler that establishes a connection to Progress \
10  * SonicMQ
11  *
12  */

```

```

12  /*
13  public class ProgressConnectionHandler extends \
14      AbstractConnectionHandler {
15
16  @Override
17  public Connection createConnection() {
18      String ip = (getProperties()).get("IP");
19      int port = Integer.valueOf((getProperties()).get("PORT"));
20      String principal = (getProperties()).get("SECURITY_PRINCIPAL" \
21          );
22      String credentials = (getProperties()).get(" \
23          SECURITY_PRINCIPAL");
24      javax.jms.ConnectionFactory factory = null;
25      String host = "";
26      Connection conn = null;
27      try {
28          host = "tcp://" + ip + ":" + port;
29          factory = new progress.message.jclient.ConnectionFactory( \
30              host);
31          conn = factory.createConnection(principal, credentials);
32      } catch (JMSEException e) {
33          e.printStackTrace();
34      }
35      return conn;
36  }
37 }

```

---

### Listing A.17: Sun OpenMQ Connection Handler

```

1 package ssi.jms.simple.p2p.sun;
2
3 import javax.jms.Connection;
4 import javax.jms.JMSEException;
5
6 import com.sun.messaging.ConnectionConfiguration;
7 import com.sun.messaging.ConnectionFactory;
8
9 import ssi.jms.simple.general.AbstractConnectionHandler;
10
11 /**
12  * Connection Handler that establishes a connection to Sun \
13  * -OpenMQ
14  *
15  * @author markt
16  */
17 public class SunConnectionHandler extends \
18     AbstractConnectionHandler {
19
20     @Override
21     public Connection createConnection() {
22         String ip = (getProperties()).get("IP");
23         int port = Integer.valueOf((getProperties()).get("PORT"));
24
25         String connectstring = ip + ":" + port;
26         ConnectionFactory cf = new ConnectionFactory();

```

