

HIGH-PERFORMANCE NEAR-TIME PROCESSING OF BULK DATA

MARTIN SWIENTEK

**RESEARCH
WITH
PLYMOUTH
UNIVERSITY**

A thesis submitted to the Plymouth University
in partial fulfilment for the degree of
DOCTOR OF PHILOSOPHY

January 2015 – version 0.1

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior consent.

Martin Swientek: *High-Performance Near-Time Processing of Bulk Data*,
© January 2015

ABSTRACT

HIGH-PERFORMANCE NEAR-TIME PROCESSING OF BULK DATA

MARTIN SWIENTEK

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

CONTENTS

I	FIELD OF RESEARCH	1
1	INTRODUCTION	3
1.1	Aims and Objectives of the Research	4
1.2	Research Methodology	4
1.3	Contributions	5
1.4	Outline of the Thesis	5
2	BACKGROUND	7
2.1	Systems for Bulk Data Processing	7
2.2	Batch processing	7
2.2.1	Integration Styles	9
2.2.2	Batch Architectures	9
2.2.3	Batch Performance Optimisations	9
2.3	Message-base processing	9
2.3.1	Integration Styles	10
2.4	Latency vs. Throughput	10
2.4.1	Batch processing	11
2.4.2	Message-based processing	12
2.5	Service-Oriented Architecture	13
2.6	Enterprise Service Bus	14
2.7	Enterprise Integration Patterns	15
2.7.1	Performance relevant Enterprise Integration Patterns (EIPs)	15
2.8	Performance Issues	16
2.8.1	Distributed Architecture	16
2.8.2	Integration of Heterogeneous Technologies	16
2.8.3	Loose Coupling	16
2.9	Current Approaches for Improving the Performance of an SOA Middleware	17
2.9.1	Hardware	17
2.9.2	Compression	19
2.9.3	Service Granularity	19
2.9.4	Degree of Loose Coupling	19
2.10	Summary	20
3	RELATED WORK	21
3.1	Performance of Service-Oriented Systems	21
3.2	Performance Measuring	22
3.3	Performance Optimisation	24
3.4	Self-Adaptive Software Systems	26
3.4.1	Definition	27
3.5	Self-Adaptive Middleware	27
3.6	SLA-Monitoring of Business Processes	29

3.7	Software Performance Engineering	30
3.8	Summary	31
II	CONTRIBUTIONS	33
4	PERFORMANCE EVALUATION OF BATCH AND MESSAGE-BASED SYSTEMS	35
4.1	Introduction	35
4.2	A real world example application	36
4.2.1	Common Architecture	37
4.2.2	Batch prototype	41
4.2.3	Messaging prototype	44
4.3	Performance evaluation	46
4.3.1	Measuring points	46
4.3.2	Instrumentation	49
4.3.3	Test environment	49
4.3.4	Clock Synchronization	50
4.3.5	Preparation and execution of the performance tests	50
4.3.6	Results	52
4.4	Impact of data granularity on throughput and latency	55
4.5	Discussion with respect to related work	59
4.5.1	Performance Modelling	60
4.5.2	Performance Measuring and Evaluation	61
4.6	Summary	63
5	AN ADAPTIVE MIDDLEWARE FOR NEAR-TIME PROCESSING OF BULK DATA	65
5.1	Introduction	65
5.2	Requirements	66
5.3	Middleware Concepts	67
5.3.1	Message Aggregation	67
5.3.2	Message Routing	67
5.3.3	Monitoring and Control	69
5.4	Middleware Components	70
5.5	Design Aspects	70
5.5.1	Usage Scenarios	72
5.5.2	Service Design	74
5.5.3	Integration and Transports	76
5.5.4	Error Handling	76
5.5.5	Controller Design	77
5.6	Prototype Implementation	79
5.6.1	Aggregator	79
5.6.2	Feedback-Control Loop	80
5.6.3	Load Generator	83
5.7	Evaluation	83
5.7.1	Test Environment	84
5.7.2	Test Design	84

5.7.3	Results	84
5.8	Discussion with respect to related work	84
5.8.1	Adaptive Middleware	84
5.8.2	Message Batching	85
5.8.3	Dynamic Scaling	85
5.8.4	Feedback Control of Computing Systems	85
5.9	Summary	86
6	CONCEPTUAL FRAMEWORK	89
6.1	Introduction	89
6.2	Metamodel	91
6.3	Phase	93
6.3.1	Plan	94
6.3.2	Build	95
6.3.3	Run	95
6.4	Roles	96
6.4.1	Business Architect	97
6.4.2	System Architect	98
6.4.3	Software Engineer	99
6.4.4	Test Engineer	100
6.4.5	Operations Engineer	101
6.4.6	Project Manager	103
6.5	Tasks	103
6.5.1	Business Architecture	107
6.5.2	System Architecture	111
6.5.3	Implementation	115
6.5.4	Test	120
6.5.5	Operations	121
6.5.6	Project Management	123
6.6	Processes	125
6.6.1	Implement Integration	125
6.6.2	Implement Aggregation	125
6.6.3	Implement Feedback-Control	128
6.7	Artifacts	128
6.7.1	Performance Requirements	130
6.7.2	Service Interface Definition	131
6.7.3	Aggregation Rules	131
6.7.4	Integration Architecture	132
6.7.5	Routing Rules	132
6.7.6	System Model	132
6.7.7	Controller Configuration	133
6.7.8	Training Concept	133
6.7.9	Staffing Plan	134
6.8	Tools	134
6.8.1	Tools for System Modelling, System Identification and Simulation	136
6.8.2	Tools for Data Visualisation	136

6.8.3	Tools for data processing	136
6.9	Relationship to other Software Development Approaches	136
6.9.1	Rational Unified Process	137
6.9.2	Scrum	140
6.10	Related Work	143
6.10.1	Software Process	143
6.10.2	Software Process Modelling	144
6.10.3	Software Process Modelling using Unified Modeling Language (UML)	145
6.10.4	Software Processes for Adaptive Software Systems	147
6.11	Summary	149
III	CONCLUSION	151
7	CONCLUSION	153
7.1	Summary of the Research project	153
7.2	Contributions	153
7.3	Limitations	153
7.4	Future Work	153
A	SOURCE CODE	155
A.1	Project Structure	155
A.2	Package Structure	155
	BIBLIOGRAPHY	157
	PUBLICATIONS	169

LIST OF FIGURES

Figure 1	A system consisting of several subsystems forming a processing chain	7
Figure 2	Batch processing	8
Figure 3	Message-based processing	9
Figure 4	Batch processing system comprised of three subsystems	11
Figure 5	Message-based system comprised of three subsystems	12
Figure 6	Latency and throughput are opposed to each other	13
Figure 7	Related Work	21
Figure 8	Billing process	36
Figure 9	Components of the billing application prototype	37
Figure 10	The prototypes share the same business components, database and data-access layer.	39
Figure 11	Business services	39
Figure 12	The prototypes use different integration layers.	40
Figure 13	Logical data model of the prototype	41
Figure 14	Batch prototype	42
Figure 15	A Step consists of an item reader, item processor and item writer	43
Figure 16	Message-based prototype	45
Figure 17	Measuring points of the batch prototype	46
Figure 18	Measuring points of the messaging prototype	47
Figure 19	Batch prototype deployment on EC2 instances	50
Figure 20	Messaging prototype deployment on EC2 instances	51
Figure 21	Throughput	52
Figure 22	Latency	53
Figure 23	Overhead batch prototype	53
Figure 24	Overhead messaging prototype	54
Figure 25	System utilisation batch prototype	54
Figure 26	System utilisation messaging prototype	55
Figure 27	The data granularity is controlled by an aggregator	56
Figure 28	Impact of different aggregation sizes on throughput	57
Figure 29	Impact of different aggregation sizes on processing overhead	58

Figure 30	Impact of different aggregation sizes on latency	58
Figure 31	Impact of different aggregation sizes on system utilisation	59
Figure 32	Monitoring and Control	69
Figure 33	Feedback loop to control the aggregation size	70
Figure 34	Usage scenarios	73
Figure 35	Components of the feedback-control loop	80
Figure 36	UML class diagram of the <i>Load Generator</i>	83
Figure 37	Overview of Conceptual Framework	90
Figure 38	Package structure	91
Figure 39	Metamodel	92
Figure 40	Attributes of a phase	93
Figure 41	Attributes of a role	97
Figure 42	Role: Business Architect	97
Figure 43	Role: System Architect	98
Figure 44	Role: Software Engineer	99
Figure 45	Role: Test Engineer	102
Figure 46	Role: Operations Engineer	102
Figure 47	Role: Project Manager	103
Figure 48	Overview of tasks	104
Figure 49	Subpackages of the Tasks package	105
Figure 50	Attributes of a task	106
Figure 51	Tasks extending the definition of the business architecture	108
Figure 52	Tasks extending the definition of the system architecture	111
Figure 53	Tasks of the process Implement Integration	126
Figure 54	UML Activity Diagram: Implement Integration	126
Figure 55	Tasks of the process Implement Aggregation	127
Figure 56	UML Activity Diagram: Implement Aggregation	127
Figure 57	Tasks for implementing the feedback-control loop	128
Figure 58	UML Activity Diagram: Implement Feedback-Control Loop	129
Figure 59	Artifacts	130
Figure 60	Attributes of an artifact	135
Figure 61	Attributes of a tool	135

Figure 62	Core process workflows (Kruchten and Royce, 1996)	138
-----------	---	-----

LIST OF TABLES

Table 1	Main characteristics of an ESB (Chappell, 2004)	14
Table 2	Levels of coupling	18
Table 3	Technologies and frameworks used for the implementation of the prototypes	38
Table 4	Measuring points of the batch prototype	47
Table 5	Measuring points of the messaging prototype	48
Table 6	Amazon EC2 instance configuration	50
Table 7	Properties of different aggregation strategies	68
Table 8	Strategies for message routing	68
Table 9	Components of the Adaptive Middleware. We are using the notation defined by Hohpe and Woolf (2003)	71
Table 10	Options for designing service interfaces	75
Table 11	Transport options for high and low aggregation sizes	76
Table 12	Phase: Plan	94
Table 13	Phase: Build	95
Table 14	Phase: Run	95
Table 15	Business Architect	98
Table 16	System Architect	99
Table 17	Software Engineer	100
Table 18	Test Engineer	100
Table 19	Operations Engineer	101
Table 20	table	103
Table 21	Define Performance Requirements	108
Table 22	Define Service Interfaces	109
Table 23	Define Aggregation Rules	110
Table 24	Define Integration Architecture	111
Table 25	Define Routing Rules	112
Table 26	Define Controller Architecture	113
Table 27	Define Control Problem	114
Table 28	Define Input/Output Variables	114
Table 29	Implement Feedback-Control Loop	115
Table 30	Perform Static Tests	116
Table 31	Perform Step Tests	117
Table 32	Create System Model / Perform System Identification	117
Table 33	Perform Controller Tuning	118

Table 34	Implement Service Interfaces	118
Table 35	Implement Aggregation Rules	119
Table 36	Implement Routing Rules	119
Table 37	Define Performance Tests	120
Table 38	Evaluate Performance Test Results	121
Table 39	Setup Monitoring infrastructure	122
Table 40	Setup Test Environment	122
Table 41	Perform Performance Tests	123
Table 42	Define Training Concept	123
Table 43	Staffing	124
Table 44	Performance Requirements	130
Table 45	Service Interface Definition	131
Table 46	Aggregation Rules	131
Table 47	Integration Architecture	132
Table 48	Routing Rules	132
Table 49	System Model	132
Table 50	Controller Configuration	133
Table 51	Training Concept	133
Table 52	Training Concept	134
Table 53	Mapping of tasks to RUP core workflows	139
Table 54	Example Product Backlog	142

LISTINGS

Listing 1	Mediation batch job definition	43
Listing 2	Mediation batch route definition	43
Listing 3	Rating batch job definition	44
Listing 4	Billing route definition	46
Listing 5	Billing route definition with an additional aggregator	56
Listing 6	Java interface of a web service offering different operations for single and batch processing.	75
Listing 7	UsageEventsAggrationStrategy	79
Listing 8	Aggregator configuration in definition of BillingRoute	80
Listing 9	ControllerStrategy Interface	80
Listing 10	Implementation of PID Controller	81
Listing 11	Actuator Interface	82

Listing 12 AggregateSizeActuator 82

ACRONYMS

API	Application Programming Interface
CDR	Call Detail Records
CSV	Comma Separated Values
DB	Database
EIP	Enterprise Integration Patterns
ESB	Enterprise Service Busx
NCDR	Normalized Call Detail Records
FIFO	First In, First Out
FTP	File Transfer Protocol
JAXB	Java Architecture for XML Binding
JMS	Java Messaging Service
JMX	Java Monitoring Extensions
JPA	Java Persistence API
JSON	JavaScript Object Notation
NTP	Network Time Protocol
ORM	Object-relational mapping
PTP	Precision Time Protocol
PML	Process Modelling Language
QoS	Quality of Service
REST	Representational State Transfer
RUP	Rational Unified Process
SASO	Stable, Accurate, Settling Times, Overshoot
SLA	Service Level Agreements
SOA	Service Oriented Architecture

SPE	Software Performance Engineering
SPEM	Software & System Process Modelling Metamodel
UML	Unified Modeling Language
UML4SPM	UML for Software Process Modelling
XML	Extended Markup Language

ACKNOWLEDGMENTS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

AUTHOR'S DECLARATION

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Graduate Committee.

Relevant scientific seminars and conferences were regularly attended at which work was often presented. Several papers were published in the course of this research project, details of which are listed in the appendices.

Word count of main body of thesis: xxx words

Frankfurt, Germany, January 2015

Martin Swientek

Part I

FIELD OF RESEARCH

INTRODUCTION

Enterprise Systems like customer-billing systems or financial transaction systems are required to process large volumes of data in a fixed period of time. For example, a billing system for a large telecommunication provider has to process more than 1 million bills per day. Those systems are increasingly required to also provide near-time processing of data to support new service offerings.

Traditionally, enterprise systems for bulk data processing are implemented as batch processing systems (Fleck, 1999). Batch processing delivers high throughput but cannot provide near-time processing of data, that is the end-to-end latency of such a system is high. End-to-end latency refers to the period of time that it takes for a business process, implemented by multiple subsystems, to process a single business event. For example, consider the following billing system of a telecommunications provider:

- Customers are billed once per month
- Customers are partitioned in 30 billing groups
- The billing system processes 1 billing group per day, running 24h under full load.

In this case, the mean time for a call event to be billed by the billing system is $1/2$ month. That is, the mean end-to-end latency of this system is $1/2$ month.

A lower end-to-end latency can be achieved by using single-event processing, for example by utilizing a message-oriented middleware for the integration of the services that form the enterprise system. While this approach is able to deliver near-time processing, it is hardly capable for bulk data processing due to the additional communication overhead for each processed message. Therefore, message-based processing is usually not considered for building a system for bulk data processing requiring high throughput.

The processing type is usually a fixed property of an enterprise system that is decided when the architecture of the system is designed, prior to implementing the system. This choice depends on the non-functional requirements of the system. These requirements are not fixed and can change during the lifespan of a system, either anticipated or not anticipated.

Additionally, enterprise systems often need to handle load peaks that occur infrequently. For example, think of a billing system with moderate load over most of the time, but there are certain events

with very high load such as New Year's Eve. Most of the time, a low end-to-end latency of the system is preferable when the system faces moderate load. During the peak load, it is more important that the system can handle the load at all. A low end-to-end latency is not as important as an optimized maximum throughput in this situation.

1.1 AIMS AND OBJECTIVES OF THE RESEARCH

This research project aims to find a solution for the following problem:

How to achieve high-performance near-time processing of bulk data?

Based on this problem, the thesis aims to answer the following research questions:

- **RQ1:** What are the performance characteristics of batch and single-event processing systems?
- **RQ2:** What is the impact of data granularity on end-to-end latency and maximum throughput?
- **RQ3:** tbd.
- **RQ4:** tbd.

To approach these research questions, the research project has the following key objectives:

1. Performance evaluation of batch and messaging systems regarding throughput and latency.
2. Development of a concept for an adaptive middleware that delivers low latency while providing high throughput.
3. Implementation of a research prototype used for demonstrating the practicability of the concept.
4. Specification and conduction of appropriate performance tests to evaluate the developed approach.
5. Development of a conceptional framework containing guidelines and rules for the practitioner how to implement an enterprise system based on the adaptive middleware for near-time processing of bulk data.

1.2 RESEARCH METHODOLOGY

- Evaluation of current approaches to optimize the performance of service-oriented systems.

- To better understand the performance characteristics of batch and message-based processing two prototypes of each processing styles has been implemented.
- Using these prototypes, a performance evaluation has been done to better understand the differences between the two processing styles, with a special focus on the latency and throughput.
- Using the message-based prototype, the impact of data granularity on latency and throughput has been studied.
- Based on the results of the performance evaluation the concept of an adaptive middleware has been developed.
- The message-based prototype has been extending to implement the concepts of the adaptive middleware.
- A performance evaluation has been conducted to evaluate the proposed concepts for an adaptive middleware for bulk data processing.

1.3 CONTRIBUTIONS

- Performance evaluation of batch and messaging systems regarding throughput and latency
- Concept and prototype implementation of an adaptive middleware
- Conceptual Framework

1.4 OUTLINE OF THE THESIS

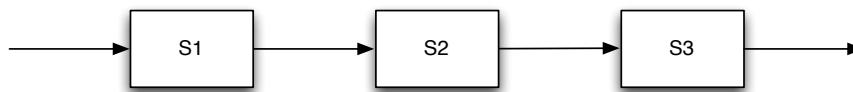
1. Introduction
2. Background
3. Related Work
4. Performance Evaluation of Batch and Message-based Systems
5. An Adaptive Middleware for Near-Time Processing of Bulk Data
6. A Conceptual Framework For Feedback-Controlled Bulk Data Processing Systems
7. Conclusion

BACKGROUND

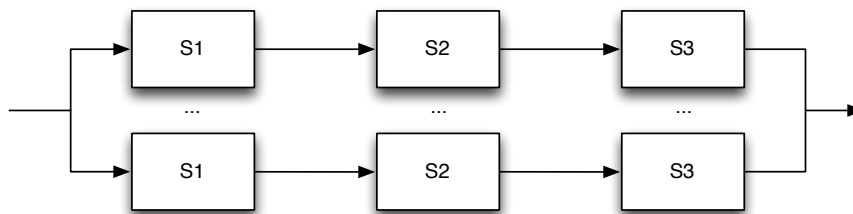
2.1 SYSTEMS FOR BULK DATA PROCESSING

We consider a distributed system for bulk data processing consisting of several subsystems running on different nodes that together form a processing chain, that is, the output of subsystem S1 is the input of the next subsystem S2 and so on (see Figure 1a).

Add reference to Pipes and Filters architectural style (EIP)



(a) Single processing line



(b) Parallel processing lines

Figure 1: A system consisting of several subsystems forming a processing chain

To facilitate parallel processing, the system can consist of several lines of subsystems with data being distributed among each line. For simplification, we consider a system with a single processing line in the remainder of this paper.

We discuss two processing types for this kind of system, batch processing and message-based processing.

2.2 BATCH PROCESSING

The traditional operation paradigm of a system for bulk data processing is batch processing (see Figure 2). A batch processing system is an application that processes bulk data without user interaction. Input and output data is usually organised in records using a file- or database-based interface. In the case of a file-based interface, the ap-

plication reads a record from the input file, processes it and writes the record to the output file.

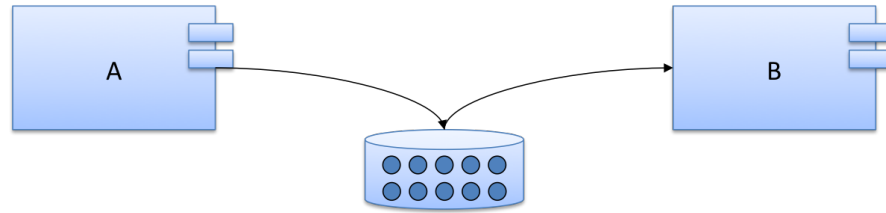


Figure 2: Batch processing

A batch processing system exhibits the following key characteristics:

- **Bulk processing of data**

A Batch processing system processes several gigabytes of data in a single run thus providing a high throughput. Multiple systems are running in parallel controlled by a job scheduler to speed up processing. The data is usually partitioned and sorted by certain criteria for optimized processing. For example, if a batch only contains data for a specific product, the system can pre-load all necessary reference data from the database to speed up the processing.

- **No user interaction**

There is no user interaction needed for the processing of data. It is impossible due to the amount of data being processed.

- **File- or database-based interfaces**

Input data is read from the file system or a database. Output data is also written to files on the file system or a database. Files are transferred to the consuming systems through FTP by specific jobs.

- **Operation within a limited timeframe**

A batch processing system often has to deliver its results in a limited timeframe due to Service Level Agreements (SLA) with consuming systems.

- **Offline handling of errors**

Erroneous records are stored to a specific persistent memory (file or database) during operation and are processed afterwards.

Applications that are usually implemented as batch processing systems are billing systems for telecommunication companies used for mediating, rating and billing of call events.

2.2.1 Integration Styles

2.2.1.1 File Transfer

2.2.1.2 Shared Database

2.2.2 Batch Architectures

2.2.3 Batch Performance Optimisations

- Distinction between different optimization types
 - Technical optimization
 - * Data formats (binary, CSV)
 - * Database transactions
 - * Database design
 - Business optimization
 - * optimization depending on data semantics
 - * Caching of reference data

2.3 MESSAGE-BASE PROCESSING

Messaging facilitates the integration of heterogeneous applications using asynchronous communication. Applications are communicating with each other by sending messages (see Figure 3). A messaging server or message-oriented middleware handles the asynchronous exchange of messages including an appropriate transaction control [Conrad et al. \(2006\)](#).

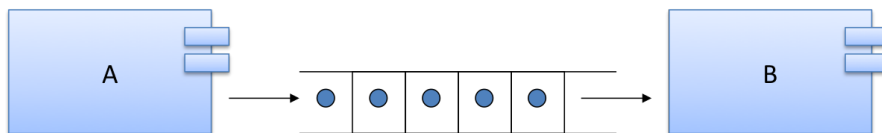


Figure 3: Message-based processing

Hohpe et al. [Hohpe and Woolf \(2003\)](#) describe the following basic messaging concepts:

- **Channels**
Messages are transmitted through a channel. A channel connects a message sender to a message receiver.
- **Messages**
A message is packet of data that is transmitted through a channel. The message sender breaks the data into messages and sends them on a channel. The message receiver in turn reads the messages from the channel and extracts the data from them.

- **Pipes and Filters**

A message may pass through several processing steps before it reaches its final destination. Multiple processing steps are chained together using a pipes and filters architecture.

- **Routing**

A message may have to go through multiple channels before it reaches its destination. A message router acts as a filter and is capable of routing a message to the next channel or to another message router.

- **Transformation**

A message can be transformed by a message translator if the message sender and receiver do not agree on the format for the same conceptual data.

- **Endpoints**

A message endpoint is a software layer that connects arbitrary applications to the messaging system.

2.3.1 *Integration Styles*

- JMS

2.3.1.1 *Point To Point*

2.3.1.2 *Publish/Subscribe*

Message-based systems are able to provide near-time processing of data due to their lower latency compared with batch processing systems. The advantage of a lower latency comes with a performance cost in regard to a lower throughput because of the additional overhead for each processed message. Every message needs amongst others to be serialised and deserialised, mapped between different protocols and routed to the appropriate receiving system.

2.4 LATENCY VS. THROUGHPUT

Needs to be thought through.

Throughput and latency are performance metrics of a system. The following definitions of throughput and latency are used in this paper:

- **Maximum Throughput**

The number of events the system is able to process in a fixed timeframe.

- **Ent-to-end Latency**

The period of time between the occurrence of an event and

its processing. End-to-end latency refers to the total latency of a complete business process implemented by multiple subsystems. The remainder of this paper focusses on end-to-end latency using the general term latency as an abbreviation.

2.4.1 Batch processing

A business process, such as billing, implemented by a system using batch processing exhibits a high end-to-end latency. For example, consider the following billing system:

- Customers are billed once per month
- Customers are partitioned in 30 billing groups
- The billing system processes 1 billing group per day, running 24h under full load.

In this case, the mean time for a call event to be billed by the billing system is 1/2 month. That is, the mean end-to-end latency of this system is 1/2 month.

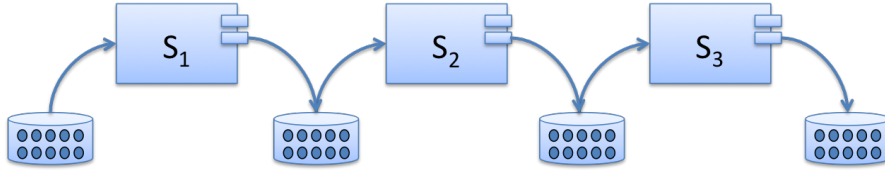


Figure 4: Batch processing system comprised of three subsystems

Assuming the system S_{Batch} which is comprised of N subsystems S_1, S_2, \dots, S_N (see Figure 4 for an example with $N = 3$):

$$S_{Batch} = \{S_1, S_2, \dots, S_N\}$$

The subsystem S_i reads its input data from the database DB_i in one chunk, processes it and writes the output to the database DB_{i+1} . When S_i has finished the processing, the next subsystem S_{i+1} reads the input data from DB_{i+1} , processes it and writes the output to DB_{i+2} , which in turn is read and processed from subsystem S_{i+3} and so on.

The latency $L_{E_{S_{Batch}}}$ of a single event processed by the system S_{Batch} is determined by the total processing time $PT_{S_{Batch}}$, which is the sum of the processing time PT_i of each subsystem S_i :

$$L_{E_{S_{Batch}}} = PT_{S_{Batch}} = \sum_{i=1}^N PT_i$$

where N is the number of subsystems.

The processing time PT_i of the subsystem S_i is the sum of the processing time of each event PT_{E_j} and the additional processing overhead OH_i , which includes the time spent for reading and writing the data, opening and closing transactions, etc:

$$PT_i = \left(\sum_{j=1}^M PT_{E_j} \right) + OH_i$$

where M is the number of events.

To allow for near-time processing, it is necessary to decrease the latency L_{E_S} of a single event. This can be achieved by using message-based processing instead of batch processing.

2.4.2 Message-based processing

The subsystem S_i of a message-based system $S_{Message}$ reads a single event from its input message queue MQ_i , processes it and writes it to the output message queue MQ_{i+1} . As soon as the event is written to the message queue MQ_{i+1} , it is read by the subsystem S_{i+1} , which processes the event and writes to the message queue MQ_{i+2} and so on (see Figure 5).

The latency $L_{E_{S_{Message}}}$ of a single event processed by the system $S_{Message}$ is determined by the total processing time $PT_{E_{S_{Message}}}$ of this event, which is the sum of the processing time PT_{E_i} and the processing overhead OH_{E_i} for the event of each subsystem:

$$L_{E_{S_{Message}}} = PT_{E_{S_{Message}}} = \sum_{i=1}^N (PT_{E_i} + OH_{E_i})$$

where N is the number of subsystems. Please note that the wait time of the event is assumed to be 0 for simplification.

The processing overhead OH_{E_i} includes amongst others the time spent for unmarshalling and marshalling, protocol mapping and opening and closing transactions, which is done for every processed event.

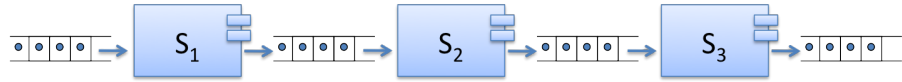


Figure 5: Message-based system comprised of three subsystems

Since the processing time $PT_{E_{S_{Message}}}$ of a single event is much shorter than the total processing time $PT_{S_{Batch}}$ of all events, the latency $L_{E_{S_{Message}}}$ of a single event using a message-based system is much smaller than the latency $L_{E_{S_{Batch}}}$ of a single event processed by a batch-processing system.

$$PT_{E_{S_{Message}}} < PT_{S_{Batch}} \Rightarrow L_{E_{S_{Message}}} < L_{E_{S_{Batch}}}$$

Message-based processing adds an overhead to each processed event in contrast to batch processing, which adds a single overhead to each processing cycle. Hence, the accumulated total processing overhead $OH_{S_{Message}}$ of a message-based system $S_{Message}$ for processing m events is larger than the total processing overhead of a batch processing system:

$$OH_{S_{Message}} = \sum_{i=1}^n OH_{E_i} * m > OH_{S_{Batch}} = \sum_{i=1}^n OH_i$$

A message-based system, while having a lower end-to-end latency, is not able to process the same amount of events in the same time as a batch processing system and therefore cannot provide the same maximum throughput.

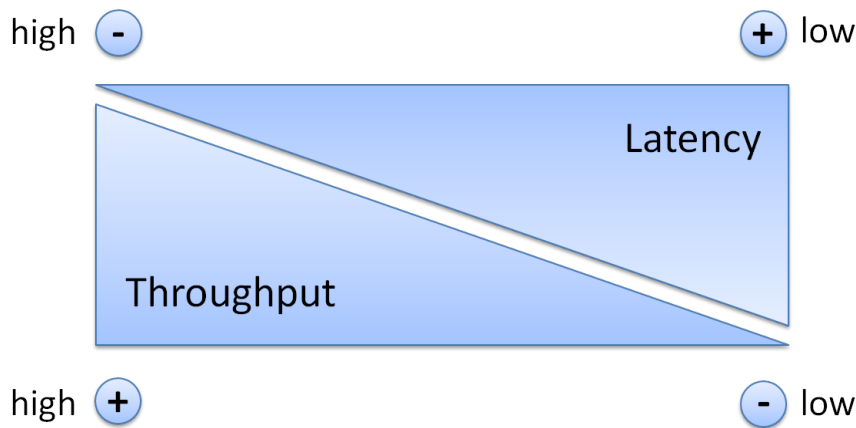


Figure 6: Latency and throughput are opposed to each other

From this follows that latency and throughput are opposed to each other (see Figure 6). High throughput, as provided by batch processing, leads to high latency, which impedes near-time processing. On the other hand, low latency, as provided by a message-based system, cannot provide the throughput needed for bulk data processing because of the additional overhead for each processed event.

2.5 SERVICE-ORIENTED ARCHITECTURE

Service Oriented Architecture ([SOA](#)) is an architectural pattern to build application landscapes from single business components. These business components are loosely coupled by providing their functionality in form of services. A service represents an abstract business view of the functionality and hides all implementation details of the component providing the service. The definition of a service acts as a contract between the service provider and the service consumer. Services are called using a unified mechanism, which provides a platform independent connection of the business components while hiding all

the technical details of the communication. The calling mechanism also includes the discovery of the appropriate service (Richter et al., 2005).

By separating the technical from the business aspects, SOA aims for a higher level of flexibility of enterprise applications.

2.6 ENTERPRISE SERVICE BUS

An Enterprise Service Bus (ESB) is an integration platform that combines messaging, web services, data transformation and intelligent routing (Schulte, 2002). Table 1 shows the main characteristics of an ESB (Chappell, 2004). All application components and integration ser-

Pervasiveness	An ESB supports multiple protocols and client technologies. It can span an entire organisation including its business partners.
Highly distributed	An ESB integrates loosely coupled application components that form a highly distributed network.
Selective deployment of integration components	The services of an ESB are independent of each other and can be separately deployed.
Security and reliability	An ESB provides reliable messaging, transactional integrity and secure authentication.
Orchestration and process flow	An ESB supports the orchestration of application components controlled by message metadata or an orchestration language like WS-BPEL.
Autonomous yet federated managed environment	Different departments can still separately manage an ESB that spans the whole organisation.
Incremental adoption	The adoption of an ESB can be incremental one project after another.
XML support	XML is the native data format of an ESB.
Real-time insight	An ESB provides real-time throughput of data by the use of its underlying message-oriented middleware and thus decreases latency.

Table 1: Main characteristics of an ESB (Chappell, 2004)

vices that are connected to the ESB are viewed as abstract service endpoints. Abstract endpoints are logical abstractions of services that are

plugged into the ESB and are all equal participants (Chappell, 2004). An abstract endpoint can represent a whole application package such as a CRM or ERP system, a small web service or an integration service of the ESB such as a monitoring, logging or transformation service. As integration platform the ESB supports various types of connections for the service endpoints. These can be SOAP, HTTP, FTP, JMS or other programming APIs for C, C++, C#, etc. It is often stated that “if you can’t bring the application to the bus, bring the bus to the application” (Chappell, 2004).

The backbone of the ESB is a message-oriented middleware (MOM), which provides an asynchronous, reliable and efficient transport of data between the service endpoints. The concrete protocol of the MOM, such as JMS, WS-Rel* or a proprietary protocol is thereby abstracted by the service endpoint. The ESB is thus a logical layer over the messaging middleware. The utilised protocol can also be varied by the ESB depending on the Quality of Service (QoS) requirements or deployment situations. Service endpoints can be orchestrated to process flows, which are mapped to concrete service invocations by the ESB.

The physical representation of a service endpoint is the service container. The service container is a remote process, which hosts the business or technical components that are connected through the bus. The set of all service containers therefore constitute the logical ESB.

A service container provides the following interfaces (Chappell, 2004):

- **Service interface**

The service interface provides an entry endpoint and exit endpoint to dispatch messages to and from the service.

- **Management interface**

The management interface provides an entry endpoint for retrieving configuration data and an exit endpoint for sending logging, event tracking and performance data.

2.7 ENTERPRISE INTEGRATION PATTERNS

EIP

2.7.1 Performance relevant EIPs

- Aggregator
- Content Filter
- Message Router
- Claim Check

2.8 PERFORMANCE ISSUES

This section describes the performance issues of an SOA middleware that inhibit their appropriateness for systems with high performance requirements.

2.8.1 *Distributed Architecture*

A system implemented according to the principles of SOA is a distributed system. Services are hosted on different locations belonging to different departments and even organizations. Hence, the performance drawbacks of a distributed system generally also apply to SOA. This includes the marshalling of the data that needs to be sent to the service provider by the service consumer, sending the data over the network and the unmarshalling of data by the service provider.

2.8.2 *Integration of Heterogeneous Technologies*

A main goal of introducing an SOA is to integrate applications implemented with heterogeneous technologies. This is achieved by using specific middleware and intermediate protocols for the communication. These protocols are typically based on XML, like SOAP (*SOAP Specification*, 2007). XML, as a very verbose language, adds a lot of meta-data to the actual payload of a message. The resulting request is about 10 to 20 times larger than the equivalent binary representation (O'Brien et al., 2007), which leads to a significant higher transmission time of the message. Processing these messages is also time-consuming, as they need to get parsed by a XML parser before the actual processing can occur.

The usage of a middleware like an Enterprise Service Bus (ESB) adds further performance costs. An ESB usually processes the messages during transferring. Among other things, this includes the mapping between different protocols used by service providers and service consumers, checking the correctness of the request format, adding message-level security and routing the request to the appropriate service provider (See, for example, Josuttis (2007) or Krafzig et al. (2005)).

2.8.3 *Loose Coupling*

Another aspect of SOA that has an impact on performance is the utilisation of loose coupling. The aim of loose coupling is to increase the flexibility and maintainability of the application landscape by reducing the dependency of its components on each other. This denotes that service consumers shouldn't make any assumptions about the implementation of the services they use and vice versa. Services be-

come interchangeable as long they implement the interface the client expects.

Engels et al. (2008) consider two components A and B loosely coupled when the following constraints are satisfied:

- **Knowlegde**
Component A knows only as much as it is needed to use the operations offered by component B in a proper way. This includes the syntax and semantic of the interfaces and the structure of the transferred data.
- **Dependence on availability**
Component A provides the implemented service even when component B is not available or the connection to component B is not available.
- **Trust**
Component B does not rely on component A to comply with pre-conditions. Component A does not rely on component B to comply with post-conditions.

Coupling between services occurs on different levels. Krafzig et al. (2005) describe the different levels of coupling that are leveraged in an SOA (see Table 2).

The gains in flexibility and maintainability of loose coupling are amongst others opposed by performance costs.

Service consumers and service provider are not bound to each other statically. Thus, the service consumer needs to determine the correct end point of the service provider during runtime. This can be done by looking up the correct service provider in a service repository either by the service consumer itself before making the call or by routing the message inside the ESB.

Apart from very few basic data types, Service consumers and service providers do not share the same data model. It is therefore necessary to map data between the data model used by the service consumer and the data model used by the service provider.

2.9 CURRENT APPROACHES FOR IMPROVING THE PERFORMANCE OF AN SOA MIDDLEWARE

This section describes current approaches to the performance issues introduced in the previous section.

2.9.1 *Hardware*

The obvious solution to improve the processing time of a service is the utilization of faster hardware and more bandwidth. SOA performance issues are often neglected by suggesting that faster hardware

Table 2: Levels of coupling

Level	Tight Coupling	Loose Coupling
Physical coupling	Direct physical link required	Physical intermediary
Communication style	Synchronous	Asynchronous
Type system	Strong type system	Weak type system
Interaction pattern	OO-style navigation of complex object trees	Data-centric, self-contained messages
Control of process logic	Central control of processing logic	Distributed logical components
Service discovery and binding	Statically bound services	Dynamically bound services
Platform dependencies	Strong OS and programming language dependencies	OS and programming languages independent

or more bandwidth will solve this problem. However, it is often not feasible to add faster or more hardware due to high cost pressure.

2.9.2 *Compression*

The usage of XML as an intermediate protocol for service calls has a negative impact on their transmission times over the network. The transmission time of service calls and responses can be decreased by compression. Simply compressing service calls and responses with gzip can do this. The World Wide Web Consortium (W3C) proposes a binary presentation of XML documents called binary XML (*EXI Working Group, 2007*) to achieve a more efficient transportation of XML over networks.

It must be pointed out that the utilisation of compression adds the additional costs of compressing and decompressing to the overall processing time of the service call.

2.9.3 *Service Granularity*

To reduce the communication overhead or the processing time of a service, the service granularity should be reconsidered.

Coarse-grained services reduce the communication overhead by achieving more with a single service call and should be the favoured service design principle (*Hess et al., 2006*). However, the processing time of a coarse grained service can pose a problem to a service consumer that only needs a fracture of the data provided by the service. To reduce the processing time it could be considered in this case to add a finer grained service that provides only the needed data (*Josuttis, 2007*).

It should be noted that merging multiple services to form a more coarse grained service or splitting a coarse grained service into multiple services to solve performance problems specific to a single service consumer reduces the reusability of the services for other service consumers (*Josuttis, 2007*).

2.9.4 *Degree of Loose Coupling*

The improvements in flexibility and maintainability gained by loose coupling are opposed by drawbacks on performance. Thus, it is crucial to find the appropriate degree of loose coupling.

Hess et al. (2006) introduce the concept of distance to determine an appropriate degree of coupling between components. The distance of components is comprised of the functional and technical distance. Components are functional distant if they share few functional similarities. Components are technical distant if they are of a different category. Categories classify different types of components like in-

ventory components, process components, function components and interaction components.

Distant components trust each other in regard to the compliance of services levels to a lesser extent than near components do. The same applies to their common knowledge. Distant components share a lesser extent of knowledge of each other. Therefore, [Hess et al. \(2006\)](#) argue that distant components should be coupled more loosely than close components.

The degree of loose coupling between components that have been identified to be performance bottlenecks should be reconsidered to find the appropriate trade-off between flexibility and performance. It can be acceptable in that case to decrease the flexibility in favour of a better performance.

2.10 SUMMARY

Message-oriented middleware facilitates the integration of applications using asynchronous messages. An Enterprise Service Bus is such a middleware combining messaging, web services, data transformation and intelligent routing. Message-based systems are able to provide near-time processing of data due to their lower latency compared with batch processing systems. The advantage of a lower latency comes with a performance cost in regard to a lower throughput because of the additional overhead for each processed message. Every message needs amongst others to be serialised and deserialised, mapped between different protocols and routed to the appropriate receiving system.

Current approaches to improve the throughput performance of message-based systems try to reduce the transmission time by compressing messages. Another approach is to adjust the service granularity to form more coarse-grained services or to adjust the degree of loose coupling to reduce the communication overhead.

While these approaches generally improve the performance of message-based systems, they are still not able provide the same throughput as that can be achieved with a batch processing system. Additionally, the current approaches are static and thus need to be considered at the design-time of the system. The next chapter presents an SOA middleware for high performance near-time processing of bulk data which is a novel approach to dynamically reduce the latency of a system while still providing high throughput.

RELATED WORK

This chapter gives an overview of work related to this PhD project (see figure 7). It starts with work that addresses the performance of Service-Oriented systems in general. Further work in the area of SOA performance can be classified into the categories performance modeling, performance measuring and performance optimisation.

The proposed middleware for high-performance near-time processing of bulk data adjusts the data granularity itself at runtime. Work on middleware discusses different approaches for self-adjustment and self-awareness of middleware, which can be classified as adaptive or reflective middleware, discussed in the next section.

In order to dynamically adjust the data granularity at runtime, the proposed middleware needs to constantly measure the throughput and latency of the system. Work on SLA-monitoring proposes different approaches to monitor the compliance of business processes to Service Level Agreements.

Finally, the chapter concludes with a summary which relates the discussed approaches to the approach proposed in this PhD project.



Figure 7: Related Work

3.1 PERFORMANCE OF SERVICE-ORIENTED SYSTEMS

O'Brien et al. (2007) argue that the introduction of an SOA generally has a negative impact on the performance of the system. They identify the following key aspects responsible for the performance degradation:

- **Network communication**
Service provider and service consumer need to communicate over a network, which usually does not offer a deterministic latency.
- **Lookup of services in a directory**
The lookup of a service provider in a directory increases the total transaction time of a service request.
- **Interoperability of services on different platforms**
The interoperability of services on different platforms is real-

ized by a middleware which handles the whole communication. The needed marshalling and unmarshalling of data adds a performance overhead to the communication.

- **Usage of standard messaging formats**

The usage of a standard message format, like XML, increases the processing time of a service due to parsing, validation and transformation of messages. An XML message can be 10 to 20 times larger than the binary representation which increases the transport time of the message over the network.

In another paper, [O'Brien et al. \(2008\)](#) state that the performance issues of an SOA are caused by:

- Overhead of XML
- Implementation of composite services
- Service orchestration
- Service invocation
- Resources, e.g. threads, CPUs
- Resource models, e.g. virtualization

The authors suggest that it is vital to consider performance aspects early in the development lifecycle, which can be supported by using an SOA performance model.

[Woodall et al. \(2007\)](#) describe in their paper the challenges they encountered when analysing a performance problem of a concrete Service-Oriented System:

- Physical distribution of services
- Continual use of services by local users or developers during the performance investigation
- Heterogeneity of the underlying service software platform

3.2 PERFORMANCE MEASURING

Performance measuring is applied to evaluate if an implemented system meets its performance requirements and to spot possible performance problems.

[Her et al. \(2007\)](#) propose the following set of metrics for measuring the performance of a service-oriented system:

- **Service response time**

Elapsed time between the end of request to service and the beginning of the response of the service. This metric is further split in 20 sub-metrics such as message processing time, service composition time and service discovery time.

- **Think time**

Elapsed time between the end of a response generated by a service and the beginning of a response of an end user.

- **Service turnaround time**

Time needed to get the result from a group of related activities within a transaction.

- **Throughput**

Number of requests served at a given period of time. The authors distinguish between the throughput of a service and the throughput of a business process.

In their work, [Henjes et al.](#) investigated the throughput performance of the JMS server FioranaMQ, SunMQ and WebsphereMQ. The authors came to the following conclusion ([Henjes et al. \(2006\)](#) and [Menth et al. \(2006a\)](#)):

- Message persistence reduces the throughput significantly.
- Message replication increases the overall throughput of the server.
- Throughput is limited either by the processing logic for small messages or by the transmission capacity for large messages.
- Filtering reduces the throughput significantly.

[Chen and Greenfield \(2004\)](#) propose that the following performance metrics should be used to evaluate a JMS server:

- Maximum sustainable throughput
- Latency
- Elapsed time taken to send batches messages
- Persistent message loss after recovery

The authors state that “although messaging latency is easy to understand, it is difficult to measure precisely in a distributed environment without synchronised high-precision clocks.” They discovered that latencies increase with increasing message sizes.

SPECjms2007 is a standard benchmark for the evaluation of Message-Oriented Middleware platforms using JMS ([Sachs et al., 2009](#)). It provides a flexible performance analysis framework for tailoring the workload to specific user requirements. According to [Sachs et al. \(2007\)](#), the workload of the SPECjms2007 benchmark has to meet the following requirements:

- **Representativeness**

The workload should reflect how the messaging platform is used in typical user scenarios.

- **Comprehensiveness**

The workload should incorporate all platform features typically used in JMS application including publish/subscript and point-to-point messaging.

- **Focus**

The workload should focus on measuring the performance of the messaging middleware and should minimize the impact of other components and services.

- **Configurability**

It should be possible to configure the workload to meet the requirements of the user.

- **Scalability**

It should be possible to scale the workload by the number of destinations with a fixed traffic per destination or by increasing the traffic with a fixed set of destinations.

3.3 PERFORMANCE OPTIMISATION

Most of the work that aims to optimise the performance of service-oriented systems is done in the area of Web Services since it is a common technology to implement a SOA.

In particular, various approaches have been proposed to optimise the performance of SOAP, the standard protocol for Web Service communication. This includes approaches for optimising the processing of SOAP messages (see for example [Abu-Ghazaleh and Lewis \(2005\)](#), [Suzumura et al. \(2005\)](#) and [Ng \(2006\)](#)), compression of SOAP messages (see for example [Estrella et al. \(2008\)](#) and [Ng et al. \(2005\)](#)) and caching (see for example [Andresen et al. \(2004\)](#) and [Devaram and Andresen \(2003\)](#)). A survey of the current approaches to improve the performance of SOAP can be found in [Tekli et al. \(2012\)](#).

[Wichaiwong and Jaruskulchai \(2007\)](#) propose an approach to transfer bulk data between web services per FTP. The SOAP messages transferred between the web services would only contain the necessary details how to download the corresponding data from an FTP server since this protocol is optimized for transferring huge files. This approach solves the technical aspect of efficiently transferring the input and output data but does not pose any solutions how to implement loose coupling and how to integrate heterogeneous technologies, the fundamental means of an SOA to improve the flexibility of an application landscape.

Data-Grey-Box Web Services are an approach to transfer bulk data between Web Services ([Habich, Richly and Grasselt, 2007](#)). Instead of transferring the data wrapped in SOAP messages, it is transferred using an external data layer. For example when using database systems

as data layer, this facilitates the use of special data transfer methods such as ETL (Extract, Transform, Load) to transport the data between the database of the service requestor and the database of the Web service. The data transfer is transparent for both service participants in this case. The approach includes an extension of the Web service interface with properties describing the data aspects. Compared to the SOAP approach, the authors measured a speedup of up to 16 using their proposed approach. To allow the composition and execution of Data-Grey-Box Web services, [Habich, Richly, Preissler, Grasselet, Lehner and Maier \(2007\)](#) developed BPEL data transitions to explicitly specify data flows in BPEL processes.

[Zhuang and Chen \(2012\)](#) propose three tuning strategies to improve the performance of Java Messaging Service (JMS) for cloud-based applications.

1. When using persistent mode for reliable messaging the storage block size should be matched with the message size to maximise message throughput.
2. Applying distributed persistent stores by configuring multiple JMS destinations to achieve parallel processing
3. Choosing appropriate storage profiles such as RAID-1

MPAB (Massively Parallel Application Bus) is an ESB-oriented messaging bus used for the integration of business applications ([Benosman et al., 2012](#)). The main principle of MPAB is to fragment an application into parallel software processing units, called SPU. Every SPU is connected to an Application Bus Multiplexor (ABM) through an interface called Application Bus Terminal (ABT). The Application Bus Multiplexor manages the resources shared across the host system and communicates with other ABM using TCP/IP. The Application Bus Terminal contains all the resources needed by SPU to communicate with its ABM. A performance evaluation of MPAB shows that it achieves a lower response time compared to the open source ESBs Fuse, Mule and Petals.

Some research has been done to add real-time capabilities to ESB or messaging middleware. [Garces-Erice \(2009\)](#) proposes an architecture for a real-time messaging middleware based on an Enterprise Service Bus. It consists of an event scheduler, a JMS-like API and a communication subsystem. While fulfilling real-time requirements, the middleware also supports already deployed infrastructure.

In their paper, [Xia and Song \(2011\)](#) suggest a real-time ESB model by extending the JBI specification with semantics for priority and time restrictions and modules for flow control and bandwidth allocation. The proposed system is able to dynamically allocate bandwidth according to business requirements.

Tempo is a real-time messaging system written in Java that can be used on either a real-time or non-real-time architecture ([Bauer et al.,](#)

2008). The authors, Bauer et al., state that existing messaging systems are designed for transactional processing and therefore not appropriate for applications with stringent requirements of low latency with high throughput. The main principle of Tempo is to use an independent queuing system for each topic. Resources are partitioned between these queueing systems by a messaging scheduler using a time-base credit scheduling mechanism. In a test environment, Tempo is able to process more than 100.000 messages per second with a maximum latency of less than 120 milliseconds.

Haesen et al. (2008) distinguishes between two types of data granularity:

- **Input data granularity**
Data that is sent to a component
- **Output data granularity**
Data that is returned by a component

The authors state that a coarse-grained data granularity reduces the communication overhead, since the number of network transfers is decreased. “Especially in the case of Web services, this overhead is high since asynchronous messaging requires multiple queuing operations and numerous XML transformations”.

3.4 SELF-ADAPTIVE SOFTWARE SYSTEMS

- Definition Self-Adaptive Software
- Reference Architectures for self-adaptive software systems
 - Three Layer Architecture Model for Self-Management (Kramer and Magee, 2007)
 - Conceptual Architecture for Self-Adaptive Software Systems (Andersson et al., 2009)
 - Mape-K (IBM Group, 2005)
- Self-adaptive properties
 - Automatic Workarounds
- Self-adaptive middleware
 - Self-adaptive SOA
 - Self-adaptive ESB
- Self-adaptive frameworks
 - Rainbow framework
 - PLASTIC
 - Javeleon

- JavAdaptor
- ArchJava

3.4.1 Definition

Self-Adaptive Software is a “a closed-loop system with a feedback loop aiming to adjust itself to changes during its operation” (Salehie and Tahvildari, 2009). These changes can originate from internal causes of the system (the system’s self) or from the context of the system.

Laddaga and Robertson (2008) provides a definition for self-adaptive software: “Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.”

Another definition is given by Oreizy et al. (1999): “Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.”

Salehie and Tahvildari (2009) describe the following properties (also called self-* properties) of a self-adaptive system:

- **Self-configuring**
The system is able to reconfigure itself in response to changes.
- **Self-healing**
The system is able to discover, diagnose and react on failures.
- **Self-optimizing**
The system is able to manage performance and resource allocation to meet different performance requirements.
- **Self-protecting**
The system is able to detect security breaches and to recover from them.

More general self-* properties are described as:

- **Self-Awareness**
The system is aware of its self states and behaviours.
- **Context-Awareness**
The system is aware of its context.

3.5 SELF-ADAPTIVE MIDDLEWARE

Duran-Limon et al. (2004) argue that “the most adequate level and natural locus for applying adaption is at the middleware level”. Adaption at the operating system level is platform-dependent and changes

at this level affect every application running on the same node. On the other hand, adaption at application level assigns the responsibility to the developer and is also not reusable.

Lee et al. (2009) propose an adaptive, general-purpose runtime infrastructure for effective resource management of the infrastructure. Their approach is comprised of three components:

1. dynamic performance prediction
2. adaptive intra-site performance management
3. adaptive inter-site resource management

The runtime infrastructure is able to choose from a set of performance predictions for a given service and to dynamically choose the most appropriate prediction over time by using the prediction history of the service.

AutoGlobe (Gmach et al., 2008) provides a platform for adaptive resource management comprised of

1. Static resource management
2. Dynamic resource management
3. Adaptive control of Service Level Agreements (SLA)

Static resource management optimises the allocation of services to computing resources and is based on on automatically detected service utilisation patterns. Dynamic resource management uses a fuzzy controller to handle exceptional situations at runtime. The Adaptive control of Service Level Agreements schedules service requests depending on their SLA agreement.

The coBRA framework proposed by Irmert et al. (2008) is an approach to replace service implementations at runtime as a foundation for self-adaptive applications. The framework facilitates the replacement of software components to switch the implementation of a service with the interface of the service staying the same.

DREAM (Dynamic Reflective Asynchronous Middleware) (Leclercq et al., 2004) is a component-based framework for the construction of reflective Message-Oriented Middleware. Reflective middleware “refers to the use of a causally connected self-presentation to support the inspection and adaption of the middleware system” (Kon et al., 2002). DREAM is based on FRACTAL, a generic component framework and supports various asynchronous communication paradigms such as message passing, event-reaction and publish/subscribe. DREAM facilitates the construction and configuration of Message-Oriented Middleware from a library of components such as message queues, filters, routers and aggregators, which can be assembled either at deploy-time or runtime.

3.6 SLA-MONITORING OF BUSINESS PROCESSES

The SECMOL framework (Service Centric Monitoring Language), developed by Guinea et al. (2009), allows to monitor the quality of service constraints of BPEL processes. It is comprised of three components. Data Collectors for capturing data, Data Analyzers for analysing the captured data and the Monitoring Manager for coordinating the monitoring process. SECMOL also defines a XML-based monitoring specification, which consists of monitoring policies that specify how the monitoring should be done and monitoring rules that express the quality of service properties the system needs to satisfy.

Duc et al. (2009) argue that a monitoring middleware component should fulfill the following requirements:

- **Coherency of data**
All data used in one decision must reflect the same state of the system.
- **Flexibility in data access**
Every monitored service provider should be able to respond using its own measurement units. This should be transparent for the client using the monitoring data.
- **Performance in data access**
The monitoring should have the slightest possible impact on the performance of the business process.
- **Network usage optimisation**
The transmission of monitoring data should have the slightest possible impact on the network performance.

The authors propose M4ABP (Monitoring for Adaptive Business Process), a distributed monitoring and data delivery middleware subsystem, which implements these requirements.

SALMon (Ameller and Franch, 2008) is a system for monitoring the services of an SOA for Service Level Agreement violations. It is itself implemented as an service-oriented system and consists of the following services:

- **Monitor**
The Monitor service collects the monitoring data from components called Measure Instruments that are instantiated in each monitored service.
- **Analyzer**
The Analyzer service manages the Monitor service and checks for Service Level Agreement violations of the monitored services.

- **Decision Maker**

The Decision Maker service is able to select an action to solve the SLA violation. The appropriate action for a specific SLA violation is stored in a repository.

The attributes measured by SALMon are taken from an ISO/IEC 9126-1-based quality model.

[Textor et al. \(2009\)](#) propose an approach to map implementation level monitoring data to business level activities. Non-functional constraints are specified on a workflow model in the modelling phase. Additionally, an instrumentation model is used to specify the instrumentation points of the application. At runtime, the monitoring data of the system is mapped to the workflow model. The monitoring data is received by a component called ConstraintMonitor, which evaluates and validates the constraints specified in the workflow model.

[Wetzstein et al. \(2009\)](#) present a framework to monitor and analyse the factors that influence the performance of WS-BPEL processes. The authors distinguish between PPM (Process Performance Metrics) and QoS (Quality of Service) metrics, which influence the Key Performance Indicators (KPI) of business processes. PPMs are based on process runtime events, that are published by the WS-BPEL runtime engine, for example the “number of orders which can be served as inhouse stock”. QoS metrics are technical parameters of the underlying services that implement the business process, for example the response time and availability of a service. KPIs are based on business goals, for example “order fulfillment lead time < 3”. The proposed framework monitors KPIs, PPMs and QoS metrics at runtime, which are modeled in a Process Metrics Definition Model (PMDM). These collected metrics can then be used to perform a dependency analysis of the influential factors of a KPI using machine learning techniques to construct dependency trees.

iBOM ([Castellanos et al., 2005](#)) is a platform to analyse, manage and optimise business operations based on business goals. Optimisations are performed by using simulation techniques. iBom simulates different configurations of a business process to identify the configuration that best meets the business goals. First, the user needs to define the optimisation metric and constraints on this metric and on the resources. The configuration candidates are then either computed by iBOM using different resource allocations of the given configuration within the defined constraints or are provided by the user in the form of a process model.

3.7 SOFTWARE PERFORMANCE ENGINEERING

- Software Performance Engineering ([SPE](#)) is a “method for constructing software systems to meet performance objectives” ([Smith, 1990](#)).

- The process begins early in the software lifecycle
- Uses quantitative methods to indentify designs that meet the performance requirements and to dismiss those that are likely to miss them.
- SPE is continued during detailed design, implementation test and operation to predict and manage the performance of a software system and to monitor and report the actual performance fo the system.
- SPE methods include performance data collection, quantitative analysis techniques, prediction strategies, management of uncertainties, data presentation and tracking, model verification and validation, critical success factors, and performance design principles.
- (Woodside et al., 2007) differentiates between two general approaches
 - measurement-based
 - * testing, diagnosis and tuning late in the development cycle, when the system is already implemented and can be run and tested.
 - model-based
 - * creates performance models and uses quantitative results from these models early in the development cycle to predict the performance of the system and to adjust the architecture of the system to meet its performance requirements.
- Best Practices for Software Performance Engineering (Smith and Williams, 2003)

3.8 SUMMARY

- Performance optimization is done at the transport layer (XML, Messaging)

Most of the work done in the field of performance of service-oriented systems involves performance aspects of Web Services including the SOAP standard. This includes performance modeling, performance measuring and performance optimisation.

Approaches to optimise the transfer of bulk data of Web services, as proposed by Wichaiwong and Jaruskulchai (2007) and Habich, Richly and Grasselt (2007) deliver an overall better performance than using SOAP. However, like a traditional batch-processing system using file-

or database-based integration, they are not able to reduce the latency and thus cannot deliver near-time processing of bulk data.

Current self-adapting middleware platforms, like the AutoGlobe platform (Gmach et al., 2008), are focused on adaptive resource management to dynamically allocate services to computing nodes or to replace service implementations at runtime, as proposed by the co-BRA framework (Irmert et al., 2008).

Work on SLA-monitoring of business processes proposes different approaches to monitor the compliance of a business process to Service Level Agreements, which include the end-to-end latency and throughput of the business process. However, they do not propose any solutions for improving the end-to-end latency in order to provide near-time processing of bulk data.

The research project presented in this report proposes an adaptive middleware to reduce the latency of a system for bulk data processing by dynamically adjusting the data granularity at runtime based on the current throughput and the minimum acceptable throughput of the system. To the best of our knowledge, this is a novel approach which has not yet been discussed in current literature.

Part II

CONTRIBUTIONS

PERFORMANCE EVALUATION OF BATCH AND MESSAGE-BASED SYSTEMS

4.1 INTRODUCTION

Traditionally, business information systems for bulk data processing are implemented as batch processing systems. Batch processing delivers high throughput but cannot provide near-time processing of data, that is the end-to-end latency of such a system is high.

A lower end-to-end latency can be achieved by using message-based processing, for example by utilising a message-oriented middleware for the integration of the services that form the business information system. While this approach is able to deliver near-time processing, it is hardly capable for bulk data processing due to the additional communication overhead for each processed message. Therefore, message-based processing is usually not considered for building a system for bulk data processing requiring high throughput.

This chapter compares the performance of a batch and message-based system. The main objectives of this comparison are:

- What is the impact of different processing styles, that is batch and message-based processing, on throughput and latency?
- What is the impact of data granularity on latency and throughput when using a message-based processing style?

To find solutions for these questions, the following approach has been taken:

- Two prototypes of a billing system for each processing type (see Section 4.2) have been built.
- A performance evaluation has been conducted to compare the prototypes with each other with the focus on throughput and latency (see Section 4.3).
- To evaluate the impact of different aggregation sizes on throughput and latency, the messaging prototype has been extended with an aggregator. A performance test has been conducted with different static aggregation sizes (see Section 4.4).

This chapter is organised as follows. Section 4.2 introduces the batch and message-based prototype systems that have been implemented. To compare the performance characteristics of the two process-

ing types, batch processing and message-based processing, a performance evaluation has been conducted, which is presented in Section 4.3. Section 4.4 shows the impact of data granularity on throughput and latency of the messaging prototype. Section 4.5 gives an overview of other work related to the contents of this chapter. Finally, this chapter concludes with a summary in Section 4.6

4.2 A REAL WORLD EXAMPLE APPLICATION

This section introduces the two prototypes of a billing system that have been built to evaluate the performance of batch and message-based processing.

A billing system is a distributed system consisting of several sub components that process the different billing sub processes like mediation, rating, billing and presentment (see Figure 8).



Figure 8: Billing process

The mediation components receive usage events from delivery systems, like switches and transform them into a format the billing system is able to process. For example, transforming the event records to the internal record format of the rating and billing engine or adding internal keys that are later needed in the process. The rating engine assigns the events to the specific customer account, called guiding, and determines the price of the event, depending on the applicable tariff. It also splits events if more than one tariff is applicable or the customer qualifies for a discount. The billing engine calculates the total amount of the bill by adding the rated events, recurring and one-time charges and discounts. The output is processed by the presentment components, which format the bill, print it, or present it to the customer in self-service systems, for example on a website.

In order to compare batch and message-based types of processing, two different prototypes of a billing application have been developed. Each prototype implements the mediation and rating steps of the billing process. Figure 9 shows the components of the billing prototype:

- **Event Generator**

The *Event Generator* generates the calling events, i.e. the Call Detail Records (CDR) that are processed by the billing application.

- **Mediation**

The *Mediation* component checks whether the calltime of the calldetail record exceeds the minimal billable length or if it belongs to

a flatrate account and sets the corresponding flags of the record. The output of the *Mediation* component are Normalized Call Detail Records (NCDR) that are further processed by the *Rating* component.

- **Rating**

The *Rating* component processes the output from the *Mediation* component. It assigns the calldetail record to a customer account and determines the price of the call event by looking up the correspondant product and tariff in the *Master Data DB*. The output of the *Rating* component (costed events) is afterwards written to the *Costed Events DB*.

- **Master Data DB**

The *Master Data DB* contains products, tariffs and accounts used by the *Event Generator* and the *Rating* component.

- **Costed Events DB**

The *Costed Events DB* contains the result of the *Rating* component, i.e. the costed events.

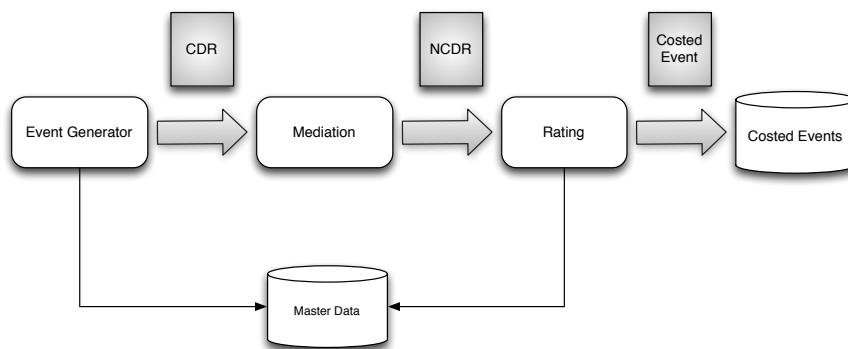


Figure 9: Components of the billing application prototype

The prototypes are implemented with Java 1.6 using Java Persistence API (JPA) for the data-access layer and a MySQL database. See Table 3 for complete list of technologies and frameworks used for the implementation of the prototypes.

4.2.1 Common Architecture

The objective of this performance evaluation is to compare the different processing styles, batch and single-event processing, with each other. It needs to be ensured that the comparison only includes the different processing styles. Therefore, the prototypes should only differ in their processing style, all other aspects should be the same, for example the business functionality, data access and datamodel.

Table 3: Technologies and frameworks used for the implementation of the prototypes

Language	Java 1.6
Dependancy Injection	Spring
Persistence API	OpenJPA (JPA 2.0)
Database	MySQL
Logging	Logback
Test	JUnit
Batch Framework	Spring Batch
Messaging Middleware	Apache Camel
Other Frameworks	Joda-Time, Apache Commons

To ensure the comparability between the prototypes, a common architecture used by both prototypes has been designed and implemented.

It consists of the following components (see Figure 10):

- **Integration Layer**
Implements the integration style, i.e. file-based integration and message-based integration.
- **Business Service**
Implements the business functionality, i.e. mediation and rating.
- **Data Access Layer**
Implements the data access.

4.2.1.1 Business Services

The business functionality, mediation and rating, is implemented by business services, which are used by both prototypes (see Figure 11):

- **MediationProcessor**
Implements the mediation functionality.
- **RatingProcessor**
Implements the rating functionality.

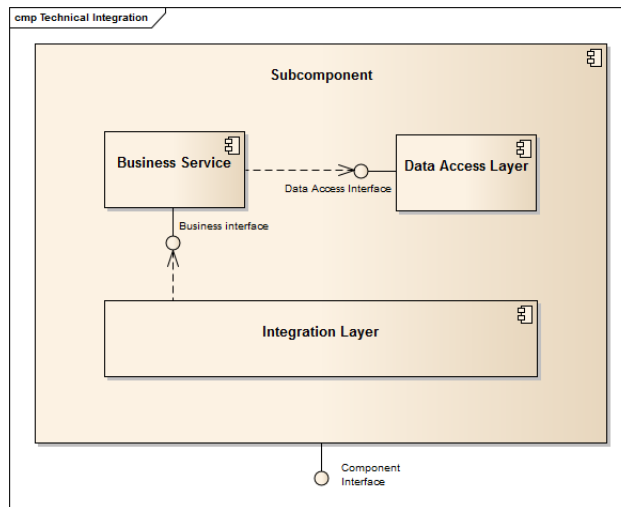


Figure 10: The prototypes share the same business components, database and data-access layer.