```

25     Connection conn = null;
26     try {
27         cf.setProperty(ConnectionConfiguration.imqAddressList,
28             connectstring);
29         cf.setProperty(ConnectionConfiguration.imqReconnectEnabled\-
30             , "true");
31         conn = cf.createConnection();
32     } catch (JMSEException e) {
33         e.printStackTrace();
34     }
35     setConnection(conn);
36     return conn;
37 }
38 }
```

---

## A.5 Kommentare zu den Bewertungen

### A.5.1 Apache ActiveMQ

Kommentare	
<b>Bearbeitungsdauer</b>	Bei einer Messagegröße von rund 200Byte, und einem Sender-/Empfänger Paar, dauert die Übertragung von 10.000 Messages rund 6 Sekunden, dh die berechnete Bearbeitungsdauer einer Message liegt bei 0,6ms. Da der gewünschte Wert bei 1ms liegt kann hier die Höchstbewertung vergeben werden.
<b>Durchsatz</b>	Der Mittelwert des Durchsatzes lag im Rahmen der Tests bei 1.547 Nachrichten pro Sekunde. Da der gewünschte Wert bei 1.000 Messages pro Sekunde liegt kann hier die Höchstbewertung vergeben werden.
<b>Priorisierung</b>	Wird standardmäßig nicht unterstützt, kann jedoch unter Verwendung von <a href="http://camel.apache.org/resequencer.html">http://camel.apache.org/resequencer.html</a> integriert werden. Da die Priorisierung nicht nativ durch das System angeboten wird, muss eine Bewertung mit 2 erfolgen.

Fortsetzung auf der nächsten Seite

**Tabelle A.1 – Fortsetzung**

Kommentare	
<b>Persistierung</b>	Wird unterstützt, es kommt standardmäßig der AMQ Store ( <a href="http://activemq.apache.org/amq-message-store.html">http://activemq.apache.org/amq-message-store.html</a> ) zum Einsatz, es jedoch auch möglich alle gängigen Datenbanken (Apache Derby, DB2, HSQL, Informix, MySQL, Oracle, Postgresql, SQLServer, Sybase) via JDBC zu verwenden.
<b>Logging</b>	Ist vorhanden, standardmäßig kommt commons-logging zum Einsatz, es kann aber auch log4j verwendet werden.
<b>Verfügbarkeit</b>	Durch Clustering kann eine durchgehende Verfügbarkeit erreicht werden. Wird kein Clustering verwendet so sinkt die Verfügbarkeit entsprechend, kann jedoch immer noch als ausreichend bewertet werden.
<b>Skalierbarkeit</b>	Die Bearbeitungsdauer und der Durchsatz auf Ebene der einzelnen Queues unterliegt Schwankungen, betrachtet man jedoch den Gesamtdurchsatz des Systems über alle Sender/Empfänger/Queues gerechnet so lässt sich feststellen das sowohl Bearbeitungsdauer als auch Durchsatz annähernd konstant bleiben.
<b>Sprachunabhängigkeit</b>	Java wird zu 100% unterstützt, plsql (Oracle) verfügt über die Möglichkeit JMS Messages zu versenden und wird insofern auch unterstützt, Perl wird unterstützt. Es können diverse weitere Sprachen verwendet werden, diese können zum Beispiel über Stomp oder OpenWire angebunden werden.
<b>Abhängigkeit zu Fremdsystemen</b>	Neben der Verwendung von Stomp (ist bereits in ActiveMQ integriert), gibt es keinerlei weitere Abhängigkeiten zu Fremdsystemen.
<b>Wartbarkeit</b>	Sehr einfache Installation, es ist lediglich notwendig das Archiv zu entpacken und den Server zu starten. Der Gesamte Installationsvorgang nimmt weniger als 10 Minuten in Anspruch. Für einfache Verwaltungs-/Monitoringaufgaben stellt ActiveMQ ein Web-Interface zur Verfügung.
<b>UNIX</b>	Unterstützt sowohl IBM AIX als auch SuSe Linux, es werden darüber hinaus auch Microsoft sowie Apple Betriebssysteme unterstützt.
<b>LAN</b>	Siehe 3.1.2 auf Seite 60.
<b>Marktposition</b>	Wird von diversen Unternehmen sowie von Universitäten eingesetzt. Findet auch vielfältige Verwendung in anderen Projekten, dh ActiveMQ wird oftmals als Basis für andere Projekte herangezogen.

Fortsetzung auf der nächsten Seite

**Tabelle A.1 – Fortsetzung**

<b>Kommentare</b>	
<b>Schulungen</b>	Werden nur durch Drittfirmen angeboten zB <a href="http://www.springsource.com/">http://www.springsource.com/</a>
<b>Support</b>	Es werden diverse Mailinglisten und Foren angeboten welche kostenfrei zur Verfügung stehen. Professioneller Support wird nur durch Drittfirmen angeboten zB <a href="http://www.springsource.com/">http://www.springsource.com/</a>
<b>Bearbeitungsrecht</b>	Es darf der Source eingesehen und geändert werden, das Produkt steht unter Apache v2 Lizenz.

Tabelle A.1: Kommentare zur Bewertung von: Apache ActiveMQ

## A.5.2 Sun OpenMQ

Kommentare	
<b>Bearbeitungsdauer</b>	Die Bearbeitungsdauer lag bei allen Test weit unterhalb 1ms pro Message und in diesem Sinne kann die Höchstbewertung vergeben werden
<b>Durchsatz</b>	Im Laufe der Test lag der geringste gemessene Durchsatz bei rund 1300 Messages pro Sekunde
<b>Priorisierung</b>	OpenMQ unterstützt nativ die Priorisierung von Nachrichten innerhalb einer Queue
<b>Persistierung</b>	OpenMQ kann sowohl in einen integrierten Message Store als auch via JDBC in diverse Datenbanken persistieren
<b>Logging</b>	Ist vorhanden.
<b>Verfügbarkeit</b>	Die Verfügbarkeit des Systems war im Laufe der Tests hervorragend. In diesem Sinne kann die Höchstbewertung vergeben werden
<b>Skalierbarkeit</b>	Sowohl die Bearbeitungsdauer als auch der Durchsatz unterlagen nur geringen Schwankungen, somit kann die Höchstbewertung vergeben werden.
<b>Sprachunabhängigkeit</b>	Java wird nativ unterstützt, Oracle verfügt über die Möglichkeit JMS Nachrichten zu versenden und Perl kann via Stomp angebunden werden