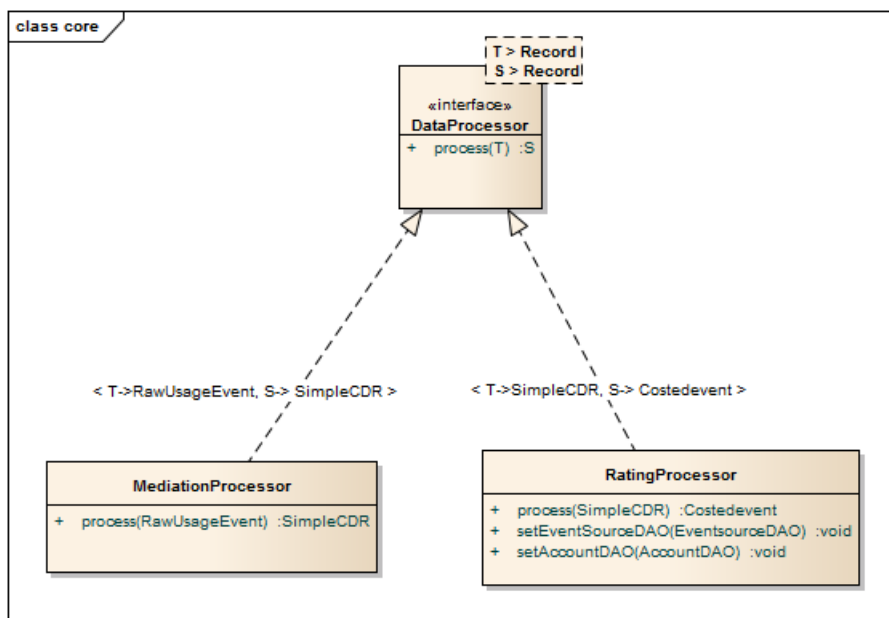


Figure 11: Business services

4.2.1.2 Integration Layer

The integration layer implements the different integration styles of the two prototypes. The batch prototype uses a batch layer which provides components for file-based data integration, transaction and control of batch processes.

The messaging prototype uses a messaging middleware for exchanging messages (see Figure 12b). The messaging middleware provides components for the transport, transformation and routing of messages.

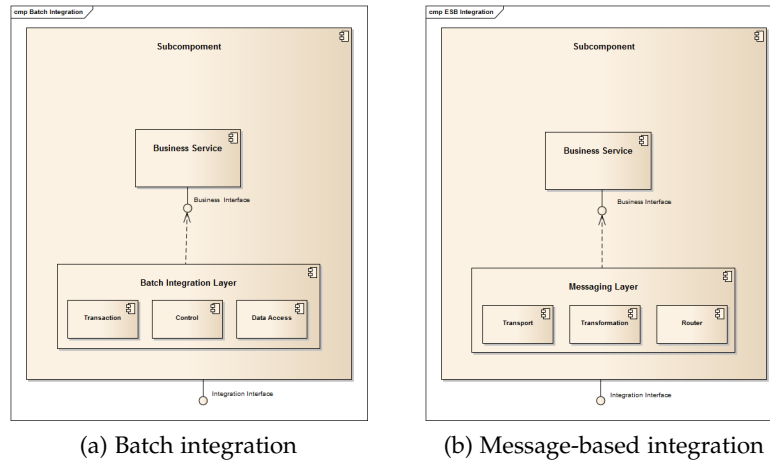


Figure 12: The prototypes use different integration layers.

4.2.1.3 Data model

The prototypes use a common data model as shown in Figure 13. It consists of the following entities:

- **Customer**
Represents a customer. A customer has an account and one or many products.
- **Account**
Contains payment informations of a customer.
- **Product**
A product such as a voice or data plan.
- **Tariff**
The tariff of a product. Defines the price of a product.
- **EventSource**
Mobile number or IP associated with a product instance of a customer.

- **CostedEvent**
An event that has been rated by the rating component.
- **SkippedEvent**
An event that has been skipped by the mediation component. For example a flat rate event.
- **CustomerProduct**
Contains the booked products of a customer. A customer can have zero or many products.
- **CustomerProductTariff**
Contains the tariffs of a product. A product can have one or many tariffs.

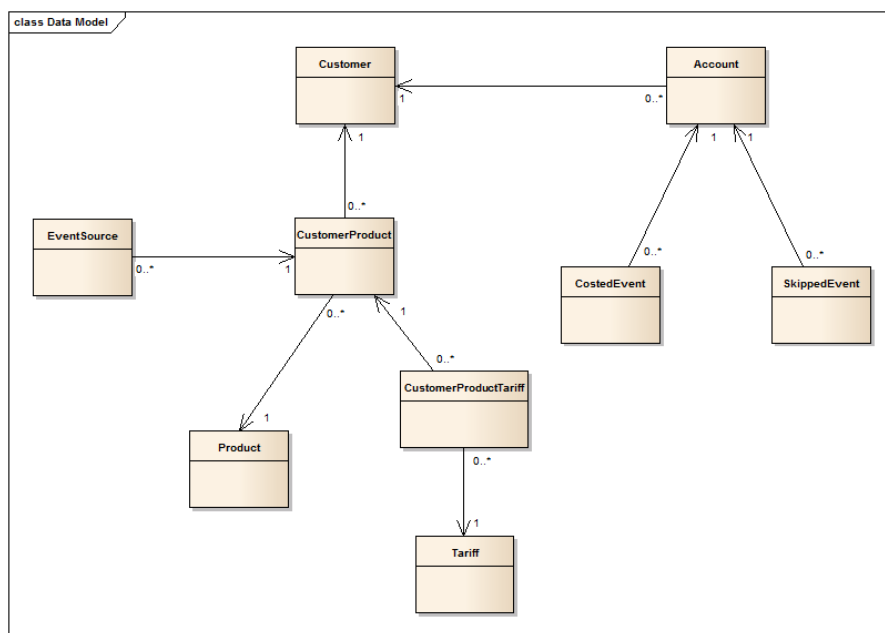


Figure 13: Logical data model of the prototype

4.2.1.4 Data Access Layer

The data access layer provides common access to the database by using the Object-relational mapping (ORM) framework OpenJPA. All business domain entities have been generated from the data model using the toolchain provided by OpenJPA. The data access for retrieving, creating and update of the domain entities is implemented using the DAO pattern (Alur et al., 2003).

4.2.2 Batch prototype

The batch prototype implements the billing application utilizing the batch processing type. It uses the Spring Batch framework (Spring

Batch, 2013), a Java framework that facilitates the implementation of batch applications by providing basic building blocks for reading, writing and processing data.

Figure 14 shows the architecture of the batch prototype. It consists of two nodes, mediation batch and rating batch, each implemented as a separate spring batch application. The nodes are integrated using Apache Camel (*Apache Camel*, 2014), an Java integration framework based on enterprise integration patterns, as described by Hohpe and Woolf (2003). Apache Camel is responsible for listening on the file system, calling the Spring batch application when a file arrives and transferring the output from the mediation batch node to the rating batch node using File Transfer Protocol (FTP).

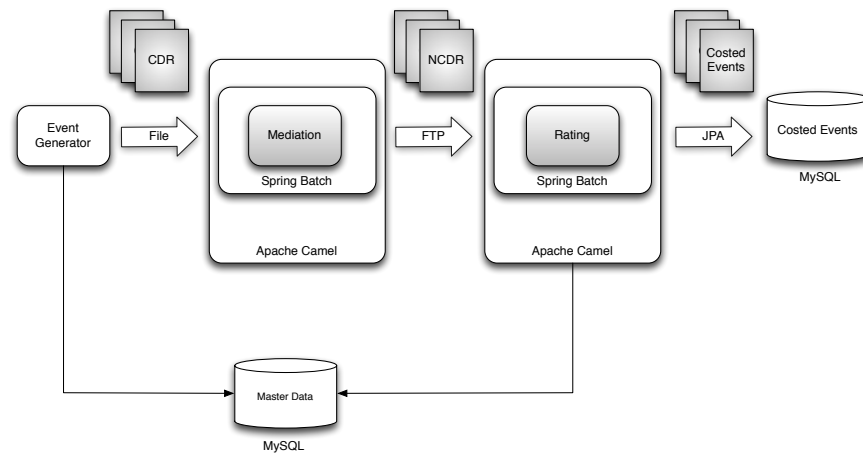


Figure 14: Batch prototype

The batch prototype performs the following steps:

1. The *Event generator* generates call detail records and writes them to a single file.
2. The *Mediation component* opens the file, processes it and writes the output to a single output file. The output file is getting transferred using [FTP](#) to the *Rating component*.
3. The *Rating component* opens the file, processes it and writes the costed events to the costed event database.

4.2.2.1 Implementation details

The main entities in Spring Batch are Jobs and Steps. A Job defines the processing flow of the batch application and consists of one or more steps. A basic step is comprised of an item reader, item processor and item writer (see Figure 15).

The item reader reads records of data in chunks, for example from a file, and converts them to objects. These objects are then processed by the item processor, which contains the business logic of the batch

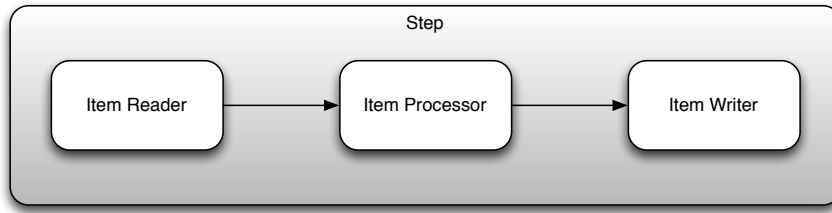


Figure 15: A Step consists of an item reader, item processor and item writer

application. Finally, the processed objects are getting written to the output destination, for example a database, by the item writer.

Listing 1: Mediation batch job definition

```

1 <batch:job id="mediationMultiThreadedJob" incrementer="
  jobRunIdIncrementer">
2   <batch:step id="mediationMultiThreadedStep" next="
     renameFileMultiThreadedStep">
3     <batch:tasklet transaction-manager="batchTransactionManager"
4       start-limit="100"
5       task-executor="taskExecutor" throttle-limit="${batch.step.
        throttle-limit}">
6       <batch:chunk reader="rawUsageMultiThreadedEventReader"
          processor="rawUsageEventProcessor" writer="
            loggingSimpleCdrWriter" commit-interval="1000" />
7     </batch:tasklet>
8   </batch:step>
9   <batch:step id="renameFileMultiThreadedStep">
10    <batch:tasklet ref="renameFileTasklet" />
11  </batch:step>
12</batch:job>

```

Listing 1 shows the definition of the mediation batch job *mediationMultiThreadedJob*. It consists of two steps, the *mediationMultiThreadedStep* (line 2) and the *renameFileMultiThreadedStep* (line 10). The step *mediationMultiThreadedStep* is multithreaded and uses 10 threads for processing. It consists of a *rawUsageMultiThreadedEventReader* (line 6), a thread safe reader implementation that reads call detail records from the input file and converts them to objects, a *rawUsageEventProcessor*, that processes the call detail objects by calling the mediation business logic and a *loggingSimpleCdrWriter* (line 7), which writes the processed call detail objects to the output file. The step uses an commit interval of 1000, meaning that the input data is processed in chunks of 1000 records. After the input file has been processed by the *mediationMultiThreadedStep* it is getting renamed to its final name by the *renameFileMultiThreadedStep* (line 10).

The mediation batch job is integrated using Apache Camel. Listing 2 shows the definition of the mediation batch route.

Listing 2: Mediation batch route definition

```

1 public void configure() {
2     from("file:data/input")
3     .to("spring-batch:mediationMultiThreadedJob?jobLauncherRef=
        jobLauncher");
4
5     from("file:data/output")
6     .to("ftp://billing@localhost/src/data?password=billing");
7 }

```

It consists of two routes, the first route listens on the file system for incoming files (line 2) and calls the mediation batch job, when a file arrives (line 3). The second route transfers the output file of the mediation batch job to the rating batch node using [FTP](#) (line 5-6).

Listing 3 shows the definition of the rating batch job *ratingMultiThreadedJob*. It consists of a single step *ratingMultiThreadedStep* (line 2), which is comprised of a *simpleCdrMultiThreadedItemReader*, which reads the normalized call detail records written by the mediation batch node, a *simpleCdrProcessor*, that processes the normalized call detail records by calling the rating business logic and a *costedEventWriter*, which writes the processed costed events to the Costed Events database (line 4).

Listing 3: Rating batch job definition

```

1 <batch:job id="ratingMultiThreadedJob" incrementer="
    jobRunIdIncrementer">
2   <batch:step id="ratingMultiThreadedStep">
3     <batch:tasklet transaction-manager="batchTransactionManager"
        start-limit="100" task-executor="taskExecutor" throttle-
        limit="${batch.step.throttle-limit}">
4       <batch:chunk reader="simpleCdrMultiThreadedItemReader"
        processor="simpleCdrProcessor" writer="
        costedEventWriter" commit-interval="1000" />
5     </batch:tasklet>
6   </batch:step>
7 </batch:job>

```

4.2.3 Messaging prototype

The messaging prototype implements the billing prototype utilizing the message-oriented processing type. It uses Apache Camel ([Apache Camel, 2014](#)) as the messaging middleware.

Figure 16 shows the architecture of the messaging prototype. It consists of three nodes, the billing route, mediation service and rating service. The billing route implements the main flow of the application. It is responsible for reading messages from the billing queue, extracting the payload, calling the mediation and rating service and writing the

processed messages to the database. The mediation service is a web-service representing the mediation component. It is a SOAP service implemented using Apache CXF and runs inside an Apache Tomcat container. The same applies to the rating service, representing the rating component.

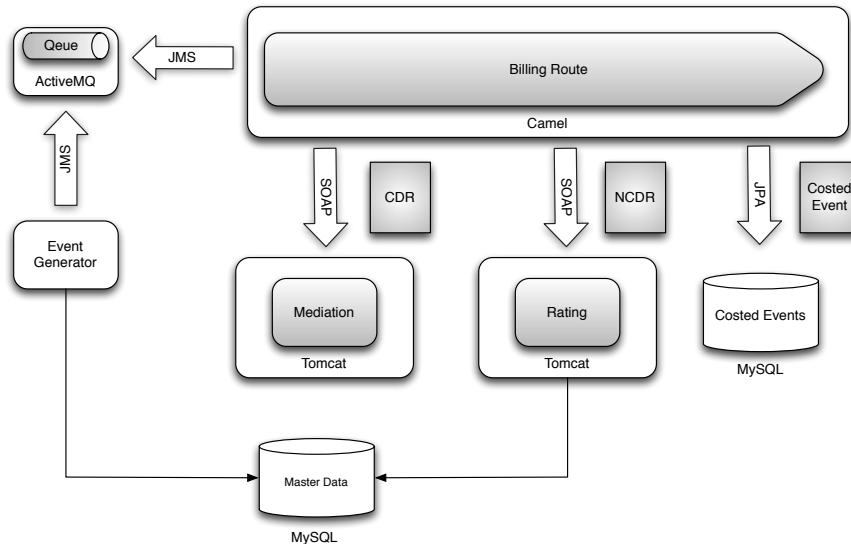


Figure 16: Message-based prototype

Listing 4 shows the definition of the billing route using the Apache Camel fluent Application Programming Interface (API). The billing route performs the following steps:

1. The message is read from the billing queue using [JMS](#) (line 5). The queue is hosted by an Apache ActiveMQ instance.
2. The message is unmarshalled using Java Architecture for XML Binding ([JAXB](#)) (line 6).
3. The *Mediation service* is called by the CXF Endpoint of the billing route (line 7)
4. The response of the *Mediation webservice*, the normalized call detail record, is unmarshalled (line 8).
5. The *Rating service* is called by the CXF Endpoint of the billing route (line 9).
6. The response of the *Rating webservice*, that is the costed event, is unmarshalled (line 10).
7. The costed event is written to the *Costed Events* DB (line 11).

If an error occurs during the processing of an event, it is written to an error [JMS](#) queue (line 3).

Listing 4: Billing route definition

```

1 public void configure() {
2
3     errorHandler(deadLetterChannel("activemq:queue:BILLING.ERRORS")
4         );
5
6     from("activemq:queue:BILLING.USAGE_EVENTS")
7         .unmarshal("jaxbContext")
8         .to("cxf:bean:mediationEndpoint?dataFormat=POJO&
9             defaultOperationName=processEvent")
10        .process(new ProcessEventPostProcessor())
11        .to("cxf:bean:ratingEndpoint?dataFormat=POJO&
12            defaultOperationName=processCallDetail")
13        .process(new ProcessCallDetailPostProcessor())
14        .process(costedEventProcessor);
15 }

```

4.3 PERFORMANCE EVALUATION

To compare the performance characteristics of the two processing types, batch processing and message-based processing, a performance evaluation has been conducted with the main focus on latency and throughput.

This section describes the approach and the results of the performance evaluation.

4.3.1 *Measuring points*

A number of measuring points have been defined for each prototype by breaking down the processing in single steps and assigning a measuring point to each step. Figure 17 and 18 show the measuring points of the batch prototype and the messaging prototype.

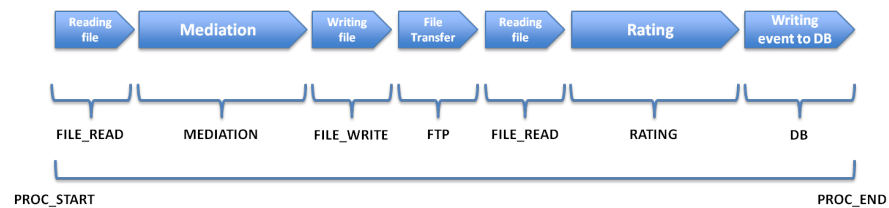


Figure 17: Measuring points of the batch prototype

A detailed description of each point is shown in Table 4 and 5.

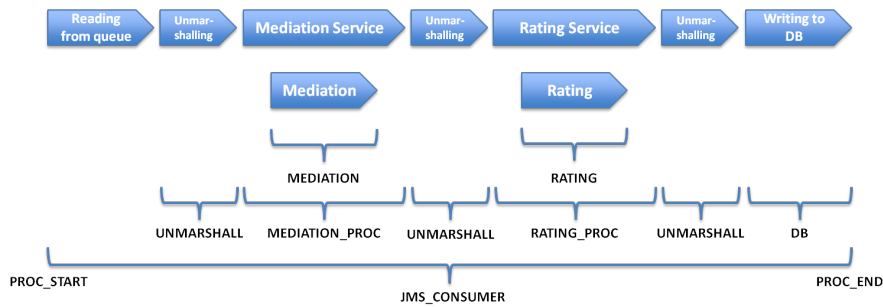


Figure 18: Measuring points of the messaging prototype

Table 4: Measuring points of the batch prototype

Measuring point	Description
PROC_START	Timestamp denoting the start of processing an event
PROC_END	Timestamp denoting the end of processing an event
FILE_READ	Elapsed time for reading events from file
MEDIATION	Elapsed time used by the mediation component
FILE_WRITE	Elapsed time for writing events to file
FTP	Elapsed time for file transfer using FTP
RATING	Elapsed time used by the rating component
DB	Elapsed time for writing event to the database

Table 5: Measuring points of the messaging prototype

Measuring point	Description
PROC_START	Timestamp denoting the start of processing an event
PROC_END	Timestamp denoting the end of processing an event
JMS_CONSUMER	Elapsed time processing a single event
UNMARSHALL	Elapsed time for unmarshalling an event
MEDIATION_PROC	Elapsed time needed for calling the mediation service
MEDIATION	Elapsed time used by the mediation component
RATING_PROC	Elapsed time needed for calling the rating service
RATING	Elapsed time used by the rating component
DB	Elapsed time for writing event to the database

4.3.2 Instrumentation

A logging statement for each measuring point has been added at the appropriate code location of the prototypes using different techniques.

1. **Directly in the code**

Whenever possible, the logging statements have been inserted directly in the code. This has been the case, when the code that should be measured, has been written exclusively for the prototype, for example the mediation and rating components.

2. **Delegation**

When the code to instrument has been part of a framework that is configurable using Spring, an instrumented delegate has been used.

3. **AOP**

Finally, when the code that should get instrumented was part of a framework that was not configurable using Spring, the logging statements have been added using aspects, which are woven into the resulting class files using AspectJ.

4.3.3 Test environment

The two prototypes have been deployed to an Amazon EC2 environment to conduct the performance evaluation, with the characteristics described in Table 6.

4.3.3.1 Batch prototype

The batch prototype comprises two EC2 nodes, the *Mediation Node* and the *Rating Node*, containing the *Mediation Batch* and the *Rating Batch*, respectively. The *Costed Event Database* is hosted on the *Rating Node* as well. Figure 19 shows the deployment diagram of the Batch prototype.

4.3.3.2 Messaging Prototype

The messaging prototype consists of three EC2 nodes, as shown in Figure 20. The *Master Node* hosts the *ActiveMQ Server* which runs the JMS queue containing the billing events, the *Billing Route*, which implements the processing flow of the prototype and the *MySQL Database* containing the *Costed Event Database*. The *Mediation Node* and *Rating Node* are containing the *Mediation Service* and *Rating Service*, respectively, with each service running inside an Apache Tomcat container.

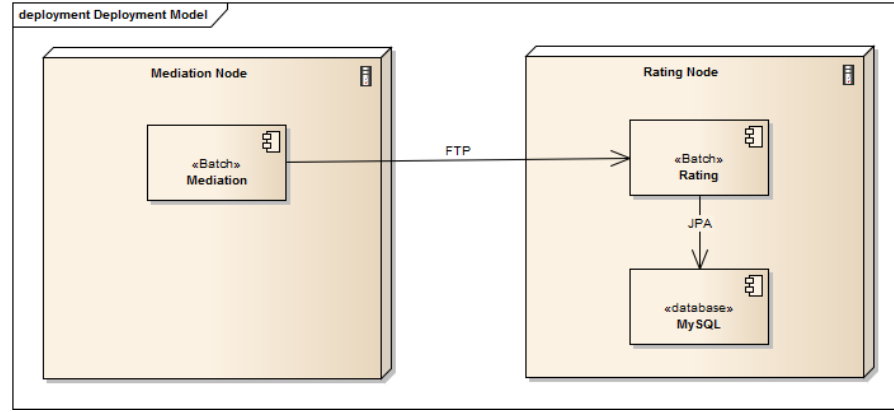


Figure 19: Batch prototype deployment on EC2 instances

4.3.4 Clock Synchronization

The clocks of the *Mediation Node* and *Rating Node* are synchronized with the clock of the *Master Node* using PTPd (*PTP daemon (PTPd)*, 2013), an implementation of the Precision Time Protocol (PTP) (IEEE, 2008). The clock of the *Master Node* itself is synchronised with a public timeserver using the Network Time Protocol (NTP). Using this approach, a sub-millisecond precision is achieved.

Table 6: Amazon EC2 instance configuration

Instance type	M1 Extra Large (EBS optimized)
Memory	15 GiB
Virtual Cores	8 (4 cores × 2 units)
Architecture	64-bit
EBS Volume	10 GiB (100 IOPS)
Instance Store Volumes	1690 GB (4x420 GB Raid 0)
Operating System	Ubuntu 12.04 LTS (GNU/Linux 3.2.0-25-virtual x86_64)
Database	MySQL 5.5.24
Messaging Middleware	Apache ActiveMQ 5.6.0

4.3.5 Preparation and execution of the performance tests

For running the performance tests, the Master Data DB has been set up with a list of customers, accounts, products and tariffs with each prototype using the same database and data. While part of the test-data like the products and tariffs have been created manually, the relationship between the customers and the products have been generated by a test data generator.

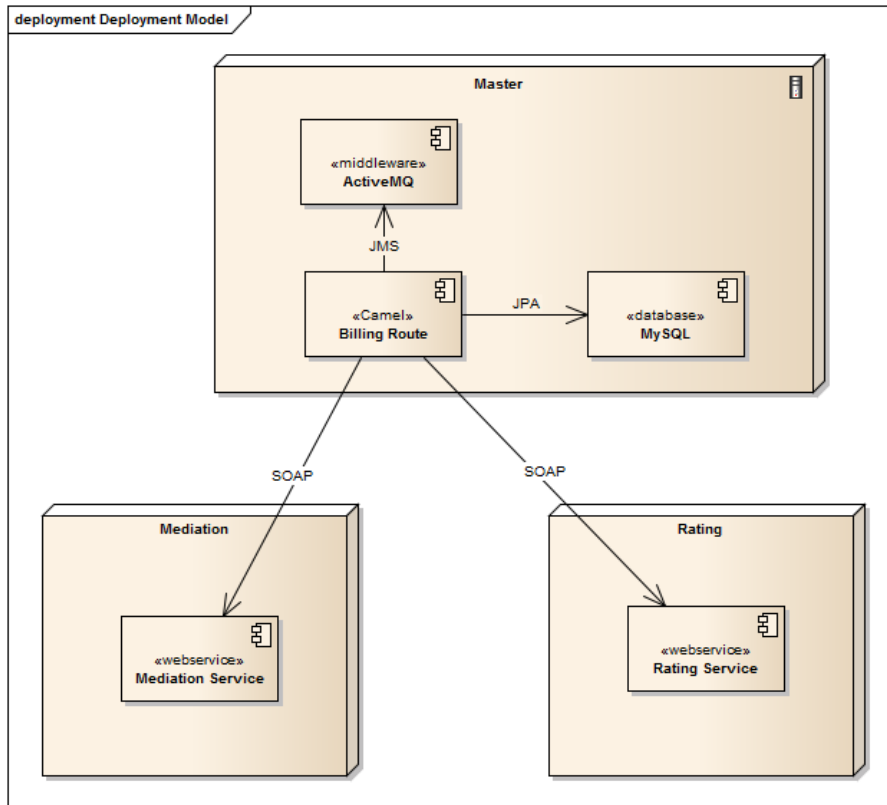


Figure 20: Messaging prototype deployment on EC2 instances

After setting up the master data, a number of test runs have been executed using different sizes of test data (1.000, 5.000, 10.000, 50.000, 100.000, 500.000, 1.000.000 records). To get reliable results, each test configuration has been run three times. Out of the three runs for each configuration, the run having the median processing time has been used for the evaluation.

For each test run, the following steps have been executed:

1. **Generating test data**

In case of the batch prototype, the event generator writes the test data to file. In case of the messaging prototype, the event generator writes the test data to a [JMS](#) queue.

2. **Running the test**

Each prototype listens on the file system and the [JMS](#) queue, respectively. Using the batch prototype, the processing starts when the input file is copied to the input folder of the mediation batch application by the event generator. Using the messaging prototype, the processing starts when the first event is written to the JMS queue by the test generator.

3. **Validating the results**

Processing the log files written during the test run

4. Cleaning up

Deleting the created costed events from the DB.

Before running the tests, each prototype has been warmed up by processing 10.000 records.

4.3.6 Results

The performance evaluation yields the following results.

4.3.6.1 Throughput

The throughput per second for a test run with N records is defined as

$$TP/s_N = N/PT_N$$

with PT_N being the total processing time for N records. Figure 21 shows the measured throughput of the batch and messaging prototypes. The messaging prototype is able to process about 70 events per second. The maximum throughput of the batch prototype is about 383 records per second which is reached with an input of 1.000.000 records.

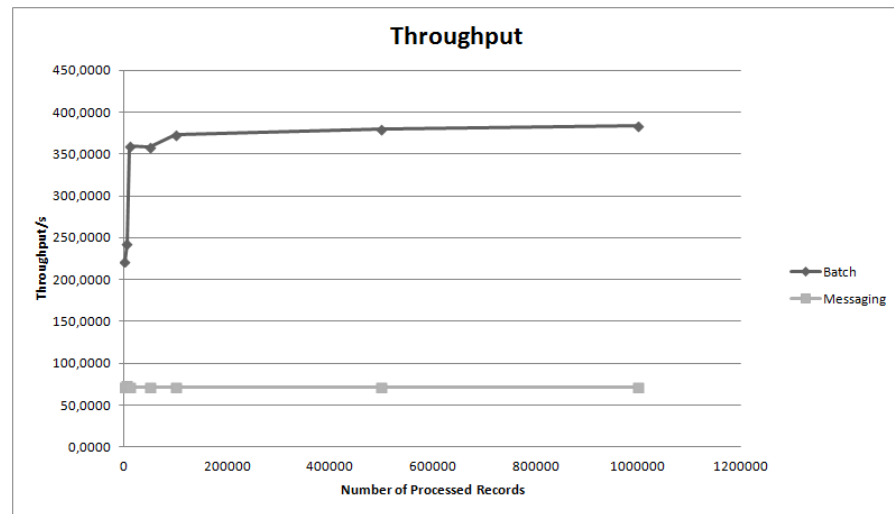


Figure 21: Throughput

4.3.6.2 Latency

Figure 22 shows the measured latencies of the batch and messaging prototypes. To rule out peaks, the 95th percentile has been used, that is, 95% of the measured latencies are below this value. In case of the batch prototype, the 95th percentile latency is a linear function of the amount of data. The latency increases proportionally to the number of processed records. In case of the messaging prototype, the

95th percentile latency is approximately a constant value which is independant of the number of processed records.

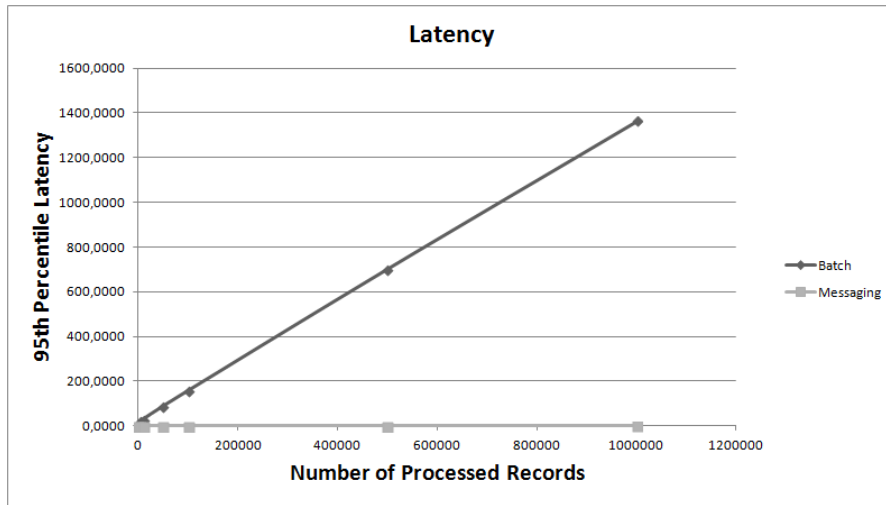


Figure 22: Latency

4.3.6.3 Processing overhead

The overhead of the batch prototype is about 7% of the total processing time, independant of the number of processed records, as shown in Figure 23. This overhead contains file operations, such as opening, reading, writing and closing of input files, the file transfer between the Mediation and Rating Nodes and the database transactions to write the the processed event to the Costed Events DB.

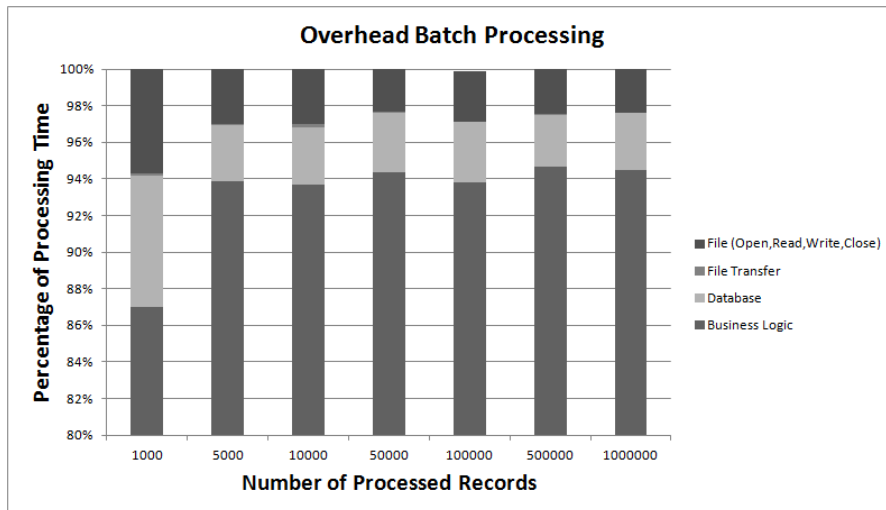


Figure 23: Overhead batch prototype

On the contrary, the overhead of the messaging prototype is about 84% of the total processing time (see Figure 24). In case of the messaging prototype, the overhead contains the JMS overhead, that is the

overhead for reading events from the message queue, the webservice overhead needed for calling the Mediation and Rating services including marshalling and unmarshalling of input data and the overhead caused the database transactions to write the processed events to the Costed Events DB. Most of the overhead is induced by the webservice overhead and the database overhead. Since every event is written to the database in its own transaction, the database overhead of the messaging prototype is much larger than the database overhead of the batch prototype.

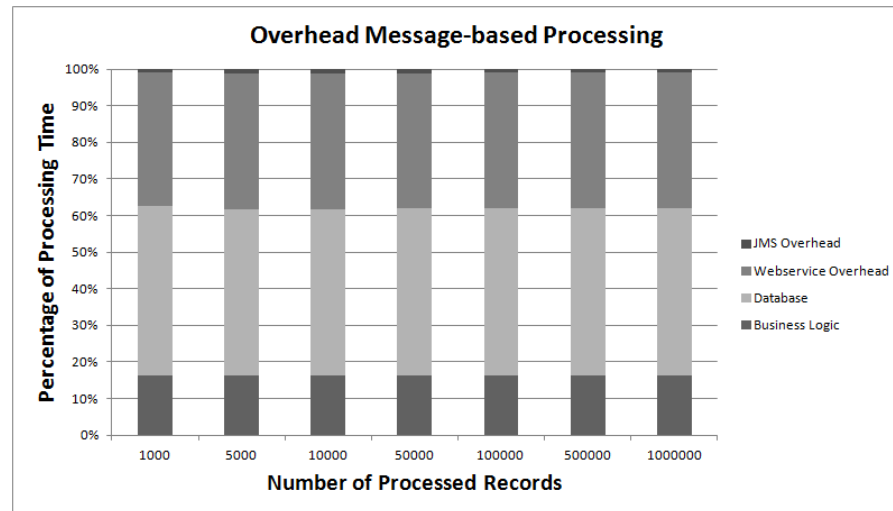


Figure 24: Overhead messaging prototype

4.3.6.4 System utilisation

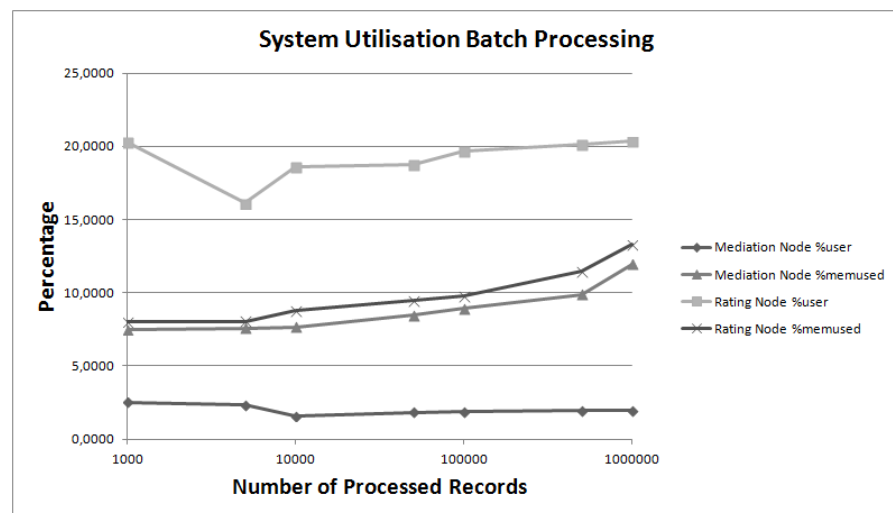


Figure 25: System utilisation batch prototype

The system utilisation has been measured using the sar (System Activity Report) command while running the performance tests. Fig-

ure 25 shows the mean percentage of CPU consumption at the user level (%user) and the mean percentage of used memory (%memused) for the Mediation node and Rating node of the Batch prototype. The CPU utilisation of Medation Node and Ratig Node is about 2% and 19%, respectively. The memory utilisation increases slowly with the number of processed records.

Figure 26 shows the mean CPU consumption and mean memory usage for the nodes of the Messaging prototype. The CPU utilisation of the Master Node, Mediation Node and Rating Node is about 9%, 1% and 6%, respectively. As the same with the batch prototye, the memory utilisation of the messaging prototype increases with the number of processed records. The memory utilisation of the master node peaks at about 38% with 500000 processed records. With 1000000 processed records, the memory utilisation is only about 25%, which presumably can be accounted to the garbage collector.

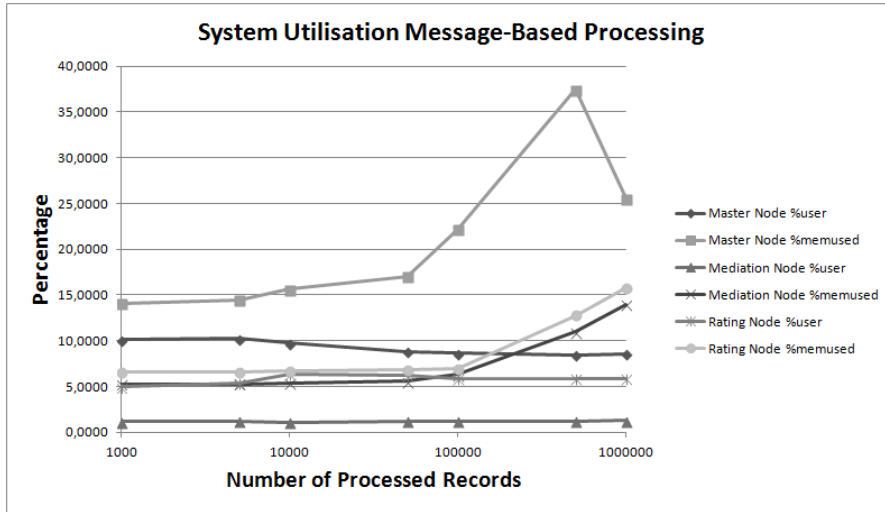


Figure 26: System utilisation messaging prototype

4.4 IMPACT OF DATA GRANULARITY ON THROUGHPUT AND LATENCY

The results presented in Section 4.3.6 suggest that the throughput of the messaging prototype can be increased by increasing the granularity of the data that is beeing processed. Data granularity relates to the amount of data that is processed in a unit of work, for example in a single batch run or an event. In order to examine this approach, we have repeated the performance tests using different package sizes for processing the data.

For this purpose, the messaging prototype has been extended to use an aggregator in the messaging route. The aggregator is a stateful filter which stores correlated messages until a set of messages is complete and sends this set to the next processing stage in the mes-

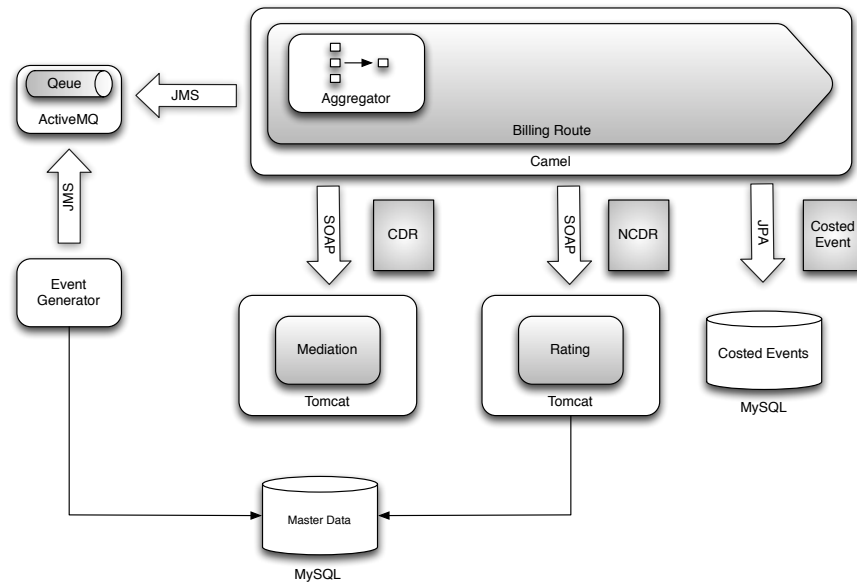


Figure 27: The data granularity is controlled by an aggregator

saging route. In case of the messaging prototype, messages are not correlated to each other and also the messages can be processed in an arbitrary order. A set of messages is complete when it reaches the configured package size. In other scenarios, it is possible to correlate messages by specific data, for example an account number or by a business rule.

Listing 5 shows the definition of the billing route using the aggregator processor, which is provided by Apache Camel (line 7). The aggregator is configured using the correlation expression constant (true), which simply aggregates messages in order of their arrival and the aggregation strategy UsageEventsAggrationStrategy, which implements the merging of incoming messages with already merged messages. The aggregation size is set by completionSize. The specific value is set in a configuration file. As a fallback, completionTimeout defines a timeout in milliseconds to send the set of aggregated messages to the next processing stage before it has reached the defined aggregation size. parallelProcessing indicates that the aggregator should use multiple threads (default is 10) to process the finished sets of aggregated messages.

Listing 5: Billing route definition with an additional aggregator

```

1 public void configure() {
2
3     errorHandler(deadLetterChannel("activemq:queue:BILLING.ERRORS")
4         );
5     from("activemq:queue:BILLING.USAGE_EVENTS")
6         .unmarshal("jaxbContext")

```

```

7      .aggregate(constant(true), new UsageEventsAggrationStrategy()
8          ).completionSize(completionSize).completionTimeout(
          completionTimeout).parallelProcessing()
9      .to("cxf:bean:mediationEndpoint?dataFormat=POJO&
          headerFilterStrategy=#dropAllMessageHeadersStrategy&
          defaultOperationName=processEvents")
10     .process(new ProcessEventsPostProcessor())
11     .to("cxf:bean:ratingEndpoint?dataFormat=POJO&
          headerFilterStrategy=#dropAllMessageHeadersStrategy&
          defaultOperationName=processCallDetails")
12     .process(new ProcessCallDetailsPostProcessor())
13     .process(costedEventsProcessor);

```

Figure 28 shows the impact of different aggregation sizes on the throughput of the messaging prototype. For each test 100.000 events have been processed. The throughput increases constantly for

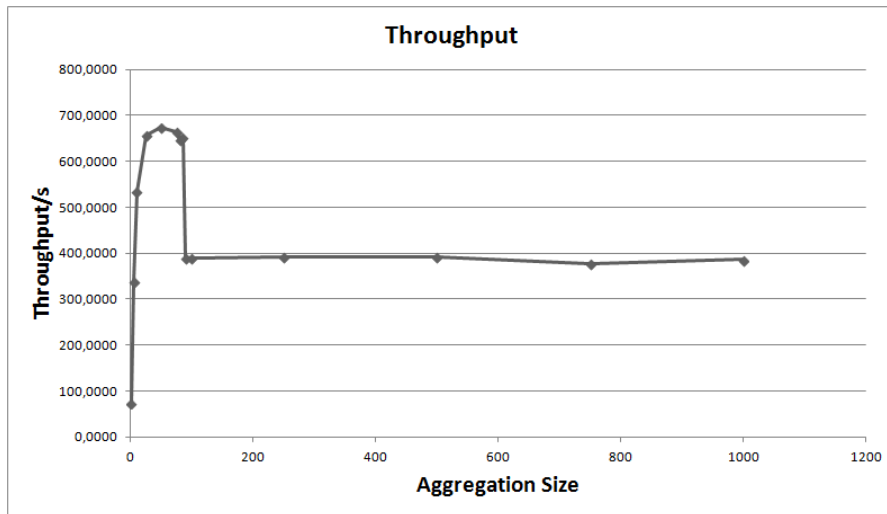


Figure 28: Impact of different aggregation sizes on throughput

$1 < \text{aggregation_size} \leq 50$ with a maximum of 673 events per second with $\text{aggregation_size} = 50$. Higher aggregation sizes than 50 do not further increase the throughput, it stays around 390 events per second. Surprisingly, the maximum throughput of 673 events per second even outperforms the throughput of the batch prototype which is about 383 records per second. This is presumably a result of the better multithreading capabilities of the camel framework.

Increasing the aggregation size also decreases the processing overhead, as shown in Figure 29. An aggregate size of 10 decreases the overhead by more than 50% compared to an aggregate size of 1. Of course, the integration of the aggregator adds an additional overhead which is insignificant for $\text{aggregation_size} > 50$.

The increased throughput achieved by increasing the aggregation size comes with the cost of a higher latency. Figure 30 shows the

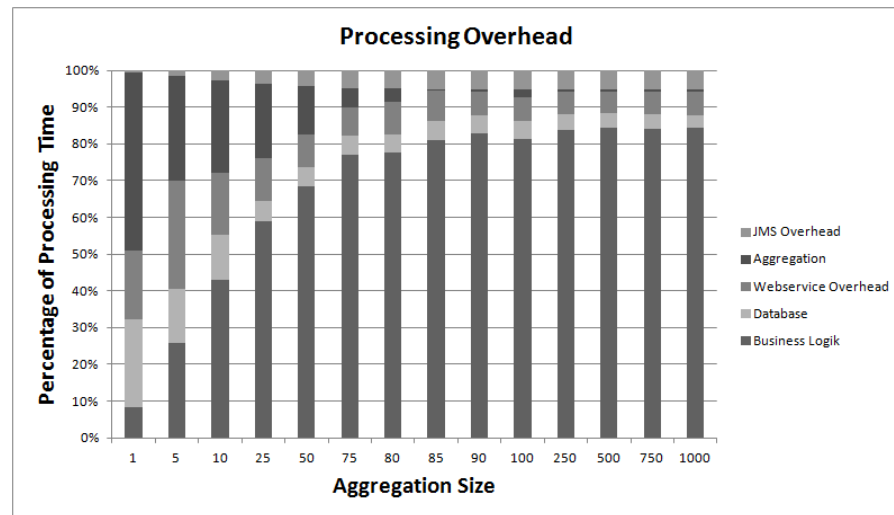


Figure 29: Impact of different aggregation sizes on processing overhead

impact of different aggregation sizes on the 95th percentile latency of the messaging prototype.

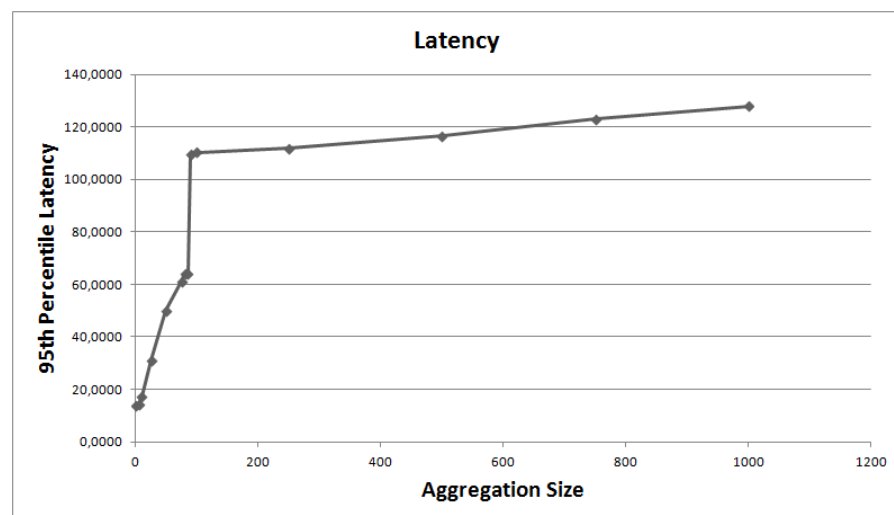


Figure 30: Impact of different aggregation sizes on latency

An aggregation size of 50, resulting in the maximum throughput of 673 events per seconds, shows a 95th percentile latency of about 68 seconds. This latency is significantly higher than the latency of the messaging system without message aggregation, which is about 0,15 seconds (see Section 4.3.6.2).

The results indicate that there is an optimal range for the aggregation size to control the throughput and latency of the system. Setting the aggregation size higher than a certain threshold leads to a throughput drop and latency gain. In case of our prototype, this threshold is between an aggregation size of 85 and 90. The observed throughput drop and latency gain is caused by a congestion in the

aggregator. Messages are read faster from the queue than they are getting processed by the aggregator.

Figure 31 shows the impact of different aggregation sizes on the system utilisation. The CPU utilisation of the Master node shows a maximum of 30% with an aggregation size of 25. An aggregation_size ≥ 90 results in a CPU utilisation of about 15%. The maximum memory utilisation of the Master node is 41% with an aggregation size of 100.

The maximum system utilisation of the Rating node is 25% with an aggregation size of 80. The memory utilisation is between 7-8% irrespective of aggregation size. Maximum system and memory utilisation of the Mediation node are also irrespective of aggregation size, being less than 2% and 8%, respectively.

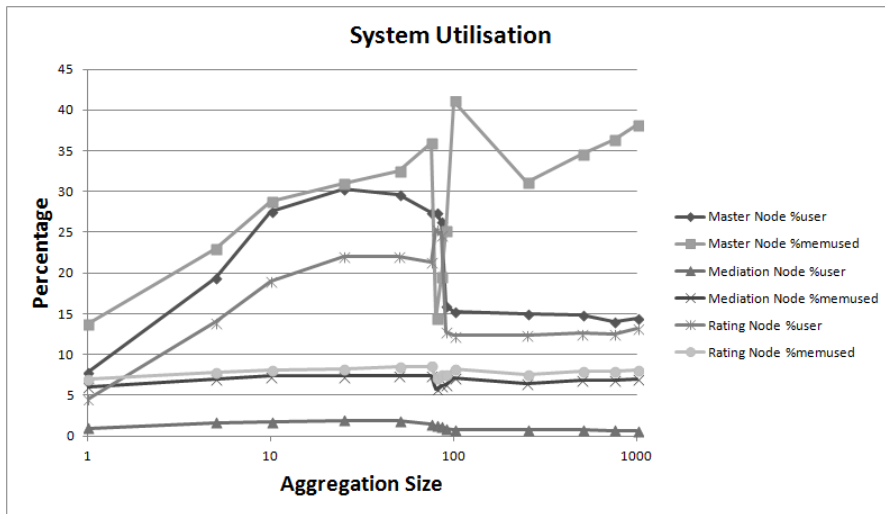


Figure 31: Impact of different aggregation sizes on system utilisation

When using high levels of data granularity, the messaging system is essentially a batch processing system, providing high throughput with high latency. To provide near-time processing an optimum level of data granularity would allow having the lowest possible latency with the lowest acceptable throughput.

4.5 DISCUSSION WITH RESPECT TO RELATED WORK

This section gives an overview of work related to the performance evaluation of batch and message-based systems presented in this chapter and discusses the approach that has been taken.

Related work can be categorised in two different topics, performance measuring and performance prediction. Performance measuring is applied to evaluate if an implemented system meets its performance requirements and to spot possible performance problems.

While performance measuring can only be done when the relevant parts of a system are already implemented, performance prediction allows to predict the performance of a system in an early stage

of development, before the system is available. It uses performance modelling to build a model of the system, which is then used for the performance evaluation. Common approaches for performance modelling use queueing networks, petri nets or simulations (Balsamo et al., 2004).

4.5.1 *Performance Modelling*

Performance modeling allows to predict the performance of a system in an early stage of development. It facilitates for example capacity and resource planning before the system is already available or helps to evaluate design alternatives in regard of their performance impact.

Brebner (2008) developed a tool for performance modeling of Service-Oriented Architectures. It is comprised of SOA models, a simulation engine and a graphical user interface. The SOA models are generated from architectural artifacts such as UML sequence or deployment diagrams and automatically transformed into runtime models for execution.

An approach to predict the performance of J2EE applications using messaging services using queueing network models has been presented by Liu and Gorton (2005). As opposed to prior approaches, their solution models the underlying component infrastructure that implements the messaging service which allows an accurate prediction with an error within 15% when compared to the real performance of the implemented system.

In another work, Liu et al. (2007) developed a performance model of an service-oriented application based on an Enterprise Service Bus using a queuing network. Their modeling approach includes the following steps:

- Mapping of application components of the design level to analytical model elements
- Characterisation of workload patterns for the application components used as input for performance model
- Calibrating the performance model
- Validating the performance model

D'Ambrogio and Bocciarelli (2007) describe “a model-driven approach for integrating performance prediction into service composition processes carried out by use of BPEL (Business Process Execution Language for Web Services).” Using their approach, a BPEL process is described using an UML model. The model is automatically annotated with performance data and transformed into a Layered Queueing Network which is used to predict the performance of the BPEL

process. For the automatic annotation of the model, a performance-oriented extension to WSDL is utilised called P-WSDL (D'Ambrogio, 2005).

Instead of using models to compare batch and message-based processing systems, a prototype for each processing type has been built. Using prototypes in this case has the following advantages over a modelling approach:

- It is difficult to build a model since every relevant aspect needs to be modelled, such as data transfer, data marshalling, database transactions.
- The relevant aspects for modelling the processing types were initially not known.
- By using state-of the art technologies and frameworks for the prototype implementation, the relevant aspects for comparing the different processing types come for “free”.
- The effort to build a prototype is a compromise between creating model and a real application.

4.5.2 *Performance Measuring and Evaluation*

Her et al. (Her et al., 2007) propose the following set of metrics for measuring the performance of a service-oriented system:

- **Service response time**
Elapsed time between the end of request to service and the beginning of the response of the service. This metric is further split in 20 sub-metrics such as message processing time, service composition time and service discovery time.
- **Think time**
Elapsed time between the end of a response generated by a service and the beginning of a response of an end user.
- **Service turnaround time**
Time needed to get the result from a group of related activities within a transaction.
- **Throughput**
Number of requests served at a given period of time. The authors distinguish between the throughput of a service and the throughput of a business process.

In their work, Henjes et al. (2006); Menth et al. (2006b) investigated the throughput performance of the JMS server FioranaMQ, SunMQ and WebsphereMQ. The authors came to the following conclusion:

- Message persistence reduces the throughput significantly.
- Message replication increases the overall throughput of the server.
- Throughput is limited either by the processing logic for small messages or by the transmission capacity for large messages.
- Filtering reduces the throughput significantly.

Chen and Greenfield (2004) propose that the following performance metrics should be used to evaluate a JMS server:

- Maximum sustainable throughput
- Latency
- Elapsed time taken to send batches messages
- Persistent message loss after recovery

The authors state that “although messaging latency is easy to understand, it is difficult to measure precisely in a distributed environment without synchronised high- precision clocks.” They discovered that latencies increase with increasing message sizes.

SPECjms2007 is a standard benchmark for the evaluation of Message-Oriented Middleware platforms using JMS (Sachs et al., 2009). It provides a flexible performance analysis framework for tailoring the workload to specific user requirements. According to Sachs et al. (2007), the workload of the SPECjms2007 benchmark has to meet the following requirements:

- **Representativeness**
The workload should reflect how the messaging platform is used in typical user scenarios.
- **Comprehensiveness**
The workload should incorporate all platform features typically used in JMS application including publish/subscript and point-to-point messaging.
- **Focus**
The workload should focus on measuring the performance of the messaging middleware and should minimize the impact of other components and services.
- **Configurability**
It should be possible to configure the workload to meet the requirements of the user.
- **Scalability**
It should be possible to scale the workload by the number of destinations with a fixed traffic per destination or by increasing the traffic with a fixed set of destinations.

Ueno and Tatsubori (2006) propose a methodology to evaluate the performance of an ESB in an early stage of development that can be used for capacity planning. Instead of using a performance model for performance prediction, they run the ESB on a real machine with a pseudo-environment using lightweight web service providers and clients. The authors state that model-based approaches “often require elemental performance measurements and sophisticated modeling of the entire system, which is usable not feasible for complex systems”.

Related research is concerned with the performance of messaging middleware such as JMS servers or ESB middleware. In the research presented in this chapter, an end-to-end performance evaluation of a batch and messaging prototype implementation has been conducted instead.

4.6 SUMMARY

Near-time processing of bulk data is hard to achieve. As shown in Section 2.4, latency and throughput are opposed performance metrics of a system for bulk data processing. Batch processing, while providing high throughput, leads to high latency, which impedes near-time processing. Message-base processing delivers low latency but cannot provide the throughput for bulk data processing due to the additional overhead for each processed message.

While it is technically possible to minimise the overhead of a messaging system by implementing a lightweight marshalling system and not use JMS or other state-of-the-art technologies such as XML, SOAP or REST, it would hurt the ability of the messaging middleware to integrate heterogenous systems or services and thus limiting its flexibility, which is one the main selling propositions of such a middleware. Furthermore, batch processing enables optimizations by partitioning and sorting the data appropriately which is not possible when each record is processed independently as a single message.

In order to compare throughput and latency of batch and message-oriented systems, a prototype for each processing type has been built. A performance evaluation has been conducted with the following results:

- The throughput of the batch prototype is 4 times the throughput of the messaging prototype.
- The latency of the messaging prototype is only a fraction of the latency of the batch prototype.
- The overhead of the messaging prototype is about 84% of the total processing time, which is mostly induced by the webservice overhead and the database transactions.

- The overhead of the batch prototype is only about 7% of the total processing time.

The results presented in Section 4.4 show that throughput and latency depend on the granularity of data that is being processed.

- The throughput increases constantly for an aggregation size > 1 and ≤ 50 with a maximum of 673 events per second with an aggregation size = 50.
- The increased throughput achieved by increasing the aggregation size comes with the cost of a higher latency. An aggregation size of 50, resulting in the maximum throughput of 673 events per seconds, shows a 95th percentile latency of about 68 seconds. This latency is significantly higher than the latency of the messaging system without message aggregation, which is about 0,15 seconds.
- Increasing the aggregation size also decreases the processing overhead of the messaging prototype. An aggregate size of 10 decreases the overhead by more than 50% compared to an aggregation size of 1.
- There is an optimal range for the aggregation size to control the throughput and latency of the system. Setting the aggregation size higher than a certain threshold leads to a throughput drop and latency gain cause by a congestion in the aggregator.

The performance tests that have been run for the evaluation described in section 4.3 are static tests, in the sense that they do not take different load scenarios of the system into account. In a real situation, the current throughput and latency also depend on the current load of the system. If the system is not able to handle the current load, messages are congested in the input queue which increases the latency of the system. A higher maximum throughput would decrease the latency in this case.

Therefore, the aggregation size used by the messaging system should depend on the current load of the system. It is not feasible to find a static aggregation size that works under all load conditions resulting in an optimum latency.

The next chapter presents a solution for this problem. It describes an adaptive middleware that is able to adjust the data aggregation size at runtime, depending on the current load of the system.

AN ADAPTIVE MIDDLEWARE FOR NEAR-TIME PROCESSING OF BULK DATA

5.1 INTRODUCTION

It has been shown in the previous Chapter 4, that the end-to-end latency can be decreased by using a message-based processing style which facilitates single-event processing. While this approach is able to deliver near-time processing, it is hardly capable for bulk data processing due to the additional communication overhead for each processed message. In contrast, the batch processing style delivers high throughput but cannot provide near-time processing of data.

The processing type is usually a fixed property of an enterprise system that is decided when the architecture of the system is designed, prior to implementing the system. This choice depends on the non-functional requirements of the system. These requirements are not fixed and can change during the lifespan of a system, either anticipated or not anticipated.

Additionally, enterprise systems often need to handle load peaks that occur infrequently. For example, think of a billing system with moderate load over most of the time, but there are certain events with very high load such as New Year's Eve. Most of the time, a low end-to-end latency of the system is preferable when the system faces moderate load. During the peak load, it is more important that the system can handle the load at all. A low end-to-end latency is not as important as an optimized maximum throughput in this situation.

The results presented in the previous Chapter 4 show that throughput and latency depend on the granularity of data that is being processed. Additionally, the current throughput and latency also depend on the current load of the system. If the system is not able to handle the current load, messages are congested in the input queue which increases the latency of the system. A higher maximum throughput would decrease the latency in this case. The aggregation size used by the messaging system should depend on the current load of the system.

This chapter introduces the concept of an adaptive middleware which is able to adapt its processing type fluently between batch processing and single-event processing. It continuously monitors the load of the system and controls the message aggregation size. Depending on the current aggregation size, the middleware automatically chooses the appropriate service implementation and transport mechanism to further optimize the processing.

In this chapter, a solution to this problem is proposed:

- The concept of a middleware is presented that is able to adapt its processing type fluently between batch processing and single-event processing. By adjusting the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios.
- A prototype has been built to evaluate the concepts of the adaptive middleware.
- A performance evaluation has been conducted using this prototype to evaluate the proposed concept of the adaptive middleware.

The remainder of this chapter is organized as follows.

Section 5.2 describes the requirements of an adaptive middleware derived from the results of Chapter 4. Section 5.3 introduces the core concepts of the *Adaptive Middleware for Bulk Data Processing*. These concepts are implemented by components of the adaptive middleware that are described in Section 5.4. There are several architectural design aspects that need to be considered to implement a system based on the adaptive middleware, which are discussed in Section 5.5. To evaluate the concepts of the adaptive middleware, a prototype has been built. The design and implementation of this prototype is outlined in Section 5.6. The prototype has been evaluated in Section 5.7. Section 5.8 gives an overview of other work related to this research. Finally, Section 5.9 concludes this chapter.

5.2 REQUIREMENTS

The *Adaptive Middleware* should implement the following requirements, which have been derived from the results of the performance analysis, as described in Chapter 4:

- **REQ1:** Message aggregation
Aggregation of single messages or events
- **REQ2:** Aggregation strategies
Support for different aggregation strategies, statically or dynamically at run-time
- **REQ3:** Message routing
Messages should be routed to the appropriate service to allow for optimized processing depending on their aggregation size.
- **REQ4:** Monitoring
Monitoring of current throughput, end-to-end latency and load of the system

- **REQ5:** Dynamic control of aggregation size
Dynamic control of the aggregation size of the processed events at run-time depending on the current load of the system

5.3 MIDDLEWARE CONCEPTS

Based on the requirements, as discussed in the previous section, this section describes the core concepts of the adaptive middleware: (1) message aggregation, (2) message routing, and (3) monitoring and control.

5.3.1 *Message Aggregation*

Message aggregation or batching of messages is the main feature of the adaptive middleware to provide a high maximum throughput. The aggregation of messages has the following goals:

- To decrease the overhead for each processed message
- To facilitate optimized processing

There are different options to aggregate messages, which can be implemented by the Aggregator:

- **No correlation:** Messages are aggregated in the order in which they are read from the input message queue. In this case, an optimized processing is not simply possible.
- **Technical correlation:** Messages are aggregated by their technical properties, for example by message size or message format.
- **Business correlation:** Messages are aggregated by business rules, for example by customer segments or product segments.

Table 7 describes the advantages and disadvantages of each aggregation strategy.

In Section 4.4, a static aggregation size has been used to optimize the latency and the throughput of a system. This is not feasible for real systems, since the latency and throughput also depends on the load of the system. Therefore, a dynamic aggregation size depending on the current load of the system is needed.

5.3.2 *Message Routing*

The goal of the message routing is to route the message aggregate to the appropriate service, which is either optimized for batch or single event processing, to allow for an optimized processing. Message

Table 7: Properties of different aggregation strategies

Aggregation Strategy	Pro	Con
No correlation	<ul style="list-style-type: none"> – Simple solution – Even distribution of events 	<ul style="list-style-type: none"> – optimization is not or hardly possible
Business correlation	<ul style="list-style-type: none"> – Optimization is possible 	<ul style="list-style-type: none"> – Analysation of processed data needed – No even distribution of data (depending on correlation rule)
Technical correlation	<ul style="list-style-type: none"> – Optimization is possible 	<ul style="list-style-type: none"> – Analysation of processed data needed – Rules can be defined after integration architecture – No even distribution of data (depending on correlation rule), leads to uneven distribution of latency

Table 8: Strategies for message routing

Routing Strategy	Examples	Description
Technical routing	<ul style="list-style-type: none"> – Aggregation size 	Routing is based on the technical properties of a message aggregate.
Content-based routing	<ul style="list-style-type: none"> – Customer segments (e.g. business customers or private customers) 	Routing is based on the content of the message aggregate, that is, what type of messages are aggregated.

routing depends on how messages are aggregated. Table 8 shows the different strategies of message routing.

With high levels of message aggregation, it is not preferred to send the aggregated message payload itself over the message bus using Java Message Service (JMS) or SOAP. Instead, the message only contains a pointer to the data payload, which is transferred using File Transfer Protocol (FTP) or a shared database.

Message routing can be static or dynamic:

- **Static routing:**
Static routing uses static routing rules, that are not changed automatically.
- **Dynamic routing:**
Dynamic routing adjusts the routing rules automatically at runtime, for example depending on Quality of Service (QoS) properties of services. See for example [Bai et al. \(2007\)](#), [Wu et al. \(2008\)](#) or [Ziyaeva et al. \(2008\)](#).

5.3.3 Monitoring and Control

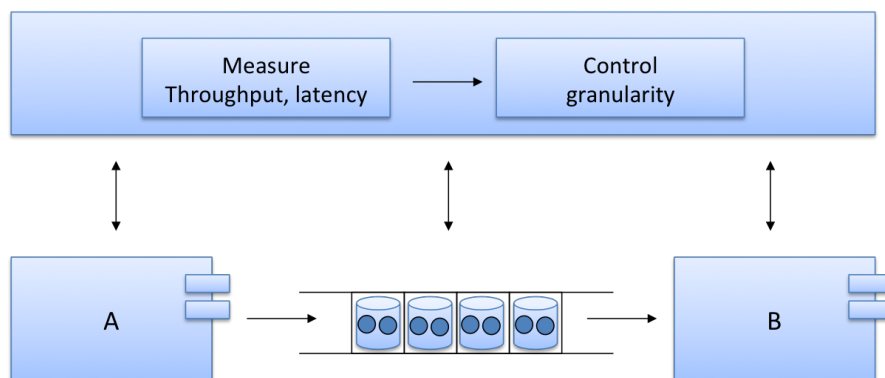


Figure 32: Monitoring and Control

Aggregation size is not static, depends on the current load of the system:

- If the current load of the system is low, aggregation size can be small to provide a low end-to-end latency of the system.
- If the current load of the system is high, aggregation size should be high to provide a high maximum throughput of the system.

Monitoring component continuously monitors the the load of the system.

To control the level of message aggregation at runtime, the adaptive middleware uses a closed feedback loop as shown in Figure 33.

- Control Engineering Methodologies have been identified as a promising solution to implement self-adaptive software systems (Patikirikoral et al., 2012), especially for performance control (Abdelzaher et al., 2003).
- In particular, feedback loops provide generic mechanisms for self-adaption (Brun et al., 2009).
- Control engineering is based on control theory, which provides a systematic approach to designing closed loop systems that are stable, accurate, have short settling times, and do not overshoot (Abdelzaher et al., 2008).