<b>Abhängigkeit zu Fremdsystemen</b>	Als Abhängigkeit kann Stomp gewertet werden, ansonsten sind keinerlei Abhängigkeiten bekannt.
<b>Wartbarkeit</b>	OpenMQ kann sowohl über command line befehle als auch über eine mitgelieferte grafische Administrationskonsole verwaltet werden.
<b>UNIX</b>	Es werden diverse Plattformen unterstützt unter anderem auch IBM AIX und Linux bzw. werden generische UNIX Versionen von OpenMQ zur Verfügung gestellt
<b>LAN</b>	Siehe 3.1.2 auf Seite 60.
<b>Marktposition</b>	Sun stellt ein starkes Unternehmen dar, das neben der Entwicklung der Java API auch intensiv Enterprise als auch Community Projekte forciert.
<b>Schulungen</b>	Es werden kostenpflichtige Schulungen angeboten
<b>Support</b>	Es gibt Mailinglisten und Foren welche kostenfrei zu nutzen sind, ebenso kann ein gesonderter Supportvertrag geschlossen werden.

Fortsetzung auf der nächsten Seite

**Tabelle A.2 – Fortsetzung**

<b>Kommentare</b>	
<b>Bearbeitungsrecht</b>	Der Sourcecode darf in jeglicher Hinsicht den aktuellen Bedürfnissen angepasst werden.

Tabelle A.2: Kommentare zur Bewertung von: Sun OpenMQ

### A.5.3 Progress SonicMQ

Kommentare	
<b>Bearbeitungsdauer</b>	Die Bearbeitungsdauer lag durchwegs unter 1ms.
<b>Durchsatz</b>	Der Durchsatz lag durchwegs über 1500
<b>Priorisierung</b>	SonicMQ unterstützt die Verwendung von JMS Priority Flags.
<b>Persistierung</b>	Standardmäßig werden Messages in einen internen Message-Store persistiert, es können jedoch via JDBC auch externe Datenbanken unter anderen auch Oracle verwendet werden.
<b>Logging</b>	SonicMQ bietet Möglichkeiten den Dienst vollständig zu überwachen.
<b>Verfügbarkeit</b>	Der Hersteller verspricht eine durchgehende Verfügbarkeit des Messagingsystems durch den Einsatz einer Real-Time Replikation aller Transaktionen auf mehrere Server. Im Laufe der Tests konnten keinerlei Verfügbarkeitsprobleme ohne diese Replikation festgestellt werden.
<b>Skalierbarkeit</b>	SonicMQ unterliegt keinen markanten Schwankungen im Bereich der Bearbeitungsdauer bzw. des Durchsatzes bei steigender Anzahl der Clients.
<b>Sprachunabhängigkeit</b>	Es werden nativ Java und C++/C# Clients angeboten, via stomp lassen sich auch Perl Clients verwenden.
<b>Abhängigkeit zu Fremdsystemen</b>	Neben Stomp sind keinerlei weitere Abhängigkeiten bekannt.
<b>Wartbarkeit</b>	Die Installation lässt sich sehr einfach über einen Installationswizard durchführen, zur Konfiguration und Überwachung des Systems steht eine grafische Console zur Verfügung, ebenso gibt einen Test-Client mit welchem die Installation validiert werden kann. Des Weiteren stehen eine Vielzahl an Unterlagen zur Verfügung.
<b>UNIX</b>	Es werden sowohl AIX als auch SuSe Linux unterstützt.
<b>LAN</b>	Siehe 3.1.2 auf Seite 60.
<b>Marktposition</b>	Progress ist seit Jahren einer der führenden Hersteller von Message Oriented Middleware und wird in diversen Benchmarks an führenden Positionen gelistet. Progress kann eine Vielzahl an Referenzen und Partnern aufweisen.

Fortsetzung auf der nächsten Seite

**Tabelle A.3 – Fortsetzung**

<b>Kommentare</b>	
<b>Schulungen</b>	Werden in den unterschiedlichsten Varianten angeboten zB e-Learning, Unterricht
<b>Support</b>	Es wird umfassender technischer Support angeboten.
<b>Bearbeitungsrecht</b>	Es dürfen keinerlei eigenständige Änderungen an der Software vorgenommen werden.

Tabelle A.3: Kommentare zur Bewertung von: Progress SonicMQ

## A.5.4 Fiorano FioranoMQ

Kommentare	
<b>Bearbeitungsdauer</b>	Die Bearbeitungsdauer lag in allen Test um 0,5ms.
<b>Durchsatz</b>	Der Durchsatz lag in allen Test über 2.000 Nachrichten. Da der gewünschte Wert bei 1.000 Messages pro Sekunde liegt kann hier die Höchstbewertung vergeben werden.
<b>Priorisierung</b>	JMS Priorisierung wird vollständig unterstützt.
<b>Persistierung</b>	Es wird standardmäßig ein interner Message Store verwendet, es können via JDBC jedoch alle gängigen Datenbanken verwendet werden. Entsprechende Beispielkonfigurationen liegen dem Handbuch bei.
<b>Logging</b>	Über das mitgelieferte Management Interface (Fiorano Studio) lassen sich auch die Aufgaben des Loggings abdecken
<b>Verfügbarkeit</b>	Die Verfügbarkeit des Systems kann im Laufe der Test als sehr gut beurteilt werden. Da FioranoMQ in diversen Unternehmen erfolgreich im Einsatz ist kann die Höchstbewertung vergeben werden.
<b>Skalierbarkeit</b>	Das System bietet eine exzellente Skalierbarkeit, dh sowohl Bearbeitungsdauer als auch Durchsatz bleiben bei steigender Zahl an Clients nahezu konstant.
<b>Sprachunabhängigkeit</b>	Java wird nativ unterstützt insofern auch plsql. Perl kann via STOMP angebunden werden.
<b>Abhängigkeit zu Fremdsystemen</b>	Neben der Verwendung von Stomp sind keinerlei weitere Abhängigkeiten bekannt.
<b>Wartbarkeit</b>	Sehr einfach Installation, Die Wartung lässt sich über das mitgelieferte Management Interface sehr einfach und angenehm erledigen.
<b>UNIX</b>	Es wird sowohl AIX als auch Linux unterstützt.
<b>LAN</b>	Siehe 3.1.2 auf Seite 60.
<b>Marktposition</b>	Fiorano ist im Bereich des Messagings ein sehr renommiertes Unternehmen und FioranoMQ wird weltweit bei rund 300 Unternehmen eingesetzt.
<b>Schulungen</b>	Fiorano bietet Schulungen an.
<b>Support</b>	Fiorano bietet technischen Support an.
<b>Code Ownership</b>	Es dürfen keinerlei eigenständige Änderungen durchgeführt werden.

Tabelle A.4: Kommentare zur Bewertung von: Fiorano FioranoMQ