The feedback loop used by the adaptive middleware has the following properties:

- **Input (u):** Current aggregation size
- **Output (y):** Change of queue size measured between sampling intervals
- **Set point (r):** The change of queue size should be zero.

Ultimately, we want to control the average end-to-end latency depending on the current load of the system. The change of queue size seems to be an appropriate quantity because it can be directly measured without a lag at each sampling interval, unlike for example the average end-to-end latency.

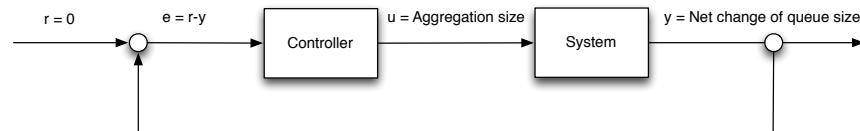


Figure 33: Feedback loop to control the aggregation size




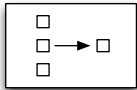
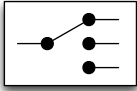
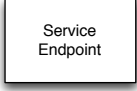
5.4 MIDDLEWARE COMPONENTS

Table 9 shows the components of the middleware, that are based on the Enterprise Integration Patterns described by Hohpe and Woolf (2003).

5.5 DESIGN ASPECTS

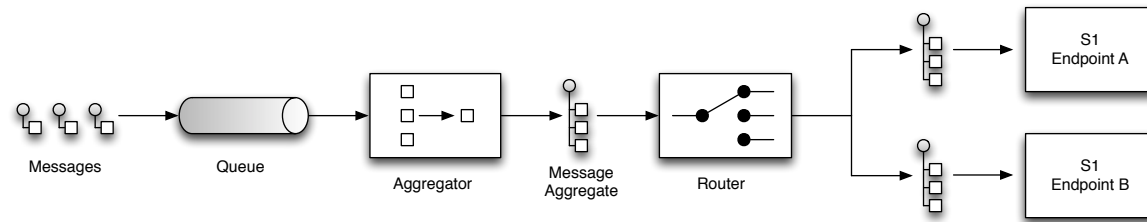
This section describes aspects that should be taken into account when designing an adaptive system for bulk data processing.

Table 9: Components of the Adaptive Middleware. We are using the notation defined by [Hohpe and Woolf \(2003\)](#)

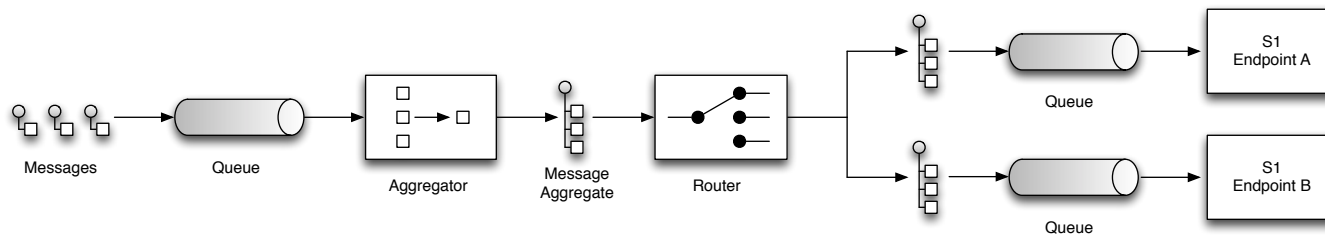
Symbol	Component	Description
	Message	A single message representing a business event.
	Message Aggregate	A set of messages aggregated by the Aggregator component.
	Queue	Storage component which stores messages using the First In, First Out (FIFO) principle.
	Aggregator	Stateful filter which stores correlated messages until a set of messages is complete and sends this set to the next processing stage in the messaging route.
	Router	Routes messages to the appropriate service endpoint, for example depending on the aggregation size of the message.
	Service Endpoint	Represents a business service.

5.5.1 *Usage Scenarios*

do we need this?



(a) request/response integration pattern



(b) point-to-point channel integration pattern

Figure 34: Usage scenarios

- different usage scenarios
- single aggregator, request/response integration pattern
- single aggregator, point to point channel
- system consisting of multiple subsystems, with each subsystem having an input queue, aggregator, router

5.5.2 *Service Design*

The services that implement the business functionality of the system need to be explicitly designed to support the run-time adaption between single-event and batch processing.

There are different options for the design of these services:

- Single Service interface with distinct operations for single and batch processing
 - The service provides different distinct operations for high and low aggregation sizes with optimized implementations for batch and single-event processing. The decision which operation should be called is done by the message router. It is generally not possible to use different transports for different aggregation sizes.
- Single Service interface with a single operation for both single and batch processing
 - The service provides a single operation that is called for all aggregation sizes. The decision which optimization should be used is done by the service implementation. It is not possible to use different transports for different aggregation sizes.
- Multiple service interfaces for single and batch processing (or different aggregation sizes)
 - The logical business service is described by distinct service interfaces which contain operations for either batch processing or single-event processing. The decision which operation should be called is done by the message router. It is possible to use different transports for different aggregation sizes.

The choice of service design relates to where you want to have the logic for the message routing for optimized processing. With a single service offering distinct operations for single-event and batch processing, as well as with distinct service for each processing style, the message router decides which service endpoint should be called. In contrast, using a single service with a single operation for both

processing styles, the service itself is responsible for choosing the appropriate processing strategy. Using a different integration type for each processing style is not possible in this case.

Table 10 shows the advantages and disadvantages for each service design option.

Listing 6 shows the interface of a service offering different operations for batch processing (line 6) and single-event processing (line 10).

Listing 6: Java interface of a web service offering different operations for single and batch processing.

```

1  @WebService
2  @SOAPBinding(style=Style.DOCUMENT, use=Use.LITERAL,
   parameterStyle=ParameterStyle.WRAPPED)
3  public interface RatingPortType {
4      @WebMethod(operationName="processCallDetails")
5      @WebResult(name="costedEvents")
6      public Costedevents processCallDetails(@WebParam(name="
   callDetailRecords") SimpleCDRs callDetailRecords) throws
   ProcessingException, Exception;
7
8      @WebMethod(operationName="processCallDetail")
9      @WebResult(name="costedEvent")
10     public Costedevent processCallDetail(@WebParam(name="
   simpleCDR") SimpleCDR callDetailRecord) throws
   ProcessingException, Exception;
11 }

```

Table 10: Options for designing service interfaces

Interface option	Pros	Cons
Single service / distinct Operations	– Lorem ipsum	– Lorem ipsum
Single service / single operation	–	– Lorem ipsum
Distinct services	–	– Lorem ipsum

5.5.3 Integration and Transports

The integration architecture defines the technologies that are used to integrate the business services. In general, different integration styles with different transports are used for batch processing and single-event processing, which needs to be taken into account when designing an adaptive system for bulk data processing (Please refer to Section 2.2 and 2.3 for a detailed description of each processing style).

When using high aggregation sizes, it is not feasible to use the same transports as with low aggregation sizes. Large messages should not be transferred over the messaging system. Instead, a file based transport using [FTP](#) or database-based integration should be used. When using a messaging system, the payload of large messages should not be transported over the messaging system. For example by implementing the *Claim Check EIP* (refer to Section 2.7 for a detailed description of this pattern). Table 11 summarizes the transport options for low and high aggregation sizes.

Table 11: Transport options for high and low aggregation sizes

Aggregation Size	Transport Options
High	<ul style="list-style-type: none"> – Database – File-based (e.g. FTP) – Claim Check EIP
Low	<ul style="list-style-type: none"> – JMS – SOAP

Additionally, the technical data format should be considered.

The concrete threshold between low and high aggregation sizes depends on the integration architecture and implementation of the system, such as the integration architecture and the deployed messaging system.

The choice of the appropriate integration transport for a service is implicitly implemented by the message router (see Section 5.3.2).

5.5.4 Error Handling

Message aggregation has also an impact on the handling of errors that occur during the processing. Depending on the cause of the error, there are two common types of errors:

- **Technical errors**

Technical errors are errors caused by technical reasons, for example an external system is not available or does not respond within a certain timeout or the processed message has an invalid format.

- **Business errors**

Business errors are caused by violation of business rules, for example a call detail record contains a tariff that is no longer valid.

The following points should be taken into account, when designing the error handling for an adaptive system for bulk data processing:

- Write erroneous messages to an error queue for later processing.
- Use multiple queues for different types of errors, for example distinct queues for technical and business errors to allow different strategies for handling them. Some type of errors can be fixed automatically, for example an error that is caused by an outage of an external system, while other errors need to be fixed manually.
- If the erroneous messages is part of an aggregated message, it should be extracted from the aggregate to prevent the whole aggregate from being written to the error queue, especially when using high aggregation sizes.

5.5.5 Controller Design

There are several approaches for the implementation of feedback-control system. [Hellerstein et al. \(2004\)](#) describe two major steps:

1. modeling the dynamics of the system
2. developing a control system

There are different approaches that are used in practice to model the dynamics of a system ([Hellerstein, 2004](#)):

- Empirical approach
- Black-box modeling
- Modeling using stochastic approaches, especially queuing theory
- Modeling using special purpose representations, for example the first principles analysis

The following approach has been taken in this research:

1. Define the control problem
2. Define the input and output variables of the system
3. Measure the dynamics of the system
4. Create a model of the system
5. Develop the control system

5.5.5.1 *Control Problem*

- Control problem: minimise the end-to-end latency of the system by controlling the message aggregation size
- aggregation size used by the messaging system should depend on the current load of the system
- when system faces high load, aggregation sizes should be increased
- when system faces low load, aggregation sizes could be decreased

5.5.5.2 *Input/Output Variables*

- **Input (u):** Current aggregation size
- **Output (y):** Change of queue size measured between sampling intervals
- **Set point (r):** The change of queue size should be zero.
- advantage of queue size as output variable: queue size can be directly measured without a delay

5.5.5.3 *Control Strategy*

SIMPLE CONTROLLER

A simple control strategy could be implemented as follows:

- change queue > 0: Increase the aggregation size by a certain amount
- change queue = 0: Do nothing

PID CONTROLLER

Another option would be to use a standard PID-Controller instead, which calculates the output value u_k at time step k of the controller depending on the current (proportional part), previous (integral part) and expected future error (differential part):

$$u_k = K_p * e_k + K_i * T_a \sum_{i=0}^k e_i + \frac{K_d}{T_a} (e_k - e_{k-1})$$

with K_p being the controller gain of the proportional part, e_k being the error ($r - y$) at step k , K_i being the controller gain of the integral part, T_a being the sampling interval and K_d being the controller gain of the differential part.

5.6 PROTOTYPE IMPLEMENTATION

This section describes the implementation of the prototype which implements the core concepts of the adaptive middleware. The prototype is based on the messaging prototype described in Section 4.2.3.

Insert image from section 4.2.3

5.6.1 Aggregator

- Implementation is based on the aggregator of the messaging prototype as described in Section 4.4

Listing 7: UsageEventsAggrationStrategy

```

1 public class UsageEventsAggrationStrategy implements
   AggregationStrategy {
2
3     @Override
4     public Exchange aggregate(Exchange oldExchange, Exchange
       newExchange) {
5         if (oldExchange == null) {
6             RawUsageEvent rawUsageEvent = newExchange.getIn().getBody(
               RawUsageEvent.class);
7             RawUsageEvents rawUsageEvents = new RawUsageEvents();
8             List<RawUsageEvent> usageEventList = new ArrayList<
               RawUsageEvent>();
9             rawUsageEvents.setUsageEvents(usageEventList);
10            usageEventList.add(rawUsageEvent);
11            newExchange.getIn().setBody(rawUsageEvents);
12            increaseAggregateSize(newExchange);
13
14            Long startTime = getStartTime(newExchange);
15            addStartTime(newExchange, startTime);
16
17            return newExchange;
18        }
19        else {
20            RawUsageEvents rawUsageEvents = oldExchange.getIn().getBody(
               (RawUsageEvents.class));
21            RawUsageEvent rawUsageEvent = newExchange.getIn().getBody(
               RawUsageEvent.class);
22            rawUsageEvents.getUsageEvents().add(rawUsageEvent);
23            increaseAggregateSize(oldExchange);
24        }

```



```

25     Long startTime = getStartTime(newExchange);
26     addStartTime(oldExchange, startTime);
27
28     return oldExchange;
29 }
30 }
31
32 //Additional methods removed for simplification...
33
34 }

```

Listing 8: Aggregator configuration in definition of BillingRoute

```

1 .aggregate(constant(true), new UsageEventsAggrationStrategy())
2 .completionSize(header(completionSizeHeader))
3 .completionTimeout(completionTimeout)
4 .parallelProcessing()

```

5.6.2 Feedback-Control Loop

Figure 35 shows the components of the feedback-control loop.

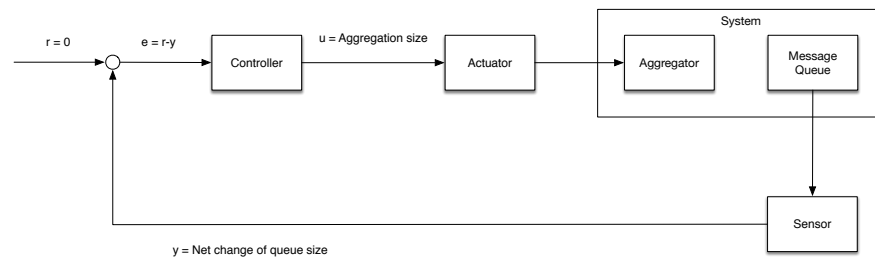


Figure 35: Components of the feedback-control loop

5.6.2.1 Sensor

- Base class JmxSensor
- Reads the current queue length of the ActiveMQ instance using JMX

5.6.2.2 Controller

- ControllerStrategy Interface

Listing 9: ControllerStrategy Interface

```

1 package com.jswiente.phd.performance.controller;
2
3 public interface ControllerStrategy {

```

```

4   public Double getOutput(Double error);
5   }

```

SIMPLE CONTROLLER

PID CONTROLLER

Listing 10: Implementation of PID Controller

```

1 public class PIDController implements ControllerStrategy {
2
3     @Value("${controller.kp}")
4     private Double kp;
5
6     @Value("${controller.ki}")
7     private Double ki;
8
9     @Value("${controller.kd}")
10    private Double kd;
11
12    @Value("${controller.ta}")
13    private Double ta;
14
15    private Double errorSum = 0.0;
16    private Double previousError = 0.0;
17
18    public Double getOutput(Double error) {
19        errorSum = errorSum + error;
20        Double output = kp * error + ki * ta * errorSum + (kd * (
21            error - previousError)/ta);
22        previousError = error;
23        return output;
24    }
25
26    //Setter methods removed for simplification...
27 }

```

5.6.2.3 Actuator

- Interface Actuator
- AggregateSizeActuator
 - Implements Actuator interface
 - Sets the completionSize of the Aggregator by setting a specific header in the currently processed exchange

Listing 11: Actuator Interface

```

1 package com.jswiente.phd.performance.actuator;
2
3 public interface Actuator<T> {
4
5     public void setValue(T value);
6 }

```

Listing 12: AggregateSizeActuator

```

1 @Component
2 public class AggregateSizeActuator implements Processor, Actuator
   <Double> {
3
4     @Value("${camel.aggregator.completionSize}")
5     private long aggregateSize;
6
7     @Value("${camel.aggregator.completionSizeHeader}")
8     private String completionSizeHeader;
9
10    private static final Logger logger = LoggerFactory
11        .getLogger(AggregateSizeActuator.class);
12
13    @Override
14    public void process(Exchange exchange) throws Exception {
15        exchange.getIn().setHeader(completionSizeHeader,
16            aggregateSize);
17    }
18
19    @ManagedAttribute
20    public long getAggregateSize() {
21        return aggregateSize;
22    }
23
24    @ManagedAttribute
25    public void setAggregateSize(long aggregateSize) {
26        logger.debug("Setting aggregateSize to: " + aggregateSize);
27        this.aggregateSize = aggregateSize;
28    }
29
30    @Override
31    public void setValue(Double value) {
32        logger.debug("Actuator: Setting aggregateSize to: " + value);
33        long aggregateSize = Math.round(value);
34        this.setAggregateSize(aggregateSize);
35    }
36 }

```

5.6.3 Load Generator

The *Load Generator* is used to generate the system load by generating events (CDRs) and writing them to the message queue of the system. It is implemented as a stand-alone Java program using a command-line interface.

5.6.3.1 Overview

Figure 36 shows the UML class diagram of the load generator.

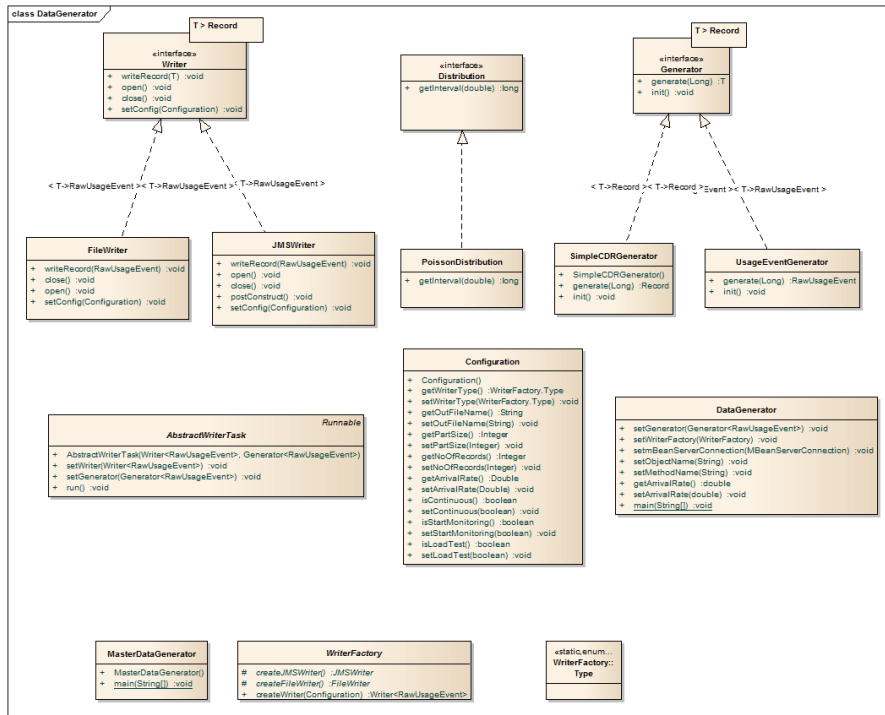


Figure 36: UML class diagram of the *Load Generator*

- Description of main classes

5.6.3.2 Event Distribution

- Poisson Process
 - Events occur continuously and independently of each other
 - Exponentially distributed inter-arrival times

5.7 EVALUATION

The prototype described in the previous section has been used to evaluate the concepts of adaptive middleware.

- Goals of the evaluation

5.7.1 *Test Environment*

- Same test environment has been used as described in Section 4.3.3

5.7.2 *Test Design*

Abdelzaher et al. (2008) define a set of properties, that should be considered when designing feedback-control systems for computing systems, called the SASO properties (Stable, Accurate, Settling times, Overshoot):

- **Stability**
- **Accuracy**
- **Settling time**
- **Overshoot**

5.7.2.1 *Static Tests*

5.7.2.2 *Step Tests*

5.7.2.3 *Dynamic Tests*

- Definition of test cases

5.7.3 *Results*

5.8 DISCUSSION WITH RESPECT TO RELATED WORK

5.8.1 *Adaptive Middleware*

Research on messaging middleware currently focusses on Enterprise Services Bus (ESB) infrastructure. An ESB is an integration platform that combines messaging, web services, data transformation and intelligent routing to connect multiple heterogeneous services (Chappell, 2004). It is a common middleware to implement the integration layer of an Service Oriented Architecture (SOA) and is available in numerous commercial and open-source packages.

Several research has been done to extend the static service composition and routing features of standard ESB implementations with dynamic capabilities decided at run-time, such as dynamic service composition (Chang et al., 2007), routing (Bai et al., 2007) (Wu et al., 2008) (Ziyaeva et al., 2008) and load balancing (Jongtaveesataporn and Takada, 2010).

Work to manage and improve the Quality of Service (QoS) of ESB and service-based systems in general is mainly focussed on dynamic

service composition and service selection based on monitored QoS metrics such as throughput, availability and response time (Calinescu et al., 2011). González and Ruggia (2011) propose an adaptive ESB infrastructure to address QoS issues in service-based systems which provides adaption strategies for response time degradation and service saturation, such as invoking an equivalent service, using previously stored information, distributing requests to equivalent services, load balancing and deferring service requests.

5.8.2 *Message Batching*

The adaption strategy of our middleware is to change the message aggregation size based on the current load of the system. Aggregating or batching of messages is a common approach to increase the throughput of a messaging system, for example to increase the throughput of total ordering protocols (Friedman and Renesse, 1997) (Friedman and Hadad, 2006) (Romano and Leonetti, 2012) (Didona et al., 2012).

5.8.3 *Dynamic Scaling*

A different solution to handle infrequent load spikes is to automatically instantiate additional server instances, as provided by current Platform as a Service (PaaS) offerings such as Amazon EC2 (Amazon EC2 Auto Scaling, n.d.) or Google App Engine (Auto Scaling on the Google Cloud Platform, n.d.). While scaling is a common approach to improve the performance of a system, it also leads to additional operational and possible license costs. Of course, our solution can be combined with these auto-scaling approaches.

5.8.4 *Feedback Control of Computing Systems*

- General definition of Feedback Control
- Properties of Feedback Control Systems
- Open-Loop and Closed-Loop Control Systems
- Applications for Feedback Control
 - Feedback-Control of Software Performance and QoS
- Feedback Control of Computing Systems (Hellerstein et al., 2004)
 - Applications of Control Theory to Computing Systems
 - Examples of Feedback Control Systems
- Introduction to Control Theory And Its Application to Computing Systems (Abdelzaher et al., 2008)

- Challenges of Feedback Control of Computing Systems (Hellerstein, 2004)
- A Systematic Survey on the Design of Self-Adaptive Software Systems using Control Engineering Approaches (Patikirikoral et al., 2012)
- Control Systems application in Java based Enterprise and Cloud Environments A Survey (Gullapalli et al., 2011)
- Engineering Self-Adaptive Systems through Feedback Loops (Brun et al., 2009)
- Feedback-Control of Software Performance and QoS
 - Feedback Performance Control in Software Services (Abdelzaher et al., 2003)
 - ControlWare: A Middleware Architecture for Feedback Control of Software Performance (Zhang et al., 2002)
 - Intelligent Enterprise Application Servers: A Vision for Self-Managing Performance (Kumar et al., 2013)
 - Throughput Improvement in Distributed Systems - An Investigation on Java Messaging Servers using Adaptive Control
 - Self-regulating Message Throughput in Enterprise Messaging Servers - A Feedback Control Solution (Kumar et al., 2012)

5.9 SUMMARY

In this paper, we have presented a middleware that is able to adapt itself to changing load scenarios by fluently shifting the processing type between single event and batch processing. The middleware uses a closed feedback loop to control the end-to-end latency of the system by adjusting the level of message aggregation depending on the current load of the system. Determined by the aggregation size of a message, the middleware routes a message to appropriate service endpoints, which are optimized for either single-event or batch processing.

To evaluate the proposed middleware concepts, we have implemented a prototype system and performed preliminary performance tests. The tests show that throughput and latency of a messaging system depend on the level of data granularity and that the throughput can be increased by increasing the granularity of the processed messages.

Next steps of our research are the implementation of the proposed middleware including the evaluation and tuning of different con-

troller architectures, performance evaluation of the proposed middleware using the prototype and developing a conceptional framework containing guidelines and rules for the practitioner how to implement an enterprise system based on the adaptive middleware for near-time processing

A CONCEPTUAL FRAMEWORK TO GUIDE THE DEVELOPMENT OF FEEDBACK-CONTROLLED BULK DATA PROCESSING SYSTEMS

6.1 INTRODUCTION

The concept for an adaptive Middleware for bulk data processing presented in chapter 5 describes the “What” (what needs to be done) but not the “How” (how should it be done).

The design, implementation and operation of such a system differs from common approaches to implement enterprise systems:

- There are specific activities or tasks needed to implement the feedback-control subsystem.
- There are roles needed with different skills.
- There are different tools needed to aid the design and development of such a system.

Developing software is a complex process, the quality of a software product depends on the people, the organisation and procedures used to create and deliver it (Fuggetta, 2000). In order to guide the implementation of an adaptive system for bulk data processing, a conceptual framework is needed. It defines artifacts, roles, tasks and their dependencies, and processes to describe the necessary steps for design, implementation and operation of a system described in Chapter 5.

Figure 37 shows an overview of the conceptual framework. It is organized among the phases plan, build and run. Each phase contains tasks, which are relevant for each phase:

- **Plan**
Contains tasks for designing the business and technical architecture of the system, tasks for defining and evaluating performance tests, and tasks for managing the development process.
- **Build**
Contains tasks for implementing the system, such as implementing the integration architecture, implementing the feedback-control subsystem, and tuning the the controller.
- **Run**
Contains tasks for operating the system, such as monitoring, setup and tuning.

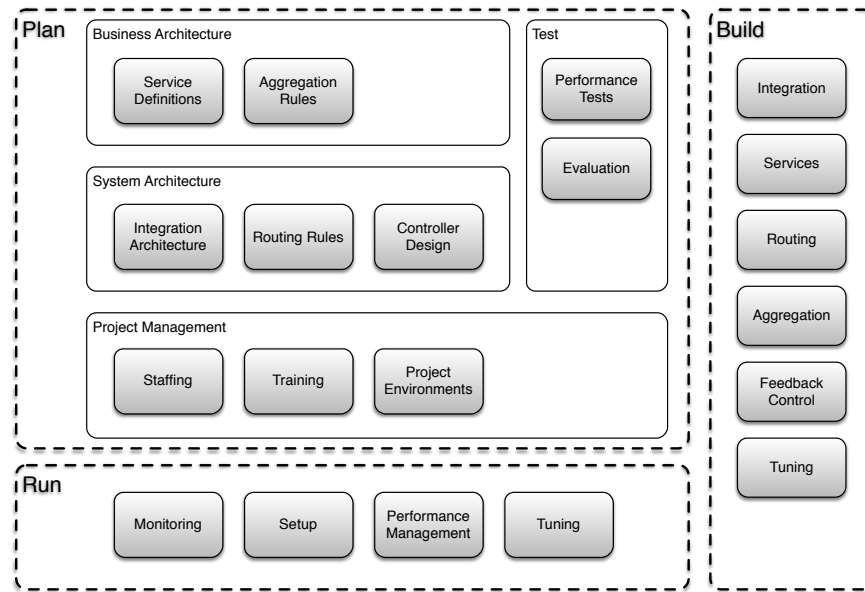


Figure 37: Overview of Conceptual Framework

The conceptual framework uses [UML 2.0](#) notation elements and diagrams, such as class diagrams, activity diagrams and use-case diagrams to describe the development process. It consists of the following packages, as shown in Figure 38:

- **Metamodel**
Contains elements and class-diagrams for describing the meta-model of the conceptual framework.
- **Tasks**
Contains elements describing the tasks of the conceptual framework.
- **Roles**
Contains elements and use-case diagrams for describing the roles of the conceptual framework.
- **Processes**
Contains activity diagrams for describing the processes of the conceptual framework.
- **Artifacts**
Contains elements to describe the artifacts of the conceptual framework.
- **Phases**
Contains elements to describe the phases of the conceptual framework.

- **Tools**

Contains elements to describe the tools needed for processing the tasks of the conceptual framework.

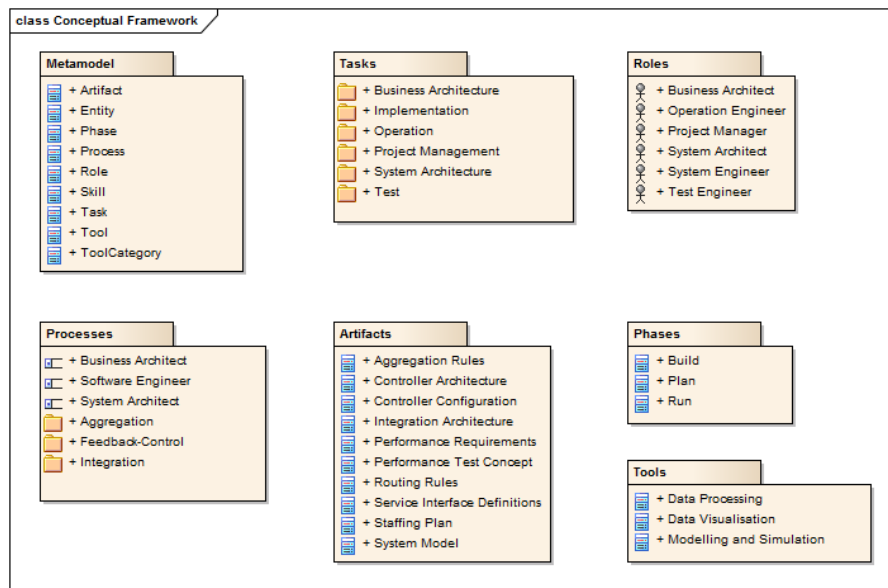


Figure 38: Package structure

The conceptual framework only describes concepts that are specific to the design and implementation of an Adaptive Middleware as described in the previous chapter. It does not describe common concepts for software development.

This chapter is organised as follows:

- Section 6.2 describes the metamodel of the conceptual framework.
- The entities of the process model, roles, tasks, artifacts and tools, are described in the Sections 6.4, 6.5, 6.7 and 6.8.
- Section 6.9 describes how the conceptual framework can be used with other architectural frameworks and software development methodologies such as TOGAF, Rational Unified Process (RUP) and Scrum.
- Section 6.10 discusses other related approaches and work.
- Finally, this chapter concludes with a summary and a discussion of the presented conceptual framework (see Section 6.11).

6.2 METAMODEL

The conceptual framework consists of the following entities, as shown in Figure 39:

- **Phase**

Phases correspond to the different phases of a software development lifecycle, such as design, implementation and operations and contain the relevant tasks.

- **Task**

Tasks represent the activities of the development process. A task

- is contained in a phase
- is processed by a role
- produces and requires artifacts
- uses tools

- **Role**

Roles represent types of actors with the needed skills to process specific tasks.

- **Artifact**

An artifact represents the result of a tasks. Additionally, an artifact is a requirement of a tasks.

- **Tool**

A tool is used by a tasks to produce its artifact.

- **Process**

A process contains an ordered list of tasks that need to be processed in a certain order.

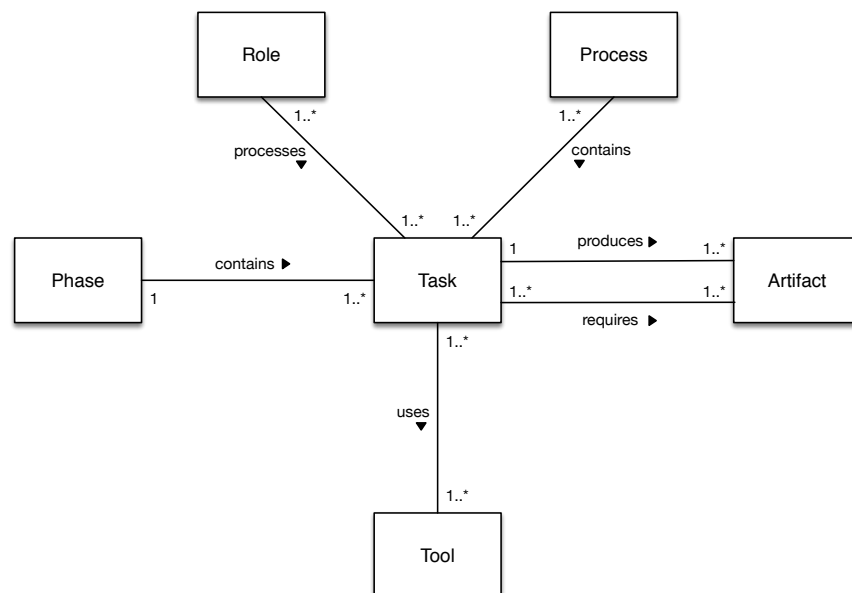


Figure 39: Metamodel

6.3 PHASE

Phases are the top-level entities of the conceptual framework. They correspond to the different phases of the software development lifecycle and are a mean to group the different tasks of the framework.

A phase is described by the following attribute, as shown in Figure 40:

- **Name**
Name of the phase.
- **Description**
Description of the phase.
- **Tasks**
The tasks that the phase contains.
- **Roles**
The needed roles by the phase.

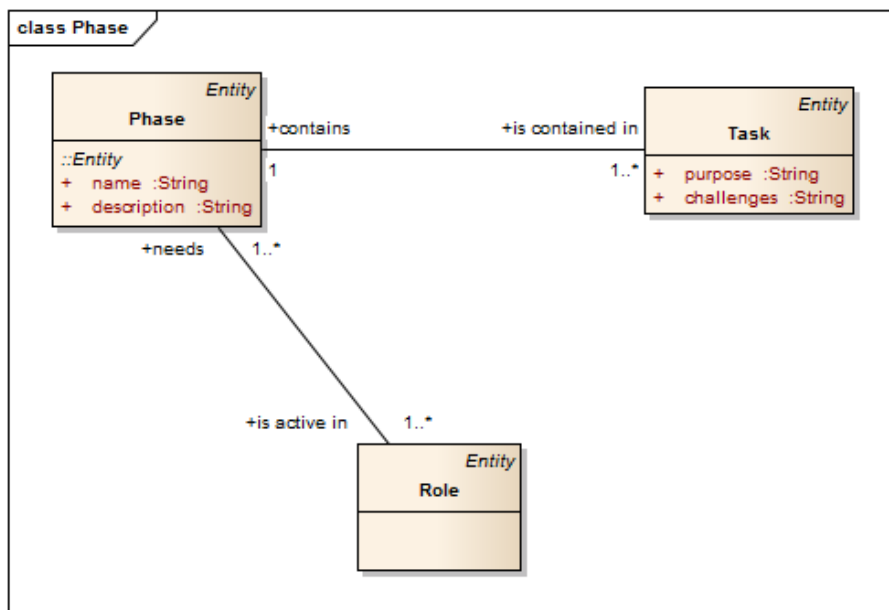


Figure 40: Attributes of a phase

The conceptual framework defines the following phases:

- **Plan**
The plan phase contains tasks relevant for the analysis and design of the system, such as the definition of the service interfaces, definition of the integration architecture and definition of performance tests.
- **Build**
The build phase contains tasks relevant for the implementation

of the system, such as the implementation of services, implementation of the integration layer and the implementation of the feedback-control subsystems.

- **Run**

The run phase contains tasks relevant to the operation of the developed system, such as monitoring, setup and tuning.

It should be noted that the framework defines no requirements regarding the general order or mode in which these phases and their tasks should be processed. It is therefore possible to use this framework with different software development methodologies such as the Waterfall model, Scrum or the V-Modell.

6.3.1 *Plan*

Table 12: Phase: Plan

Phase	Plan
Description	This phase contains tasks concerning the technical and business design of the system.
Tasks	<ul style="list-style-type: none"> • Define Service Interfaces • Define Aggregation Rules • Define Integration Architecture • Define Routing Rules • Define Controller Architecture • Define Performance Tests • Evaluate Test Results • Perform Staffing • Define Training Concept • Source Project Environments

Roles

- Project Manager
- Business Analyst
- System Architect
- Test Engineer

6.3.2 *Build*

Table 13: Phase: Build

Phase	Build
Description	This phase contains tasks concerning the implementation of the system.
Tasks	<ul style="list-style-type: none"> • Implement Integration Architecture • Implement Service Interfaces • Implement Aggregation Rules • Implement Routing Rules • Implement Feedback-Control • Perform Controller Tuning
Roles	Software Engineer

6.3.3 *Run*

Table 14: Phase: Run

Phase	Run
Description	This phase contains tasks concerning the operation of the implemented system in the production environment.

Tasks

- Setup Monitoring Infrastructure
- Setup Test Environment
- Perform Performance Tests

Roles

- Operations Engineer
 - Test Engineer
-

6.4 ROLES

Roles represent the actors, which process tasks, that is, they describe *who* does something. The description of a role contains its responsibilities and needed skills. A role is not the same as a person, a single person can have multiple roles and change the role according to the context of the current task.

A role is described by the following attributes, as shown in Figure 41:

- **Name**
The name of the role.
- **Description**
Description of the responsibilities of the role.
- **Tasks**
The tasks the role is responsible to process.
- **Needed skills**
The skills the role has to have in order to successfully process its tasks.

The Conceptual Framework defines the following roles:

- **Business Architect**
The business architect is responsible for defining the business architecture of the software system.
- **System Architect**
The system architect is responsible for defining the technical architecture of the software system.
- **Software Engineer**
The software engineer is responsible for implementing the software system.

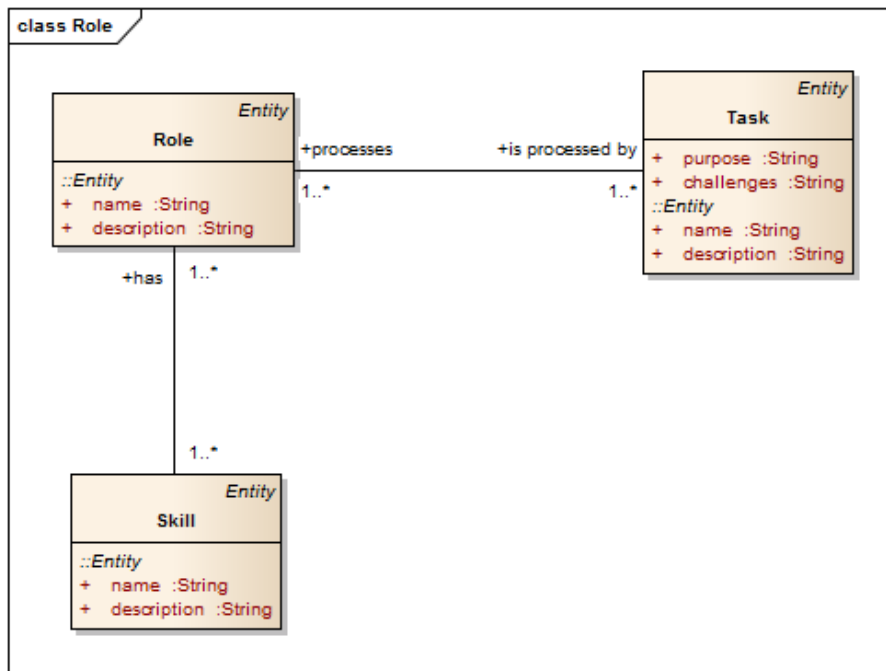


Figure 41: Attributes of a role

- **Test Engineer**
The test engineer is responsible for defining and performing the system test.
- **Operations Engineer**
The operations engineer is responsible for all aspects concerned with running the developed software system.
- **Project Manager**
The project manager is responsible for managing the software development process.

6.4.1 Business Architect

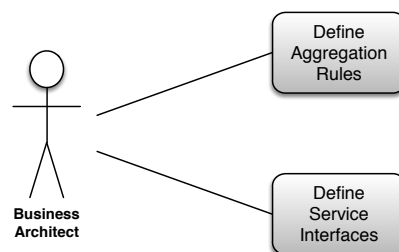


Figure 42: Role: Business Architect

Table 15: Business Architect

Role	Business Architect
Description	The Business Architect is responsible for designing the business architecture of the system, including the definition of services and aggregation rules.
Tasks	<ul style="list-style-type: none"> • Define Service Interfaces • Define Aggregation Rules
Needed skills	<ul style="list-style-type: none"> • Integration styles and patterns, e.g. SOA • Concepts of the Adaptive Middleware for Bulk Data Processing • Business domain knowledge

6.4.2 System Architect

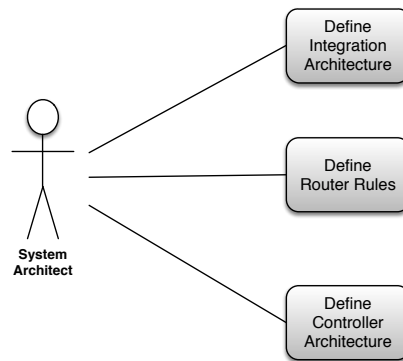


Figure 43: Role: System Architect

Table 16: System Architect

Role	System Architect
Description	The System Architect is responsible for designing the technical architecture of the system, including the integration and controller architecture.
Tasks	<ul style="list-style-type: none">• Define Integration Architecture• Define Controller Architecture
Needed skills	<ul style="list-style-type: none">• System modelling languages, e.g. UML and tools• Integration styles and patterns, e.g. SOA• Processing styles, e.g. batch and single-event processing• Integration middleware technologies and products, e.g. Apache Camel, ESB• Concepts of the Adaptive Middleware for Bulk Data Processing• Control theory

6.4.3 Software Engineer

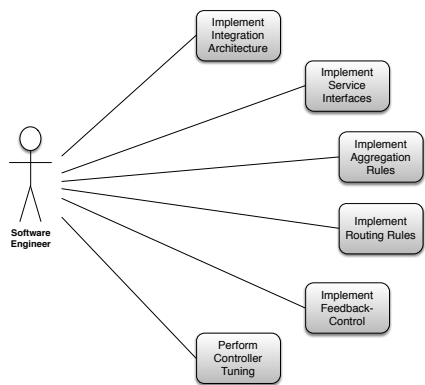


Figure 44: Role: Software Engineer

Table 17: Software Engineer

Role	Developer
Description	The Developer is responsible for the implementation of the system, including the implementation and tuning of the feedback-controll loop.
Tasks	<ul style="list-style-type: none"> • Implement Integration Architecture • Implement Service Interfaces • Implement Aggregation Rules • Implement Routing Rules • Implement Feedback-Control • Perform Controller Tuning
Needed skills	<ul style="list-style-type: none"> • Integration styles and patterns, e.g. SOA • Processing styles, e.g. batch and single-event processing • Batch optimisations • Integration middleware technologies and products, e.g. Apache Camel, ESB • Concepts of the Adaptive Middleware for Bulk Data Processing • Control theory

6.4.4 *Test Engineer*

Table 18: Test Engineer

Role	Test Engineer
Description	The Tester is responsible for defining and performing the performance tests of the system.

Tasks

- Define Performance Tests
- Perform Performance Tests
- Evaluate Performance Tests

Needed skills

- Design and evaluation of performance tests
- Concepts of the Adaptive Middleware for Bulk Data Processing
- Control theory (basics)

6.4.5 Operations Engineer

Table 19: Operations Engineer

Role	Operations Engineer
Description	The Operations Engineer is responsible for operating the system, including setup, deployment and monitoring.

Tasks

- Setup Monitoring Infrastructure
- Setup System Environments
- Perform System Tuning

Needed skills

- Monitoring technologies and products, e.g. [JMX](#)
- Concepts of the Adaptive Middleware for Bulk Data Processing

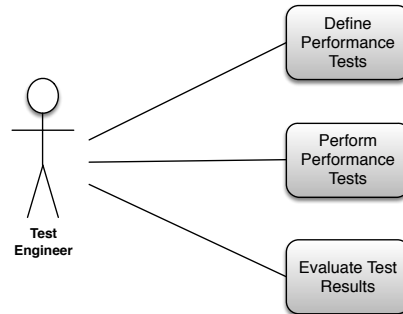


Figure 45: Role: Test Engineer

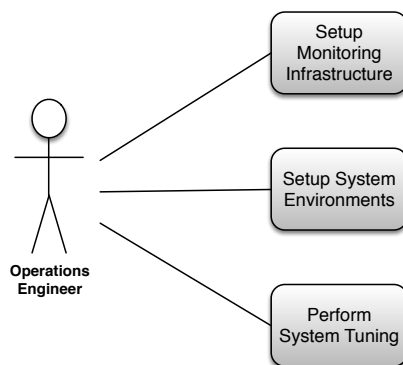


Figure 46: Role: Operations Engineer

6.4.6 Project Manager

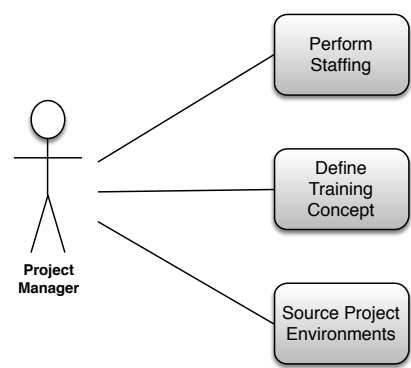


Figure 47: Role: Project Manager

Project Manager	
Table 20: table	
Role	Project Manager
Description	The Project Manager is responsible for the project coordination, including the staffing and planing of the required environments.
Tasks	<ul style="list-style-type: none">• Perform Staffing• Define Training Concept• Source Project Environments
Needed skills	<ul style="list-style-type: none">• Framework for Feedback-Controlled Bulk Data Processing Systems• Concepts of the Adaptive Middleware for Bulk Data Processing

6.5 TASKS

Tasks are the main entities of the conceptual framework. A Tasks describes *what* should be done, *why* should it be done, and *who* should do it. Additionally, it describes the required and produced artifacts,

the tools that should be used to process the task and the expected challenges.

Tasks depend on each other, some tasks must be processed in a certain order. A task can have multiple subtasks.

The Conceptual Framework only describes tasks that are specific to the design and implementation of an Adaptive Middleware for Bulk Data Processing as described in chapter 5. It does not describe common tasks or activities that are needed for every software system.

Figure 48 shows an overview of the tasks grouped by the different phases of the Conceptual Framework.

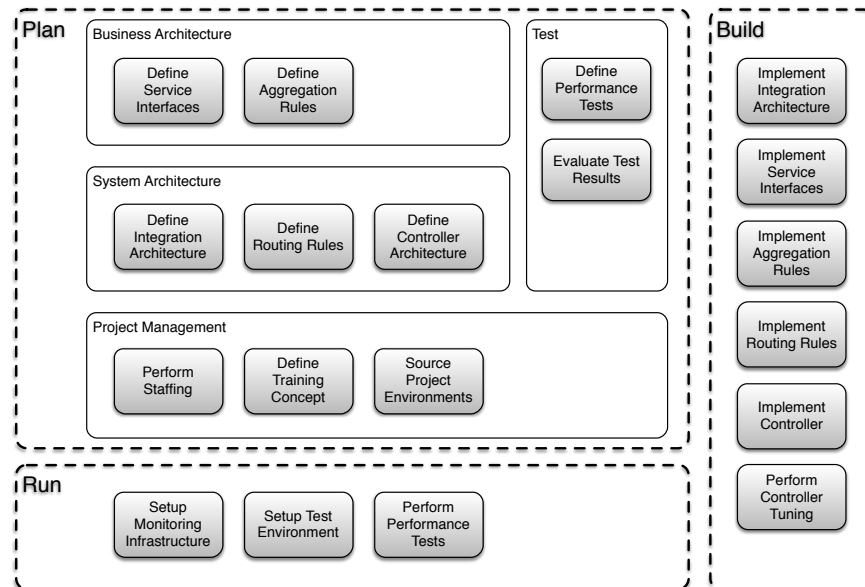


Figure 48: Overview of tasks

Tasks are organised in different packages, as shown in Figure 49:

- **Business Architecture**
Contains tasks concerned with the business architecture of the system.
- **System Architecture**
Contains tasks concerned with the system architecture of the system.
- **Implementation**
Contains tasks concerned with the implementation of the system.
- **Test**
Contains tasks concerned with the test of the system.
- **Operation**
Contains tasks concerned with the operation of the system.

- **Project Management**

Contains tasks concerned with management of the development process.

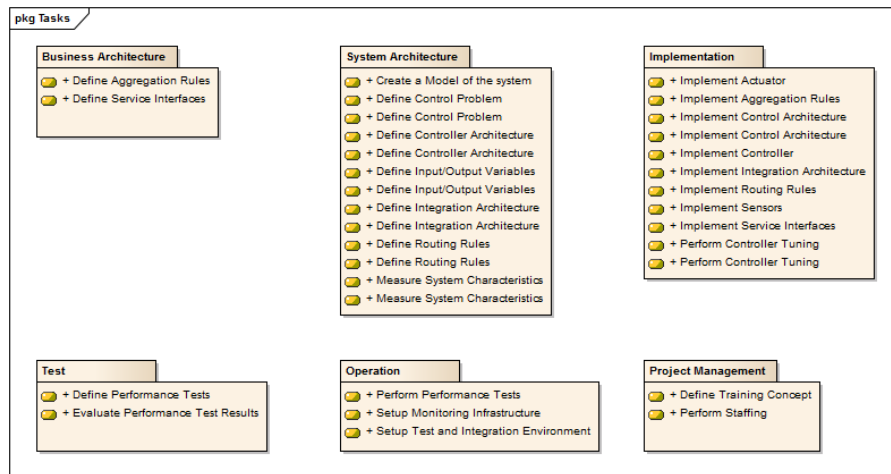


Figure 49: Subpackages of the Tasks package

A Task is described by the following attributes, as shown in Figure 50:

- **Name**
The name of the task.
- **What**
Describes the content of the task.
- **Why**
Describes the purpose of the task.
- **Who**
Describes the roles, that are responsible for processing the task.
- **Input**
The required artifacts of the task.
- **Output**
The artifacts produced by the task.
- **Tools**
The tools that are needed to process the task.
- **Challenges**
Describes the expectable challenges when processing the task.

The following tasks are defined:

- Business Architecture
 - Define Performance Requirements

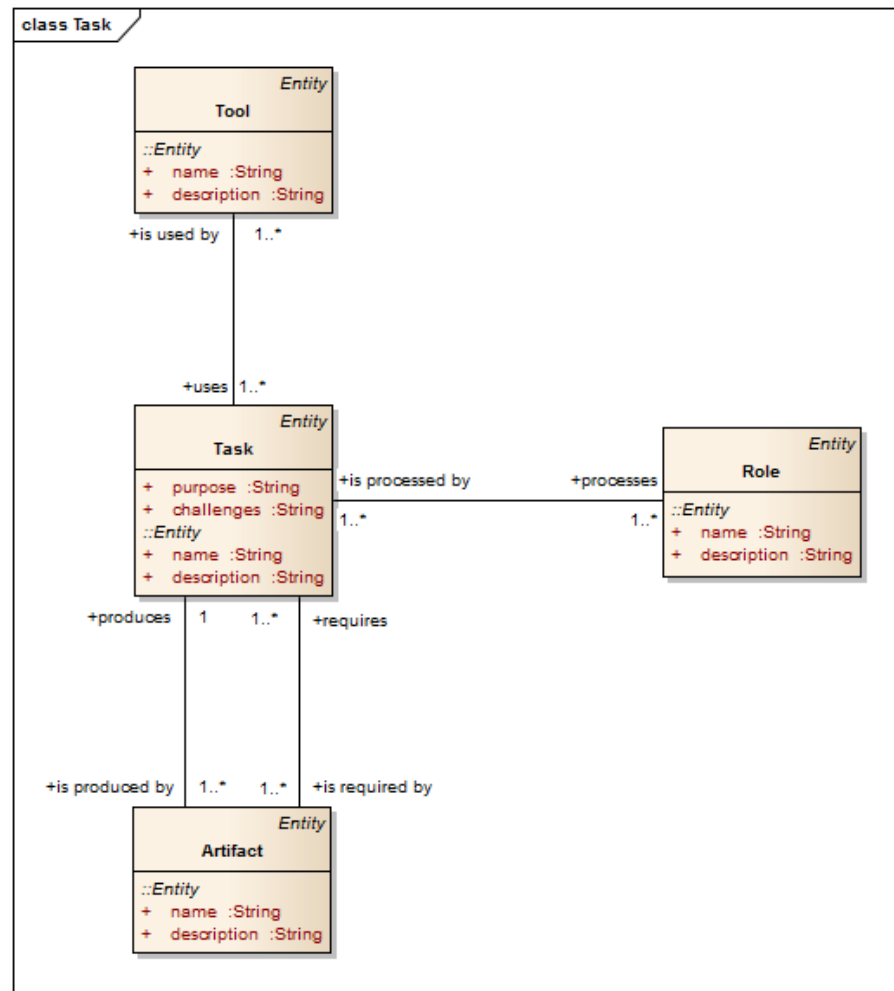


Figure 50: Attributes of a task

- Define Service Interfaces
- Define Aggregation Rules
- System Architecture
 - Define Integration Architecture
 - Define Routing Rules
 - Define Controller Architecture
 - * Define Control Problem
 - * Define Input/Output Variables
 - Define Routing Rules
- Implementation
 - Implement Feedback-Control Loop
 - Create System Model / Perform System Identification
 - Perform Static Tests
 - Perform Step Tests
 - Perform Controller Tuning
 - Implement Integration Architecture
 - Implement Service Interfaces
 - Implement Aggregation Rules
 - Implement Routing Rules
- Test
 - Define Performance Tests
 - Evaluate Performance Test Results
- Operation
 - Setup Monitoring infrastructure
 - Setup Test and Integration Environment
 - Perform Performance Tests
- Project Management
 - Define Training Concept
 - Perform Staffing

6.5.1 *Business Architecture*

This packages contains tasks concerned with defining the business architecture of the system. The business architecture defines the business components of the system and their relationships independantly of the technical implementation. It contains only tasks that are specific to the developement of the adaptive middleware.

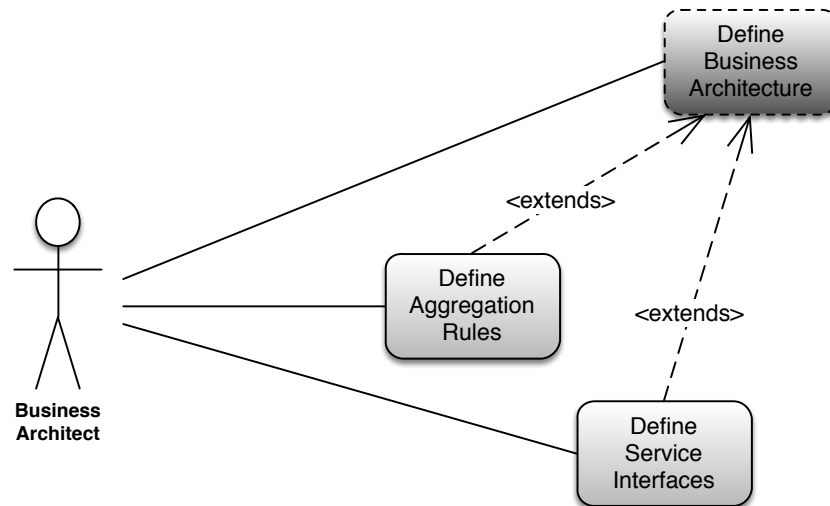


Figure 51: Tasks extending the definition of the business architecture

6.5.1.1 Define Performance Requirements

Table 21: Define Performance Requirements

Task	Define Performance Requirements
What	<p>This task is concerned with the definition of the performance requirements of the system, including</p> <ul style="list-style-type: none"> • Definition of workload scenarios • Definition of the adaptive features of the system • Definition of requirements regarding throughput and latency of the system <ul style="list-style-type: none"> – What is the required range (minimum/maximum) of latency of the system? – What is the required range (minimum/maximum) of throughput of the system?
Why	The performance requirements are needed for the desing of the system architecture.
Who	<ul style="list-style-type: none"> • Busines Analyst • System Architect

Output	Performance Requirements
Challenges	The performance requirements must be explicitly defined in a way that they can be evaluated.

6.5.1.2 Define Service Interfaces

Table 22: Define Service Interfaces

Task	Define Service Interfaces
What	<p>This task is concerned with the definition of the service interfaces, that together implement the business functionality of the system, including:</p> <ul style="list-style-type: none"> • Structuring the functionality of the system into business services • Defining the needed services and their operations. Every service needs operations for single event and batch processing, with the following options (see Section 5.5.2 for details). <ul style="list-style-type: none"> – Distinct operations for batch and single event processing – Common operation for both processing styles • Evaluating which services already exist or need to be implemented or adapted. • Defining the structure of input and output data. This does not include informations about the technical format, such as XML or JSON, and the integration style, such SOAP or REST.
Why	<ul style="list-style-type: none"> • Defines the business components (services) of the system • Basis for the definition of the integration architecture and the implementation of the services
Who	Business Architect
Output	Service Interface Definitions

Challenges Finding the appropriate services and service granularity.

6.5.1.3 *Define Aggregation Rules*

Table 23: Define Aggregation Rules

Task	Define Aggregation Rules
What	<p>This task is concerned with the definition of rules used in the aggregator for correlating events. There are different options for the aggregation (see Section 5.3.1 for details):</p> <ul style="list-style-type: none"> • No correlation: Messages are aggregated in the order in which they are read from the input message queue. In this case, an optimized processing is not simply possible. • Technical correlation: Messages are aggregated by their technical properties, for example by message size or message format. • Business correlation: Messages are aggregated by business rules, for example by customer segments or product segments.
Why	The aggregation Rules are needed by the Aggregator to correlate events.
Who	<ul style="list-style-type: none"> • Business Architect • System Architect
Output	Aggregation Rules
Challenges	<ul style="list-style-type: none"> • Finding aggregation rules that allows for an even distribution of events. • Rules using the technical correlation of events can be defined only after the definition of the integration architecture.

6.5.2 System Architecture

This package contains tasks concerned with the system architecture of the system. The system architecture defines the technical architecture of the system. It contains only tasks that are specific to the development of the adaptive middleware.

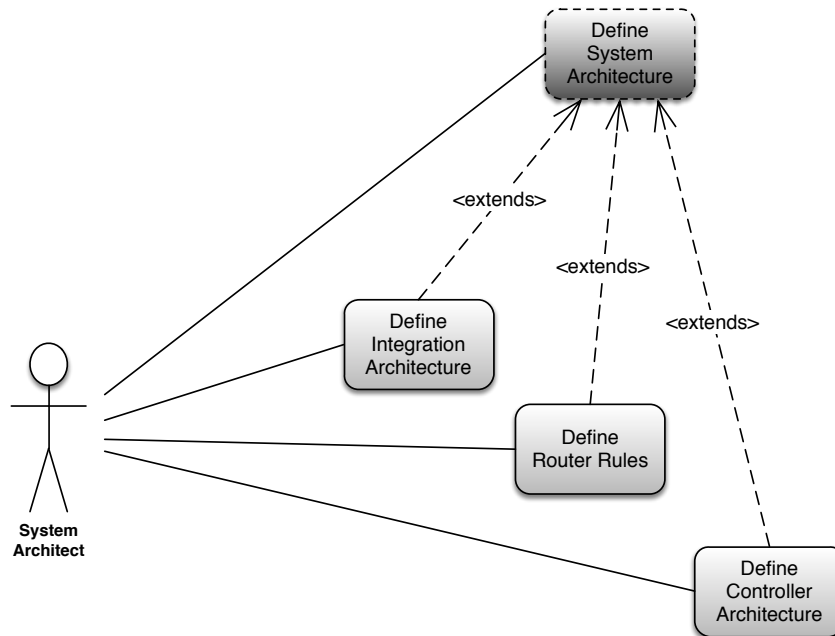


Figure 52: Tasks extending the definition of the system architecture

6.5.2.1 Define Integration Architecture

Table 24: Define Integration Architecture

Task	Define Integration Architecture
------	---------------------------------

What	<p>This task is concerned with the definition of the integration architecture of the system, including</p> <ul style="list-style-type: none"> • Definition of communication styles, such as <ul style="list-style-type: none"> – Synchronous communication – Asynchronous communication • Choosing a middleware technology or product • Definition of transports, for example <ul style="list-style-type: none"> – JMS – SOAP – REST – FTP – DB • Different transports and integration patterns need to be considered for different aggregation sizes (see Section 5.5.3 for details).
-------------	---

Why	The integration architecture defines the technologies to integrate the services into the system.
Who	System Architect
Input	Service Interface Definitions
Output	Integration Architecture
Challenges	Choosing the appropriate middleware technology and or product.

6.5.2.2 *Define Routing Rules*

Table 25: Define Routing Rules

Task	Define Routing Rules
What	<p>This task is concerned with the definition of the routing rules used by the message router. The message router routes the messages to the appropriate service endpoint, depending on the aggregation size of the message (see Section 5.3.2 for details). This task includes</p> <ul style="list-style-type: none"> • Defining which service endpoint should be called for a given aggregation size.

Why	The routing rules define, which service endpoint should be called for a given aggregation size to facilitate optimised processing for single-event and batch processing.
Who	System Architect
Input	Integration Architecture
Output	Routing Rules Definition
Challenges	Finding the data aggregation threshold to route messages to the appropriate service endpoint.

6.5.2.3 Define Controller Architecture

Table 26: Define Controller Architecture

Task	Define Controller Architecture
What	<p>This task is concerned with the definition of the controller architecture, including</p> <ul style="list-style-type: none"> • Defining the controller type, for example <ul style="list-style-type: none"> – PID Controller – Fuzzy Controller • Defining sensors and actuators and their distribution architecture • Defining filters and additional components of the control-loop. • Depends on the <i>Control Problem</i> and the system dynamics (linear, non-linear).
Why	The <i>Controller Architecture</i> is the basis for the implementation of the feedback-control loop to control the message aggregation size at run-time
Who	<i>System Architect</i>
Input	<ul style="list-style-type: none"> • <i>Integration Architecture</i> • <i>Control Problem</i>

Output	<i>Controller Architecture</i>
Challenges	Finding the right <i>Controller Architecture</i> is an iterative process. A simple solution should be used initially, which should be refined when the system is implemented. Alternatively, a simulation can be used to evaluate the <i>Controller Architecture</i> beforehand.

6.5.2.4 Define Control Problem

Table 27: Define Control Problem

Task	Define Control Problem
What	<p>This task is concerned with the definition of the <i>Control Problem</i>, including</p> <ul style="list-style-type: none"> • Defining what properties of the system should be controlled. • In case of the Adaptive Middleware (see Chapter 5) the control problem is already defined.
Why	The <i>Control Problem</i> defines the goal of the feedback-control.
Who	<i>System Architect</i>
Output	<i>Control Problem</i>
Challenges	The <i>Control Problem</i> is not in all cases obvious and needs to be derived from the <i>Performance Requirements</i> of the system.

6.5.2.5 Define Input/Output Variables

Table 28: Define Input/Output Variables

Task	Define Input/Output Variables
-------------	-------------------------------

What	<p>This task is concerned with the definition of the input and output variables used by the controller, for example</p> <ul style="list-style-type: none"> • Number of messages in the system • Input queue length • Current end-to-end latency • Current throughput
-------------	--

Why	The <i>Input/Output Variables</i> are needed for the implementation of the controller.
Who	<i>System Architect</i>
Input	<i>Control Problem</i>
Output	<i>Input/Output Variables</i>
Challenges	The selected input variables should be measured easily and directly, without delay such as when calculating averages.

6.5.3 Implementation

This package contains specific tasks that are concerned with the implementation of an adaptive system for bulk data processing.

6.5.3.1 Feedback-Control Loop

Table 29: Implement Feedback-Control Loop

Task	Implement Feedback-Control Loop
-------------	---------------------------------

What	<p>This task is concerned with the implementation of the <i>Controller Architecture</i>, including</p> <ul style="list-style-type: none"> • Implementation of sensors • Implementation of the controller • Implementation of actuators • Implementation of additional components, such as filters • Implementation of monitoring components, such as JMX beans • Implementation of mechanisms for performing static and step tests
-------------	--

Why	The Feedback-Control Loop implements the automatic adjustment of data granularity at runtime.
Who	<i>Software Engineer</i>
Input	<i>Controller Architecture</i>
Challenges	The implementation of the feedback-control loop should provide the appropriate performance for collecting and aggregating sensor data.

6.5.3.2 Perform Static Tests

Table 30: Perform Static Tests

Task	Perform Static Tests
What	Perform static tests in order to determine the static behaviour of the system. See Section 5.7.2.1 for details.
Why	The static behaviour of the system is needed to determine the characteristics of the system.
Who	<i>System Architect</i>
Output	<i>Static Test Results</i>
Tools	<ul style="list-style-type: none"> • Tools for data processing • Tools for data visualisation

Challenges The system needs to be already implemented. Alternatively, an appropriate model of the system can be used.

6.5.3.3 Perform Step Tests

Table 31: Perform Step Tests

Task	Perform Step Tests
What	Perform step tests to determine the dynamic behaviour of the system. See Section 5.7.2.2 for details.
Why	The dynamic behaviour of the system is needed for building a model of the system and to tune the controller.
Who	<i>System Architect</i>
Output	<i>Step Test Results</i>
Tools	<ul style="list-style-type: none"> • Tools for data processing • Tools for data visualisation
Challenges	The system needs to be already implemented. Alternatively, an appropriate model of the system can be used.

6.5.3.4 Create System Model / Perform System Identification

Table 32: Create System Model / Perform System Identification

Task	Create System Model / Perform System Identification
What	Build a model of the system.
Why	The system model is used to build a simulation of the system, which can be used for implementing the controller.
Who	<i>System Architect</i>
Input	Static and dynamic behaviour of the system
Output	<i>System Model</i>
Tools	Tools for system modelling and system identification

Challenges The *Software Engineer* needs to have a profound knowledge of controller theory and system identification in order to build a relevant model of the system.

6.5.3.5 Perform Controller Tuning

Table 33: Perform Controller Tuning

Task	Perform Controller Tuning
What	<p>This task is concerned with the tuning of the implemented controller.</p> <ul style="list-style-type: none"> • The Controller Tuning can either be done using the implementation of the system or with using a model of the system, alternatively. • The specific tuning depends on the chosen <i>Controller Architecture</i>.
Why	The Controller needs to be adjusted to the system characteristics.
Who	<i>Software Engineer</i>
Input	<i>Controller Architecture</i>
Output	<i>Controller Configuration</i>
Tools	Tools for Simulation
Challenges	The software engineer needs to have a profound knowledge of controller theory and the controller architecture in order to properly tune the implemented controller.

6.5.3.6 Implement Service Interfaces

Table 34: Implement Service Interfaces

Task	Implement Service Interfaces
What	<p>This task is concerned with the implementation of the business services, including</p> <ul style="list-style-type: none"> • Implementation of batch operations • Implementation of single-event operations

Why	The services implement the business functionality of the system.
Who	<i>Software Engineer</i>
Input	<i>Service Interface Definitions</i>
Challenges	Implementing appropriate optimisations for batch and single-event processing.

6.5.3.7 *Implement Aggregation Rules*

Table 35: Implement Aggregation Rules

Task	Implement Aggregation Rules
What	<p>This task is concerned with the implementation of the message aggregation, including</p> <ul style="list-style-type: none"> • Implementation and configuration of the Aggregator component. • Implementation of the aggregation rules. <p>The <i>Aggregation Rules</i> should be configurable during run-time or configuration-time and should not be hard-coded.</p>
Why	The aggregator component is responsible for aggregating events according to the aggregation rules and is one of the main building blocks of the Adaptive Middleware (see Chapter 5).
Who	<i>Software Engineer</i>
Input	<i>Aggregation Rules</i>
Challenges	Implementation of mechanisms to dynamically load aggregation rules at run-time or configuration-time.

6.5.3.8 *Implement Routing-Rules*

Table 36: Implement Routing Rules

Task	Implement Routing Rules
-------------	-------------------------

What	<p>This task is concerned with the implementation of the message routing, including</p> <ul style="list-style-type: none"> • Implementation and configuration of the Router component. • Implementation of the routing rules. <p>The <i>Routing Rules</i> should be configurable during run-time or configuration-time and should not be hard-coded.</p>
Why	The message router routes messages to the appropriate service endpoint depending on the current aggregation size. It is one of the main building blocks of the <i>Adaptive Middleware</i> (see Section 5.3.2 for details).
Who	<i>Software Engineer</i>
Input	<i>Routing Rules</i>
Challenges	Implementation of mechanisms to dynamically load routing rules at run-time or configuration-time.

6.5.4 Test

This package contains the specific tasks that are concerned with the test of an adaptive system for bulk data processing.

6.5.4.1 Define Performance Tests

Table 37: Define Performance Tests

Task	Define Performance Tests
What	<p>This task is concerned with the definition of the performance tests, including</p> <ul style="list-style-type: none"> • Definition of load scenarios • Definition of test data • Definition of the test environment • Implementation of the workload generator • Implementation of tools and scripts for the evaluation and data visualisation

Why	The Performance Test Concept defines what should be done to test whether the system meets its performance requirements.
Who	<i>Test Engineer</i>
Output	<i>Performance Test Concept</i>
Tools	<ul style="list-style-type: none"> • Tools for data processing • Tools for data visualisation
Challenges	The performance test should include tests concerning the adaptive behaviour of the system.

6.5.4.2 Evaluate Performance Test Results

Table 38: Evaluate Performance Test Results

Task	Evaluate Performance Test Results
What	Visualise the test results using the tools/skripts implemented in the task <i>Define Performance Tests</i> .
Why	The performance test evaluation is conducted to understand the performance characteristics of the system.
Who	<ul style="list-style-type: none"> • <i>Test Engineer</i> • <i>System Engineer</i>
Input	<i>Performance Test Result</i>
Output	<i>Performance Test Evaluation</i>
Tools	<ul style="list-style-type: none"> • Tools for data processing • Tools for data visualisation

6.5.5 Operations

This package contains the specific tasks that are concerned with the operation of an adaptive system for bulk data processing.

6.5.5.1 *Setup Monitoring infrastructure*

Table 39: Setup Monitoring infrastructure

Task	Setup Monitoring infrastructure
What	Setting up the monitoring infrastructure, including <ul style="list-style-type: none"> Integrating the monitoring facilities (for example JMX Beans) of the system into the existing monitoring infrastructure.
Why	The monitoring infrastructure is needed to monitor the system at run-time. Based on the monitoring the operation engineer is able to further tune the system.
Who	<i>Operations Engineer</i>
Input	<i>System Architecture</i>
Challenges	The infrastructure needs to be adjusted to the adaptive capabilities of the system.

6.5.5.2 *Setup Test Environment*

Table 40: Setup Test Environment

Task	Setup Test Environment
What	Setup up the test environment used for the performance tests, including <ul style="list-style-type: none"> Setup / Mock external Services Setup test data Deployment of the system to the test environment
Why	The test environment is needed to perform the performance tests.
Who	<i>Operations Engineer</i>
Input	<i>Performance Test Concept</i>

Challenges

- The test environment should be comparable to the production environment to get valid test results.
- Additionally, the test data should also be comparable to production data.

6.5.5.3 Perform Performance Tests

Table 41: Perform Performance Tests

Task	Perform Performance Tests
What	Perform the tests as defined by the task <i>Define Performance Tests</i> .
Why	The performance tests are necessary to assure that the system meets the performance requirements.
Who	<i>Test Engineer</i>
Input	<i>Performance Test Concept</i>
Output	<i>Performance Test Results</i>
Challenges	To ensure the reliability of the performance test results, the tests should be run multiple times. This is often difficult with regard of the needed resources for the performance test, such as availability of external systems.

6.5.6 Project Management

This package contains specific tasks that concerned with the management of the development process an adaptive system for bulk data processing.

6.5.6.1 Define Training Concept

Table 42: Define Training Concept

Task	Define Training Concept
------	-------------------------

What	Definition of the training concept, including <ul style="list-style-type: none"> • Definition of target audience, for example <i>Operation Engineers, System Engineers</i> • Definition of training content, for example <ul style="list-style-type: none"> – Discussion of the different operation modes (batch, single event processing) – Performance characteristics (regarding latency and throughput) depend on current operation mode – Tuning options (Controller, Aggregation Rules, Routing Rules)
Why	<ul style="list-style-type: none"> • The operation engineers need to have the knowledge to operate and tune the system in production. • Additionally, the team members also need to have the knowledge to design and implement the system.
Who	<i>System Architect</i>
Input	<ul style="list-style-type: none"> • <i>Business Architecture</i> • <i>System Architecture</i> • <i>Controller Architecture</i>
Output	<i>Training Concept</i>
Challenges	The <i>Training Concept</i> should consider the respective audience and its existing knowledge.

6.5.6.2 Perform Staffing

Table 43: Staffing

Task	Perform Staffing
-------------	------------------

What	<p>This task is concerned with the staffing of the project. There are special skills needed for staffing the project, for example</p> <ul style="list-style-type: none"> • Know how about the adaptive middleware concepts as introduced in Chapter 5. • Controller design, implementation, and tuning
Why	The <i>Staffing Plan</i> is needed to get the appropriate team members with the needed skills.
Who	<i>Project Manager</i>
Output	<i>Staffing Plan</i>
Challenges	It may be hard to find the right project members with the needed skillset, since control theory is not a common skill of enterprise software developers. In this case, an appropriate training should be considered upfront.

6.6 PROCESSES

A process contains an ordered list of tasks that are concerned with the implementation of a certain feature of the software system. Processes are modeled using UML activity diagrams. The conceptual framework describes the following processes:

- Implement Integration
- Implement Aggregation
- Implement Feedback-Control

6.6.1 Implement Integration

This process describes the necessary tasks to implement the integration layer and the integrated service interfaces. It contains the following tasks, as shown in Figure 53

Figure 54 shows the UML activity diagram of the process.

6.6.2 Implement Aggregation

This process is concerned with the implementation of the message aggregation. It contains the following tasks, as shown in Figure 55

Figure 56 shows the UML activity diagram of the process.

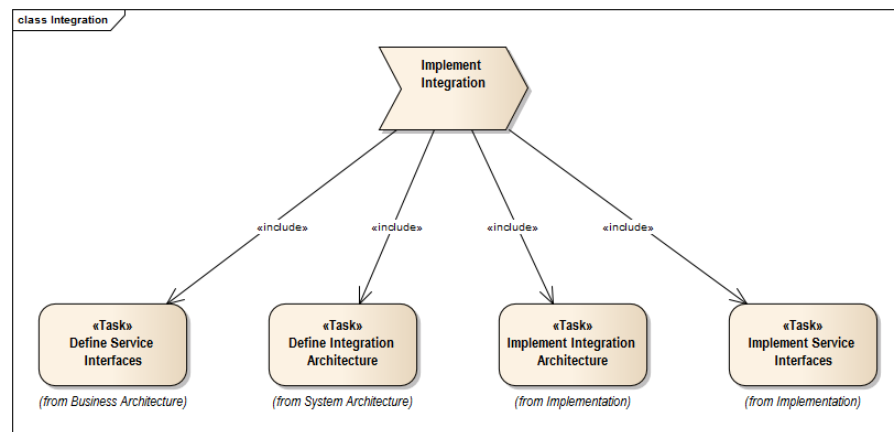


Figure 53: Tasks of the process Implement Integration

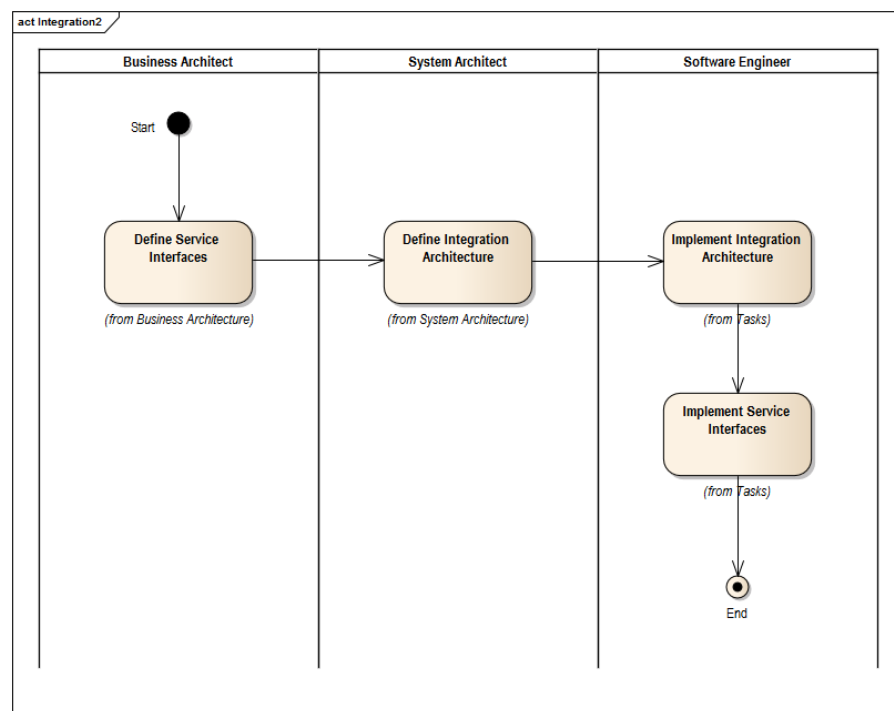


Figure 54: UML Activity Diagram: Implement Integration

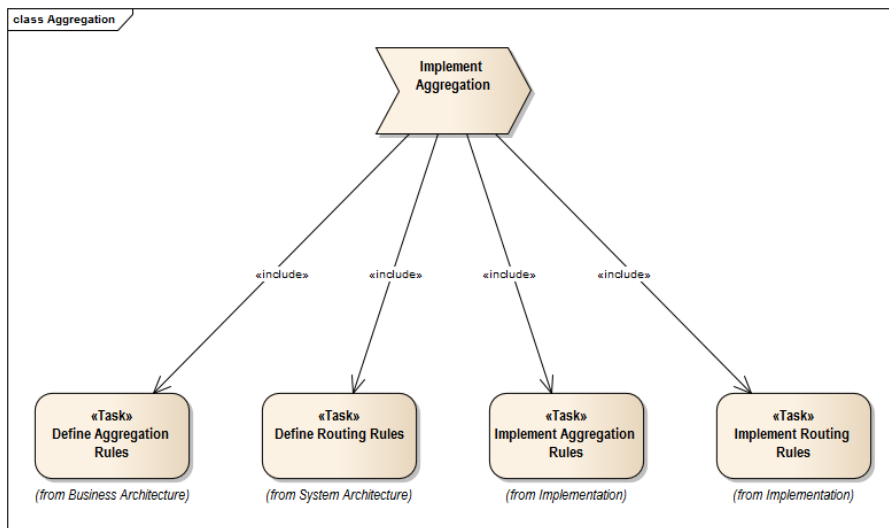


Figure 55: Tasks of the process Implement Aggregation

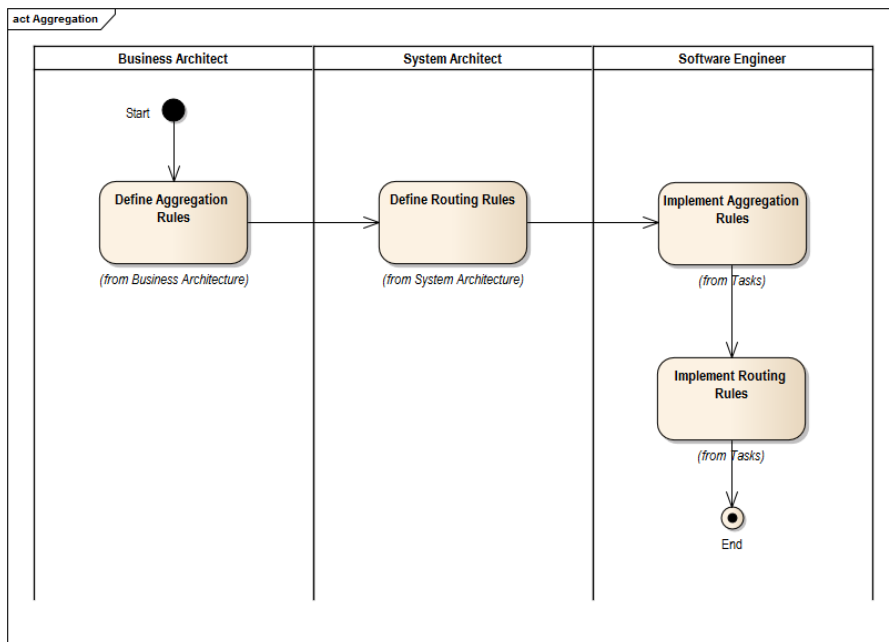


Figure 56: UML Activity Diagram: Implement Aggregation

6.6.3 Implement Feedback-Control

This process contains tasks that are concerned with the design, implementation and tuning of the feedback-control loop. It contains the following tasks, as shown in Figure 57.

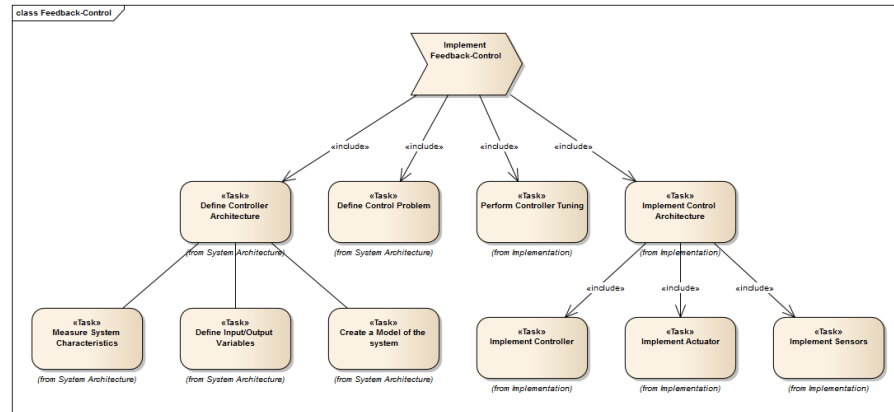


Figure 57: Tasks for implementing the feedback-control loop

There are two options for implementing the feedback-control loop:

- Using a system model for performing the controller tuning, as shown in the UML activity diagram in Figure 58a.
- Without using a model, the control architecture needs to be implemented prior to the controller tuning, as shown in the UML activity diagram in Figure 58b.

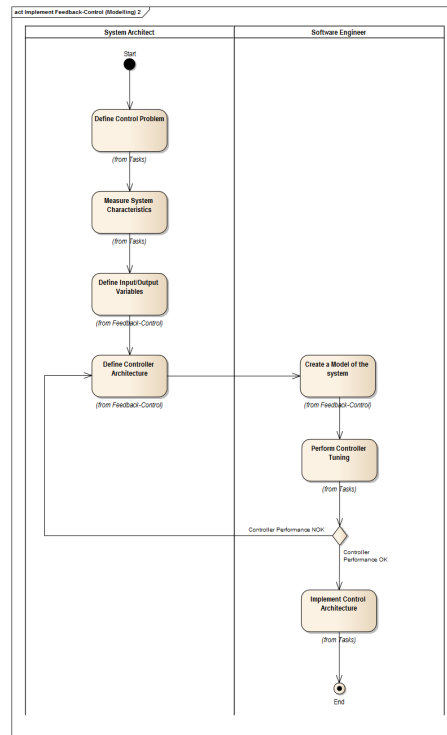
6.7 ARTIFACTS

An artifact is a result of a task. It is an intermediate result, that is needed for development of the software, but not the software product itself. Additionally, it can also be prerequisite of another task.

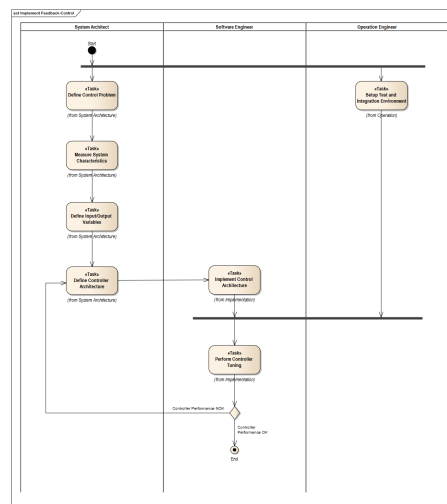
The conceptual framework only describes artifacts that are specific for the implementation of the adaptive middleware as described in Chapter 5. Artifacts that are common to every software development process are out of scope.

An artifact is described by the following attributes, as shown in Figure 50:

- **Name**
The name of the artifact.
- **Description**
A description of the artifact.
- **Task**
The task that produces the artifact.



(a) Using a model



(b) Without a model

Figure 58: UML Activity Diagram: Implement Feedback-Control Loop

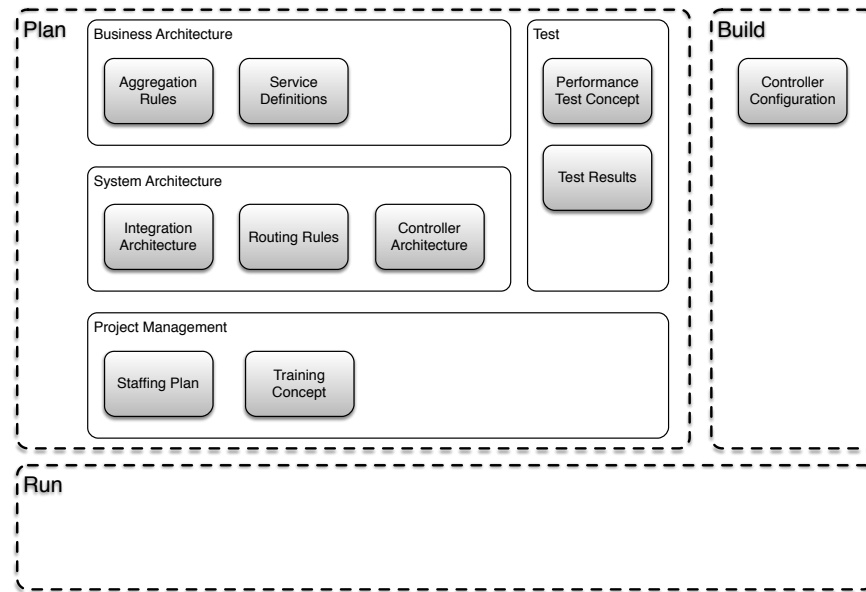


Figure 59: Artifacts

- **Role**

The role that is responsible for producing the artifact.

The conceptual framework describes the following artifacts:

- Performance Requirements
- Service Interface Definition
- Aggregation Rules
- Integration Architecture
- Routing Rules
- Controller Architecture
- System Model
- Controller Configuration
- Performance Test Concept
- Training Concept
- Staffing Plan

6.7.1 Performance Requirements

Table 44: Performance Requirements

Artifact	Performance Requirements
Description	<ul style="list-style-type: none"> • Defines the structure of input and output data • Does not include informations about the technical format, such as XML or JSON, and the integration style, such SOAP or REST
Task	Define Performance Requirements
Role	<ul style="list-style-type: none"> • Business Architect • System Architect

6.7.2 Service Interface Definition

Table 45: Service Interface Definition

Artifact	Service Interface Definition
Description	<ul style="list-style-type: none"> • Defines the structure of input and output data • Does not include informations about the technical format, such as XML or JSON, and the integration style, such SOAP or REST
Task	Define Service Interfaces
Role	Business Architect

6.7.3 Aggregation Rules

Table 46: Aggregation Rules

Artifact	Aggregation Rules
Description	Defines how events should be correlated with each other by the Aggregator.
Task	Define Aggregation Rules

Role

- Business Architect
- System Architect

6.7.4 *Integration Architecture*

Table 47: Integration Architecture

Artifact	Integration Architecture
Description	<p>Defines the technical integration of the business services, including</p> <ul style="list-style-type: none"> • Middleware technology or product • Transports, such as JMS, SOAP! or FTP • Technical format of the input and output data, such as XML or JSON, CSV or binary formats.
Task	Define Integration Architecture
Role	System Architect

6.7.5 *Routing Rules*

Table 48: Routing Rules

Artifact	Routing Rules
Description	Defines which service endpoint should be called by the Router for a given aggregation size.
Task	<i>Define Routing Rules</i>
Role	<i>System Architect</i>

6.7.6 *System Model*

Table 49: System Model

Artifact	System Model
-----------------	--------------

Description The system model is used to build a simulation of the system which can be used for implementing the controller.

Task *System Identification / Modelling*

Role *System Architect*

6.7.7 Controller Configuration

Table 50: Controller Configuration

Artifact	Controller Configuration
Description	The controller configuration specifies the parameter of the Controller.
Task	<i>Perform Controller Tuning</i>
Role	<i>Software Engineer</i>

6.7.8 Training Concept

Table 51: Training Concept

Artifact	Training Concept
Description	<ul style="list-style-type: none"> • Defines the audience of the training, for example Operations Engineers, Software Engineers or Test Engineers. • Defines the content of the training, for example basics of control theory, details about the Adaptive Middleware for Bulk Data Processing. • Defines the type of training, such as virtual training, on-site training, face-to-face training. • Defines a timeplan, learning modules and needed facilities to conduct the training.
Task	<i>Define Training Concept</i>

Role

- *Project Manager*
- *System Architect*

6.7.9 *Staffing Plan*

Table 52: Training Concept

Artifact	Staffing Plan
Description	<p>The staffing plan contains</p> <ul style="list-style-type: none"> • The required team members and their utilisation over the project time (staffing curve). • The required roles and their assignment to team members. • A skill matrix that shows the required skills and the knowlegde of each team member.
Task	<i>Perform Staffing</i>
Role	<i>Project Manager</i>

6.8 TOOLS

The design and implementation of the adaptive middleware requires the use of some specific tools. A tool is deccribed by the following attributes, as shown in Figure 61:

- **Name**
The name of the tool.
- **Description**
The description of the tool.
- **Category**
The category the tool belongs to.

Tools are grouped in the following categories:

- Tools for system modelling, system identification and simulation
- Tools for data visualisation
- Tools for data processing

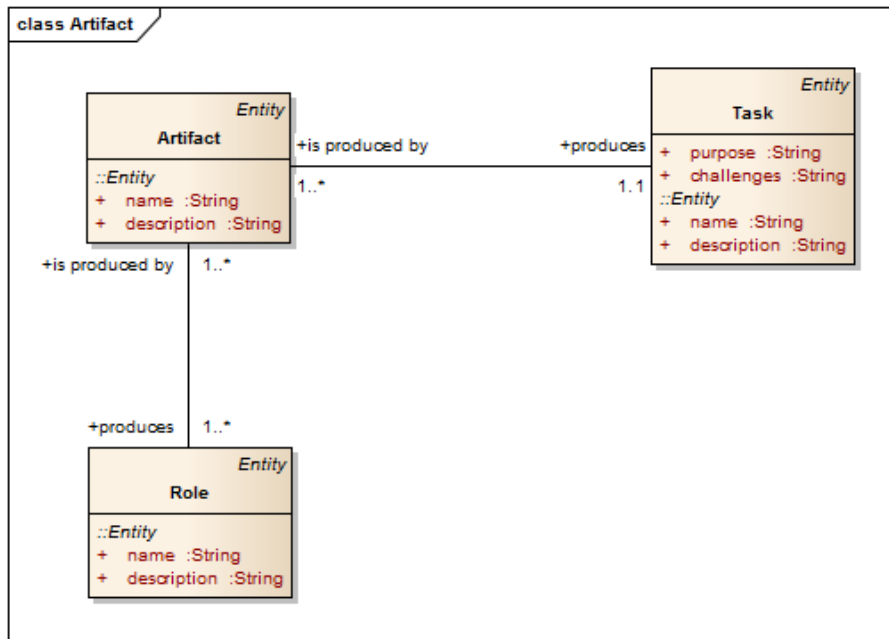


Figure 60: Attributes of an artifact

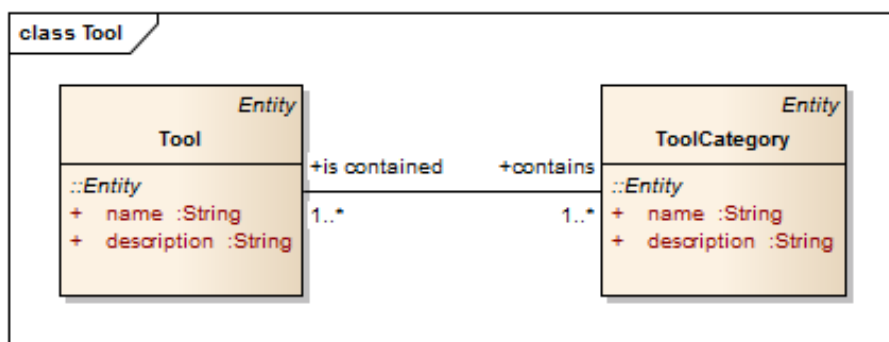


Figure 61: Attributes of a tool

6.8.1 *Tools for System Modelling, System Identification and Simulation*

The implementation of the feedback-control loop can be aided with special tools for system modelling, system identification or simulation. Examples of such tools include:

- Discrete Event Simulation Frameworks, such as SimPy (*SimPy*, 2014), SystemC (*SystemC*, 2014)
- MATLAB/Simulink (*MathWorks*, 2014)
- Scilab/Xcos (*Scilab*, 2014)

6.8.2 *Tools for Data Visualisation*

In order to gain insights from the performance test and controller tuning results, the test results should be visualized with a suitable data visualisation tool. Examples of data visualisation tools include:

- Microsoft Excel (*Microsoft Excel*, 2012)
- MATLAB (*MathWorks*, 2014)
- Gnuplot (*Gnuplot*, 2014)
- matplotlib (*matplotlib*, 2012)

6.8.3 *Tools for data processing*

For the evaluation of the performance test results, it is often necessary to process log files, which have been generated during the test runs. For example for the calculation of statistical values. While this can be done with an arbitrary programming language, the following programming or scripting languages are in particular suitable for data processing:

- Perl (*The Perl Programming Language*, 2014)
- Python (*Python*, 2013)

6.9 RELATIONSHIP TO OTHER SOFTWARE DEVELOPEMENT APPROACHES

The conceptual framework is only concerned with the special aspects of the design, implementation and operation of the adaptive middleware presented in Chapter 5. It does not describe a complete software development approach. The conceptual framework therefore needs to be integrated in common software development frameworks or methodologies.

In principle, the conceptual framework can be integrated in any iterative software lifecycle approach, such as the Rational Unified Process, the spiral model (Boehm, 1988) or agile development frameworks such as Scrum (Schwaber and Sutherland, 2013). Linear lifecycle models such as the waterfall model (Royce, 1987) are not suited because tasks like controller design, controller implementation and controller tuning need to be iterative.

This section describes briefly how the conceptual framework can be used with two common software development methodologies, the RUP and Scrum.

6.9.1 Rational Unified Process

The Rational Unified Process (RUP) is an approach to assigning activities and responsibilities within a development organization to produce high-quality software that meets the requirements of its users within a predictable schedule and budget (Rational Software, 2001).

RUP divides the software lifecycle into cycles, where each cycle is concerned with a new iteration of the software system. A cycle consists of the following phases (Kruchten and Royce, 1996):

- **Inception**
Establish the business case for the system and define the project scope.
- **Elaboration**
Analyse the the problem domain, establish an architectural foundation and develop the project plan.
- **Construction**
Develop and test the components and application features.
- **Transition**
Transition of the software to its end users.

Additionally, RUP describes nine core workflows, 6 engineering workflows and 3 supporting workflows:

- Engineering workflows
 - **Business modelling workflow**
Documentation of business processes using business use cases.
 - **Requirements workflow**
Description of *what* the system should do.
 - **Analysis & Design workflow**
Definition *how* the system will be realised in the implementation phase.

- **Implementation workflow**
Implementation, unit testing and integration of the system.
- **Test workflow**
Verification that all requirements have been correctly implemented.
- **Deployment workflow**
Production of the product release and delivering the software to its end users.
- Supporting workflows
 - **Project Management**
Management of the software development process including its risks.
 - **Configuration and Change Management**
Management of the artifacts produced by the software development process.
 - **Environment**
Provisioning the software development organisation with the software development environment.

Figure 62 shows the core workflows of the RUP and when they are conducted during the different phases.

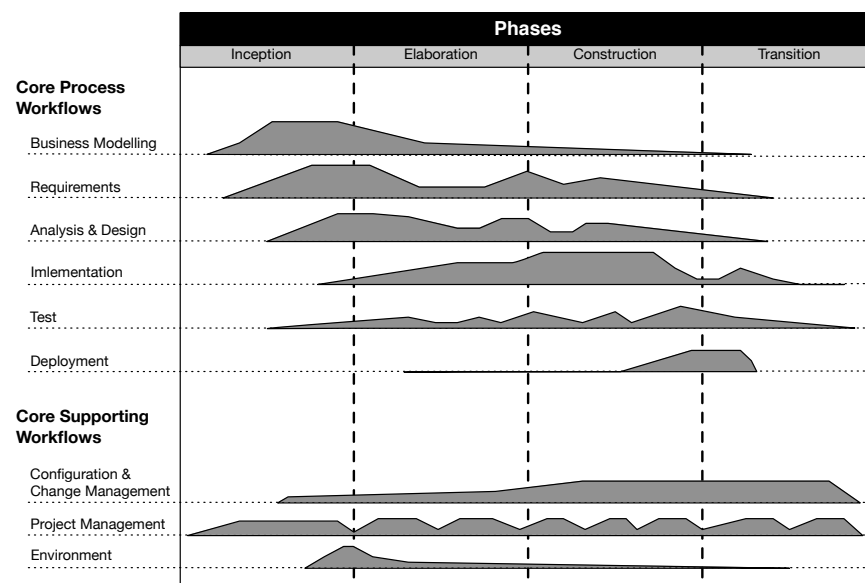


Figure 62: Core process workflows (Kruchten and Royce, 1996)

The following Table 53 shows the assignment of the tasks of the conceptual framework presented in this chapter to the core workflows of the RUP.

Table 53: Mapping of tasks to RUP core workflows

RUP core workflow	Activity
Business modelling	<ul style="list-style-type: none"> • Define Service Interfaces • Define Aggregation Rules
Requirements	<ul style="list-style-type: none"> • Define Performance Requirements
Analysis & Design	<ul style="list-style-type: none"> • Define Integration Architecture • Define Routing Rules • Define Controller Architecture
Implementation	<ul style="list-style-type: none"> • Implement Integration Architecture • Implement Service Interfaces • Implement Aggregation Rules • Implement Routing Rules • Implement Controller • Perform Controller Tuning

Test

- Define Performance Tests
- Evaluate Test Results
- Perform Performance Tests

Deployment

- Setup Monitoring Infrastructure
- Setup Test environment

Project Management

- Perform Staffing
- Define Training Concept

Configuration & Change Management

- Perform Controller Tuning

Environment

- Source Project Environments

6.9.2 *Scrum*

Scrum is a process framework that has been used to manage complex product development (Schwaber and Sutherland, 2013). It consists of Scrum Teams and their roles, artifacts, events, and rules. It is an iterative, incremental approach to optimise predictability and control risks by constantly inspecting and adapting the process.

Scrum partitions the development of software products in Sprints, a timeframe of maximum one month during which a usable and potentially releasable software product is created.

- All requirements for the software product are kept in the Product Backlog
- The Product Backlog is an ordered list, contains any changes that should be made to the software product
- Higher ordered items are more refined than lower items
- Evolves during the course of the project, items are added, refined, sorted, estimated
- Requirements are sorted according to their business values
- Managed by the Product Owner
- At the start of each sprint, the Scrum team decides which backlog items should be implemented during this sprint

A Backlog item has following properties:

- It has a description, order, estimate and value.
- It should be possible to be implemented during a single sprint.
- Contains all tasks, that are necessary to implement the described feature, such as design, coding, configuration and testing.
- Items can be grouped into epics, which represent an important theme of the software product.

Table 54 shows an example Backlog containing items for implementing a system based on the adaptive middleware. Every item contains all the necessary tasks to design, implement and test a feature. For example, the item *REQ-13* contains the tasks define controller architecture, implement control architecture and perform controller tuning.

Table 54: Example Product Backlog

ID	Priority	Description	Epic	Estimation	Status
REQ-5	1	Rating of basic events	Rating Service	15	Ready
REQ-6	2	Mediation of basic events	Mediation Service	10	Ready
REQ-11	3	Monitoring	Feedback-Control	10	Ready
REQ-10	4	Message-Aggregation	Integration Layer	8	Ready
REQ-12	5	Message-Routing	Integration Layer	8	Ready
REQ-13	6	Basic Controller	Feedback-Control	10	Ready
...

The Scrum team is self-organised and cross-functional. The team members have all the needed competencies and skill to do their work. Scrum defines the following roles:

- Product Owner
 - Responsible for maximising the value of the product and the work of the development team.
- Scrum Master
 - Ensures that everybody understands the Scrum concepts and that the process is properly enacted.
 - Coaches the Development team, removes impediments of the Development Team
- Development Team
 - There are no special roles such as system architect or test engineer.

Although Scrum does not define specific roles for the development team, the skills needed for the design and implementation of an enterprise system based on the adaptive middleware defined by the conceptual framework need to be considered when staffing the scrum team.

6.10 RELATED WORK

This section discusses work related to the conceptual framework presented in this chapter. It introduces the terms *Software Process* and *Software Process Modelling* and discusses approaches to model the general software process using UML and process models for adaptive software systems.

6.10.1 *Software Process*

“The software process is a partially ordered set of activities undertaken to manage, develop and maintain software systems.” (Acuña and Ferré, 2001a)

McChesney (1995) describes the software process as “collection of policies, procedures, and steps undertaken in the transformation of an expressed need for a software product into a software product to meet that need.”.

Another similar definition comes from Fuggetta (2000). He defines the software process as the “coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product.”

It is necessary to differentiate between the terms software process and software lifecycle. A software lifecycle describes the states through

which the software passes from the start of the development until the operation and finally the retirement (Acuña and Ferre, 2001b). Examples of software lifecycle models are the waterfall model (Royce, 1987) or the spiral model (Boehm, 1988).

6.10.2 Software Process Modelling

Software process modelling describes the creation of software development models (Acuña and Ferré, 2001a). Feiler and Humphrey (1993) describes the software process model as “an abstract representation of a process architecture, process design or process definition, where each of these describe, at various levels of detail, an organization of process elements of either a completed, current or proposed software process.”

Process models are described using Process Modelling Languages (PMLs). A PML is defined in terms of a notation, a syntax and semantics, often suitable for computational processing (Bendraou et al., 2005).

Fuggetta (2000) describes different purposes of process models:

- Process understanding
- Process design
- Training and education
- Process simulation optimization
- Process support

Typical elements of PMLs are (see for example Benali and Darniame (1992), Acuña and Ferré (2001a), Fuggetta (2000) and Curtis et al. (1992)):

- Agent or Actor
- Role
- Activity
- Artefact or Product
- Tools

Process models typically answer the following questions (Curtis et al., 1992):

- What is going to be done?
- Who is going to do it?
- When and where will it be done?

- How and why will it be done?
- Who is dependant on its being done?

Additionally, process models commonly use the following perspectives related to these questions:

- Functional: what activities are being performed
- Behavioral: In which order (when) are activities performed
- Organizational: where and by whom is an activity performed
- Informational: the entities produced by the process

McChesney (1995) provides two main categories of software process models (see also Acuña and Ferré (2001a))

- **Prescriptive**

A prescriptive software process model defines the required or recommended means of executing the software development process. It answers the question “how should the software be developed”.

- **Descriptive**

A descriptive software process model describes an existing process model. It answers the question “how has the software been developed”.

Examples of software process models include the IEEE and ISO standards IEEE 1974-1991, ISO/IEC 12207 and the Rational Unified Process (RUP).

6.10.3 Software Process Modelling using UML

UML is commonly used for modelling software processes.

UML for Software Process Modelling (UML4SPM) is an UML-based metamodel for software process modelling (Bendraou et al., 2005, 2006). It takes advantages of the expressiveness of UML 2.0 by extending a subset of its elements suitable for process modelling. UML4SPM contains two packages. The process structure package, which contains the set of primary process elements and the foundation package, which contains the subset of UML 2.0 concepts extended by this process elements to provide concepts and mechanisms for the coordination and execution of activities.

Software & System Process Modelling Metamodel (SPEM) 2.0 is a metamodel for modeling software development processes and a conceptual framework, which provides concepts for for modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes (OMG, 2008). It provides a clear separation between method content, for example deliverables and key

roles, and workflows supporting different software lifecycle models. The [SPEM](#) 2.0 metamodel consists of seven main metamodel packages, with each package extending the package it depends on:

- **Core**
Contains common classes and abstractions as the base for classes in all other packages.
- **Process Structure**
Defines the base for all process models.
- **Process Behaviour**
Extends the concepts of the process structure package with behavioural models.
- **Managed Content**
Contains concepts for managing the textual content of natural language documentation.
- **Method Content**
Provides concepts to build up development knowledge base independent of any specific processes and development projects.
- **Process With Methods**
Defines structures for integrating processes defined with concepts of process structure package with instances of concepts of the method content package.
- **Method Plugin**
Contains concepts for designing and managing maintainable, large scale, reusable, and configurable libraries or repositories of method content and processes.

Both approaches, [UML4SPM](#) and [SPEM](#) 2.0 extend the [UML](#) 2.0 notation with additional elements, which does not allow the usage of standard [UML](#) tools.

[Dietrich et al. \(2013\)](#) use [UML](#) 2.0 for modelling software processes at Siemens AG. According to the authors, the usage of standard [UML](#) 2.0 notation, which is supported by standard modelling tools, increases readability of processes for software developers since [UML](#) is also used for modelling the software itself. They describe four distinct process views:

- Process-oriented view
- Activity-oriented view
- Product-oriented view
- Role-oriented view

The following [UML](#) diagram types are used by their approach:

- Activity diagrams (process-oriented view)
- Class diagrams (activity-oriented view, product-oriented view, role-oriented view)
- Use-case diagrams (activity-oriented view, product-oriented view, role-oriented view)

The conceptual framework for feedback-controlled systems for bulk data processing presented in this chapter is based on the properties of the described approaches in this section for modelling the software development process. It uses standard [UML](#) use-case and activity diagrams for describing tasks and processes for the following reasons:

- **Understandability**
Using standard [UML](#) 2.0 notation elements and diagrams facilitate the understanding of the conceptual framework since they are commonly used by software engineers for the design of the software system itself.
- **Tool support**
Standard [UML](#) 2.0 notation elements and diagrams are supported by a wide range of modelling tools.

Standard metamodels for software process modelling such as [SPEM](#) 2.0 have not been used because they seemed to heavyweight for this purpose.

6.10.4 *Software Processes for Adaptive Software Systems*

It has been understood that software processes need to be reconceptualised to engineer self-adaptive software systems (see for example [Blair et al. \(2009\)](#), [Inverardi and Tivoli \(2008\)](#), [De Lemos et al. \(2013\)](#) or [Andersson et al. \(2013\)](#)). Self-adaptive systems adjust their behaviour automatically to respond to changes in their context and requirements. Activities that are traditionally done at development-time need to be shifted to run-time. Additionally, some activities that are previously performed by software engineers are now performed by the system itself. In a way, the role of the human software engineer is to some extent shifted from operational to strategic. The engineer implements the adaption mechanisms, the adaption itself is performed by system.

[Andersson et al. \(2013\)](#) extend the [SPEM](#) metamodel to specify which activities should be performed off-line and on-line and the dependencies between them. They distinguish between off-line activities, manual activities that are performed externally at development-time and on-line activities, that are performed internally at run-time, by the system itself, for example evolution and adaption activities performed by

the adaption logic of the system. The authors argue, that on-line activities must be explicitly reflected in software process models, since they are not independent from off-line activities. In addition to on-line activities, on-line roles and work products also need to be addressed by process models. To meet this requirements, they extend the [SPEM](#) metamodel with

- On-line and off-line stereotypes to define whether an activity should be performed on-line or off-line
- Dependencies to relate two or more arbitrary process elements
- Elements to describe the costs and benefits of performing an activity on-line in contrast to perform it off-line.

[Inverardi and Mori \(2010\)](#) describe a process methodology to support the development of context-aware adaptive applications. It consists of four different activities: *Explore*, *Integrate*, *Validate* and *Evolve*:

- **Exploration Phase**
Exploits a feature library containing the implementation and corresponding requirements description.
- **Integration phase**
Uses these features to produce a feature-diagram to describe the space of system changes, called variants.
- **Validation phase**
Validates the variants by using context analysis and model checking.
- **Evolution phase**
Reconfigures the system by switching to the new configuration.

[Gacek et al. \(2008\)](#) propose a conceptual model for self-adaptation which uses the ITIL Change Management process as a starting point. It consists of a reference process, activities, roles and responsibilities and artifacts. The reference process consists of two processes that interact iteratively, the adaption process and the evolution process:

- The inner *Adaption Process* relates to the feedback-loop of a single adaptable element of the system and is comprised of the activities *Sense*, *Trigger*, *Select Adaption Rules* and *Change*. All these activities are fully automated.
- The *Evolution Process* is executed for a single or multiple occurrences of the inner adaptive process. It consists of the activities *Aggregate Metrics*, *Analyze*, *Evolve Adaption Rules*, *Adjust* and *Synchronize*, and *Reflect*. These tasks might require human involvement.

The related work on process models for adaptive systems is focused on generic adaptation mechanisms to evolve and adapt a system, which are carried out at run-time. In contrast, the conceptual framework presented in this chapter is aimed to guide the design, development and operation of a specific system, that is, an adaptive system for bulk data processing, which provides a specific adaptation mechanism, that is, the adaption of the aggregation size at run-time depending on the current load of the system.

6.11 SUMMARY

In this chapter a conceptual framework has been presented to guide the design, implementation and operation of an enterprise system for bulk data processing that is based on the adaptive middleware as described in the previous Chapter 5.

The conceptual consists of the entities phases, roles, tasks, artifacts, processes and tools. It describes:

- The needed roles and their skills for the design, implementation and operation.
- The necessary tasks and their relationships for the design, implementation and operation.
- The artifacts that are created and required by the different tasks.
- The tools that are needed to process the different tasks.
- The processes that describe the order of tasks to implement a certain feature of the software system.

It uses standard [UML](#) notation elements, which facilitates understandability by software architects and developers. Additionally, this approach offers an extensive tool support.

The conceptual framework is only concerned with the special aspects of the design, implementation and operation of an adaptive system for bulk data processing. It does not describe a complete software development approach. The conceptual framework therefore needs to be integrated in common software development frameworks or methodologies. It has been shown how the conceptual framework can be used with two common software development methodologies, the [RUP](#) and Scrum.

It should be noted that software processes are not fixed during their lifetime, they need to be continuously improved. ([Fuggetta, 2000](#)) The conceptual model can therefore be tailored to specific projects requirements, it does not have to be followed strictly.

Part III

CONCLUSION

CONCLUSION

7.1 SUMMARY OF THE RESEARCH PROJECT

7.2 CONTRIBUTIONS

7.3 LIMITATIONS

Limitations of the concept for an *Adaptive Middleware*:

- Only a single processing pattern is implemented
- Integrated services need to be changed to support batch and single event processing
 - Integration of Off-the-shelf components difficult
- Only a single adaption pattern is considered in the adaptive middleware concept and prototype implementation.

Limitations of the research prototype:

- Impact of different controller architectures have not been exhaustively analysed and researched. Only two controller architectures have been implemented.

Limitations of the *Conceptual Framework*:

- The services integrated by the prototype do not implement any further optimisations for batch processing. They use the same implementation for batch and single-event processing. Thus, the impact of batch optimisations has not been investigated.
- The *Conceptual Framework* has not been validated, for example by qualitative research methods, such as expert interviews are applied with real-life projects.

7.4 FUTURE WORK

Based on the limitations of this research, as described in the previous section, the following open issues could be addressed by future work.

- System consisting of multiple Aggregators and input queues
 - Does this approach still work with the proposed control strategy (every aggregator optimizes the aggregation size independently)?

- Is a different approach better or even necessary? For example a central control strategy?
- Add a figure for illustration
- Support for different messaging patterns such as Publish/Subscribe
- Extending the middleware concept to support different adaptation patterns, such as dynamic service composition and selection and load balancing.
- Validation of the *Conceptual Framework* by qualitative research methods or in real-life projects

SOURCE CODE

This section describes the source code of the research prototypes, which have been implemented during the course of this research.

The complete source code is available on Github: <https://github.com/mswiente/phd>

A.1 PROJECT STRUCTURE

- BatchProcessor
- BillingRouter
- BillingService
- CamelUtils
- DataGenerator
- FunctionalCore
- MediationBatchRoute
- MediationService
- PerformanceMonitor
- RatingBatchRoute
- RatingService

A.2 PACKAGE STRUCTURE

BIBLIOGRAPHY

- Abdelzaher, T., Diao, Y., Hellerstein, J., Lu, C. and Zhu, X. (2008). Introduction to Control Theory And Its Application to Computing Systems, in Z. Liu and C. Xia (eds), *Performance Modeling and Engineering*, Springer US, pp. 185–215–215. (Cited on pages 70, 84, and 85.)
- Abdelzaher, T. F., Stankovic, J. A., Lu, C., Zhang, R. and Lu, Y. (2003). Feedback performance control in software services, *Control Systems, IEEE* 23(3): 74–90. (Cited on pages 70 and 86.)
- Abu-Ghazaleh, N. and Lewis, M. J. (2005). Differential Deserialization for Optimized SOAP Performance, *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, Washington, DC, USA, p. 21. (Cited on page 24.)
- Acuña, S. T. and Ferré, X. (2001a). Software process modelling., *ISAS-SCI (1)*, pp. 237–242. (Cited on pages 143, 144, and 145.)
- Acuña, S. T. and Ferre, X. (2001b). The software process: Modelling, evaluation and improvement, *Handbook of Software Engineering and Knowledge Engineering* 1: 193–237. (Cited on page 144.)
- Alur, D., Malks, D., Crupi, J., Booch, G. and Fowler, M. (2003). *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*, 2 edn, Sun Microsystems, Inc., Mountain View, CA, USA. (Cited on page 41.)
- Amazon EC2 Auto Scaling (n.d.). <http://aws.amazon.com/autoscaling>. [retrieved: March 2014]. (Cited on page 85.)
- Ameller, D. and Franch, X. (2008). Service Level Agreement Monitor (SALMon), *ICCBSS '08: Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, IEEE Computer Society, Washington, DC, USA, pp. 224–227. (Cited on page 29.)
- Andersson, J., Baresi, L., Bencomo, N., de Lemos, R., Gorla, A., Inverardi, P. and Vogel, T. (2013). Software engineering processes for self-adaptive systems, in R. de Lemos, H. Giese, H. Müller and M. Shaw (eds), *Software Engineering for Self-Adaptive Systems II*, Vol. 7475 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 51–75. Available from: http://dx.doi.org/10.1007/978-3-642-35813-5_3. (Cited on page 147.)

- Andersson, J., de Lemos, R., Malek, S. and Weyns, D. (2009). Reflecting on self-adaptive software systems., *SEAMS* pp. 38–47. (Cited on page 26.)
- Andresen, D., Sexton, D., Devaram, K. and Ranganath, V. (2004). LYE: a high-performance caching SOAP implementation, *Proceedings of the 2004 International Conference on Parallel Processing (ICPP-2004)*, pp. 143–150. (Cited on page 24.)
- Apache Camel* (2014). <http://camel.apache.org>. [retrieved: July 2014]. (Cited on pages 42 and 44.)
- Auto Scaling on the Google Cloud Platform* (n.d.). <https://cloud.google.com/developers/articles/auto-scaling-on-the-google-cloud-platform>. [retrieved: March 2014]. (Cited on page 85.)
- Bai, X., Xie, J., Chen, B. and Xiao, S. (2007). Dresr: Dynamic routing in enterprise service bus, *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pp. 528–531. (Cited on pages 69 and 84.)
- Balsamo, S., Di Marco, A., Inverardi, P. and Simeoni, M. (2004). Model-Based Performance Prediction in Software Development: A Survey., *IEEE Trans. Software Eng.* () 30(5): 295–310. (Cited on page 60.)
- Bauer, D., Garces-Erice, L., Rooney, S. and Scotton, P. (2008). Toward scalable real-time messaging, *IBM Systems Journal* 47(2): 237–250. (Cited on page 25.)
- Benali, K. and Derniame, J. C. (1992). Software processes modeling: What, who, and when, *Software Process Technology*, Springer Berlin Heidelberg, Berlin/Heidelberg, pp. 21–25. (Cited on page 144.)
- Bendraou, R., Gervais, M.-P. and Blanc, X. (2005). UML4SPM: A UML2.0-Based Metamodel for Software Process Modelling, *Model Driven Engineering Languages and Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 17–38. (Cited on pages 144 and 145.)
- Bendraou, R., Gervais, M.-P. and Blanc, X. (2006). Uml4spm: An executable software process modeling language providing high-level abstractions, *Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International*, pp. 297–306. (Cited on page 145.)
- Benosman, R., Albrieux, Y. and Barkaoui, K. (2012). Performance evaluation of a massively parallel esb-oriented architecture, *Service-Oriented Computing and Applications (SOCA), 2012 5th IEEE International Conference on*, pp. 1–4. (Cited on page 25.)

- Blair, G., Bencomo, N. and France, R. (2009). Models@ run.time, *Computer* **42**(10): 22–27. (Cited on page [147](#).)
- Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement., *IEEE Computer* () **21**(5): 61–72. (Cited on pages [137](#) and [144](#).)
- Brebner, P. C. (2008). Performance Modeling for Service Oriented Architectures, *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, ACM, New York, NY, USA, pp. 953–954. (Cited on page [60](#).)
- Brun, Y., Serugendo, G., Gacek, C. and Giese, H. (2009). Engineering self-adaptive systems through feedback loops, ... *Self-Adaptive Systems* . (Cited on pages [70](#) and [86](#).)
- Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R. and Tamburrelli, G. (2011). Dynamic qos management and optimization in service-based systems, *Software Engineering, IEEE Transactions on* **37**(3): 387–409. (Cited on page [85](#).)
- Castellanos, M., Casati, F., Shan, M.-C. and Dayal, U. (2005). iBOM: A Platform for Intelligent Business Operation Management, *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, IEEE Computer Society, Washington, DC, USA, pp. 1084–1095. (Cited on page [30](#).)
- Chang, S.-H., La, H. J., Bae, J. S., Jeon, W. Y. and Kim, S. D. (2007). Design of a dynamic composition handler for esb-based services, *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pp. 287–294. (Cited on page [84](#).)
- Chappell, D. (2004). *Enterprise Service Bus*, O'Reilly Media, Inc., Sebastopol, CA, USA. (Cited on pages [xi](#), [14](#), [15](#), and [84](#).)
- Chen, S. and Greenfield, P. (2004). QoS Evaluation of JMS: An Empirical Approach, *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, IEEE Computer Society, Washington, DC, USA, p. 90276.2. (Cited on pages [23](#) and [62](#).)
- Conrad, S., Hasselbring, W., Koschel, A. and Tritsch, R. (2006). *Enterprise Application Integration: Grundlagen, Konzepte, Entwurfsmuster, Praxisbeispiele*, Elsevier, Spektrum, Akad. Verl. (Cited on page [9](#).)
- Curtis, B., Kellner, M. I. and Over, J. (1992). Process modeling, *Communications of the ACM* **35**(9): 75–90. (Cited on page [144](#).)
- D'Ambrogio, A. (2005). A WSDL Extension for Performance-Enabled Description of Web Services, *Lecture notes in computer science* **3733**: 371. (Cited on page [61](#).)

- D'Ambrogio, A. and Bocciarelli, P. (2007). A Model-driven Approach to Describe and Predict the Performance of Composite Services, *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, ACM, New York, NY, USA, pp. 78–89. (Cited on page 60.)
- De Lemos, R., Giese, H., Müller, H. A. and Shaw, M. (2013). Software engineering for self-adaptive systems: A second research roadmap, *Software Engineering for . . .* . (Cited on page 147.)
- Devaram, K. and Andresen, D. (2003). SOAP optimization via parameterized client-side caching, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, pp. 785–790. (Cited on page 24.)
- Didona, D., Carnevale, D., Galeani, S. and Romano, P. (2012). An extremum seeking algorithm for message batching in total order protocols, *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on*, pp. 89–98. (Cited on page 85.)
- Dietrich, S., Killisperger, P., Stückl, T., Weber, N., Hartmann, T. and Kern, E.-M. (2013). Using uml 2.0 for modelling software processes at siemens ag, in R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry and M. Lang (eds), *Information Systems Development*, Springer New York, pp. 561–572. Available from: http://dx.doi.org/10.1007/978-1-4614-4951-5_45. (Cited on page 146.)
- Duc, B. L., Châtel, P., Rivierre, N., Malenfant, J., Collet, P. and Truck, I. (2009). Non-functional Data Collection for Adaptive Business Processes and Decision Making, *MWSOC '09: Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*, ACM, New York, NY, USA, pp. 7–12. (Cited on page 29.)
- Duran-Limon, H. A., Blair, G. S. and Coulson, G. (2004). Adaptive Resource Management in Middleware: A Survey, *IEEE Distributed Systems Online* 5(7): 1. Available from: http://portal.acm.org/ft_gateway.cfm?id=1018100&type=external&coll=ACM&dl=GUIDE&CFID=59338606&CFTOKEN=18253396. (Cited on page 27.)
- Engels, G., Hess, A., Humm, B., Juwig, O., Lohmann, M., Richter, J.-P., Voß, M. and Willkomm, J. (2008). *Quasar Enterprise - Anwendungslandschaften serviceorientiert gestalten*, dpunkt Verlag. (Cited on page 17.)
- Estrella, J. C., Santana, M. J., Santana, R. H. C. and Monaco, F. J. (2008). Real-Time Compression of SOAP Messages in a SOA Environment, *SIGDOC '08: Proceedings of the 26th annual ACM international conference on Design of communication*, ACM, New York, NY, USA, pp. 163–168. (Cited on page 24.)

- EXI Working Group [online]. 2007. Available from: <http://www.w3.org/XML/EXI> [cited January 2008]. (Cited on page 19.)
- Feiler, P. and Humphrey, W. (1993). Software process development and enactment: concepts and definitions, *Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*, pp. 28–40. (Cited on page 144.)
- Fleck, J. (1999). A distributed near real-time billing environment, *Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA '99*, pp. 142–148. (Cited on page 3.)
- Friedman, R. and Hadad, E. (2006). Adaptive batching for replicated servers, *Reliable Distributed Systems, 2006. SRDS '06. 25th IEEE Symposium on*, pp. 311–320. (Cited on page 85.)
- Friedman, R. and Renesse, R. V. (1997). Packing messages as a tool for boosting the performance of total ordering protocols, *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, HPDC '97*, IEEE Computer Society, Washington, DC, USA, pp. 233–. (Cited on page 85.)
- Fuggetta, A. (2000). Software process: a roadmap., *ICSE - Future of SE Track* pp. 25–34. (Cited on pages 89, 143, 144, and 149.)
- Gacek, C., Giese, H. and Hadar, E. (2008). Friends or foes?: a conceptual analysis of self-adaptation and its change management., *SEAMS* pp. 121–128. (Cited on page 148.)
- Garces-Erice, L. (2009). Building an enterprise service bus for real-time soa: A messaging middleware stack, *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, Vol. 2, pp. 79–84. (Cited on page 25.)
- Gmach, D., Krompass, S., Scholz, A., Wimmer, M. and Kemper, A. (2008). Adaptive Quality of Service Management for Enterprise Services, *ACM Trans. Web* 2(1): 1–46. (Cited on pages 28 and 32.)
- Gnuplot (2014). <http://gnuplot.info>. [retrieved: November 2014]. (Cited on page 136.)
- González, L. and Ruggia, R. (2011). Addressing qos issues in service based systems through an adaptive esb infrastructure, *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, MW4SOC '11*, ACM, New York, NY, USA, pp. 4:1–4:7. Available from: <http://doi.acm.org/10.1145/2093185.2093189>. (Cited on page 85.)
- Guinea, S., Baresi, L., Spanoudakis, G. and Nano, O. (2009). Comprehensive Monitoring of BPEL Processes, *IEEE Internet Computing* 99. (Cited on page 29.)

- Gullapalli, R. K., Muthusamy, C. and Babu, V. (2011). Control systems application in java based enterprise and cloud environments—a survey, *Journal of ACSA* . (Cited on page 86.)
- Habich, D., Richly, S. and Grasselt, M. (2007). Data-Grey-Box Web Services in Data-Centric Environments, *IEEE International Conference on Web Services, 2007. ICWS 2007*, pp. 976–983. (Cited on pages 24 and 31.)
- Habich, D., Richly, S., Preissler, S., Grasselt, M., Lehner, W. and Maier, A. (2007). BPEL-DT – Data-Aware Extension of BPEL to Support Data-Intensive Service Applications, *Emerging Web Services Technology* 2: 111–128. (Cited on page 25.)
- Haesen, R., Snoeck, M., Lemahieu, W. and Poelmans, S. (2008). On the definition of service granularity and its architectural impact, in Z. Bellahsene and M. Léonard (eds), *Advanced Information Systems Engineering*, Vol. 5074 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 375–389. Available from: http://dx.doi.org/10.1007/978-3-540-69534-9_29. (Cited on page 26.)
- Hellerstein, J. L. (2004). Challenges in control engineering of computing systems, *American Control Conference, 2004. Proceedings of the 2004*, pp. 1970–1979. (Cited on pages 77 and 86.)
- Hellerstein, J. L., Diao, Y., Parekh, S. and Tilbury, D. M. (2004). *Feedback Control of Computing Systems*, John Wiley & Sons. (Cited on pages 77 and 85.)
- Henjes, R., Menth, M. and Zepfel, C. (2006). Throughput Performance of Java Messaging Services Using WebsphereMQ, *ICDCSW '06: Proceedings of the 26th IEEE International Conference Workshops on Distributed Computing Systems*, IEEE Computer Society, Washington, DC, USA, p. 26. (Cited on pages 23 and 61.)
- Her, J. S., Choi, S. W., Oh, S. H. and Kim, S. D. (2007). A Framework for Measuring Performance in Service-Oriented Architecture, *NWESP '07: Proceedings of the Third International Conference on Next Generation Web Services Practices*, IEEE Computer Society, Washington, DC, USA, pp. 55–60. (Cited on pages 22 and 61.)
- Hess, A., Humm, B. and Voß, M. (2006). Regeln für serviceorientierte Architekturen hoher Qualität, *Informatik Spektrum* 29(6): 395–411. (Cited on pages 19 and 20.)
- Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. (Cited on pages xi, 9, 42, 70, and 71.)

- IBM Group (2005). *An architectural blueprint for autonomic computing*, IBM White paper. (Cited on page 26.)
- IEEE (2008). IEEE standard for a precision clock synchronization protocol for networked measurement and control systems, *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* pp. c1–269. (Cited on page 50.)
- Inverardi, P. and Mori, M. (2010). A Software Lifecycle Process to Support Consistent Evolutions., *Software Engineering for Self-Adaptive Systems* 7475(Chapter 10): 239–264. (Cited on page 148.)
- Inverardi, P. and Tivoli, M. (2008). The Future of Software: Adaptation and Dependability., *ISSSE* 5413(Chapter 1): 1–31. (Cited on page 147.)
- Irmert, F., Fischer, T. and Meyer-Wegener, K. (2008). Runtime Adaptation in a Service-Oriented Component Model, *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, ACM, New York, NY, USA, pp. 97–104. (Cited on pages 28 and 32.)
- Jongtaveesataporn, A. and Takada, S. (2010). Enhancing enterprise service bus capability for load balancing, *W. Trans. on Comp.* 9(3): 299–308. Available from: <http://dl.acm.org/citation.cfm?id=1852392.1852401>. (Cited on page 84.)
- Josuttis, N. (2007). *SOA in practice*, O'Reilly, Sebastopol, CA, USA. (Cited on pages 16 and 19.)
- Kon, F., Costa, F., Blair, G. and Campbell, R. H. (2002). The Case for Reflective Middleware, *Commun. ACM* 45(6): 33–38. (Cited on page 28.)
- Krafzig, D., Banke, K. and Slama, D. (2005). *Enterprise SOA*, Prentice Hall. (Cited on pages 16 and 17.)
- Kramer, J. and Magee, J. (2007). Self-Managed Systems: an Architectural Challenge., *FOSE* pp. 259–268. (Cited on page 26.)
- Kruchten, P. and Royce, W. (1996). A rational development process, *CrossTalk* 9(7): 11–16. (Cited on pages xi, 137, and 138.)
- Kumar, G. R., Muthusamy, C. and Vinaya Babu, A. (2013). Intelligent Enterprise Application Servers: A Vision for Self-Managing Performance, in V. V. Das (ed.), *Lecture Notes in Electrical Engineering*, Springer New York, pp. 577–584–584. (Cited on page 86.)
- Kumar, R., Muthusamy, C. and Babu, A. V. (2012). Self-regulating Message Throughput in Enterprise Messaging Servers—A Feedback Control Solution, *Self* 3(1). (Cited on page 86.)

- Laddaga, R. and Robertson, P. (2008). Abstract Self Adaptive Software: A Position Paper. (Cited on page 27.)
- Leclercq, M., Quéma, V. and Stefani, J.-B. (2004). DREAM: a Component Framework for the Construction of Resource-Aware, Reconfigurable MOMs, *ARM '04: Proceedings of the 3rd workshop on Adaptive and reflective middleware*, ACM, New York, NY, USA, pp. 250–255. (Cited on page 28.)
- Lee, B.-D., Weissman, J. B. and Nam, Y.-K. (2009). Adaptive middleware supporting scalable performance for high-end network services, *J. Netw. Comput. Appl.* 32(3): 510–524. (Cited on page 28.)
- Liu, Y. and Gorton, I. (2005). Performance prediction of j2ee applications using messaging protocols, in G. Heineman, I. Crnkovic, H. Schmidt, J. Stafford, C. Szyperski and K. Wallnau (eds), *Component-Based Software Engineering*, Vol. 3489 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–16. Available from: http://dx.doi.org/10.1007/11424529_1. (Cited on page 60.)
- Liu, Y., Gorton, I. and Zhu, L. (2007). Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus, *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference - Vol. 1- (COMPSAC 2007)*, IEEE Computer Society, Washington, DC, USA, pp. 327–334. (Cited on page 60.)
- MathWorks (2014). MATLAB, <http://www.mathworks.com>. [retrieved: November 2014]. (Cited on page 136.)
- matplotlib (2012). <http://matplotlib.org>. [retrieved: November 2014]. (Cited on page 136.)
- McChesney, I. (1995). Toward a classification scheme for software process modelling approaches, *Information and Software Technology* 37(7): 363 – 374. Available from: <http://www.sciencedirect.com/science/article/pii/095058499591492I>. (Cited on pages 143 and 145.)
- Menth, M., Henjes, R., Zepfel, C. and Gehrsitz, S. (2006a). Throughput Performance of Popular JMS Servers, *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, ACM, New York, NY, USA, pp. 367–368. (Cited on page 23.)
- Menth, M., Henjes, R., Zepfel, C. and Gehrsitz, S. (2006b). Throughput Performance of Popular JMS Servers, *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, ACM, New York, NY, USA, pp. 367–368. (Cited on page 61.)

- Microsoft Excel (2012). <http://products.office.com/en-us/excel>. [retrieved: November 2014]. (Cited on page 136.)
- Ng, A. (2006). Optimising Web Services Performance with Table Driven XML, *ASWEC '06: Proceedings of the Australian Software Engineering Conference*, IEEE Computer Society, Washington, DC, USA, pp. 100–112. (Cited on page 24.)
- Ng, A., Greenfield, P. and Chen, S. (2005). A Study of the Impact of Compression and Binary Encoding on SOAP Performance, *Proceedings of the Sixth Australasian Workshop on Software and System Architectures (AWSA2005)*. (Cited on page 24.)
- O'Brien, L., Brebner, P. and Gray, J. (2008). Business Transformation to SOA: Aspects of the Migration and Performance and QoS issues, *SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments*, ACM, New York, NY, USA, pp. 35–40. (Cited on page 22.)
- O'Brien, L., Merson, P. and Bass, L. (2007). Quality Attributes for Service-Oriented Architectures, *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, IEEE Computer Society, Washington, DC, USA, p. 3. (Cited on pages 16 and 21.)
- OMG (2008). Software Process Engineering Metamodel SPEM 2.0, *Technical Report ptc/08-04-01*, Object Management Group. (Cited on page 145.)
- Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S. and Wolf, A. L. (1999). An Architecture-Based Approach to Self-Adaptive Software, *IEEE Intelligent Systems* 14(3): 54–62. (Cited on page 27.)
- Patikirikorala, T., Colman, A., Han, J. and Wang, L. (2012). A systematic survey on the design of self-adaptive software systems using control engineering approaches, *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on* pp. 33–42. (Cited on pages 70 and 86.)
- PTP daemon (PTPd) (2013). <http://ptpd.sourceforge.net>. [retrieved: July 2014]. (Cited on page 50.)
- Python (2013). <https://www.python.org/>. [retrieved: November 2014]. (Cited on page 136.)
- Rational Software (2001). Rational Unified Process. [retrieved: November 2014]. Available from: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf. (Cited on page 137.)

- Richter, J.-P., Haller, H. and Schrey, P. (2005). Serviceorientierte Architektur, *Informatik Spektrum* **28**(5): 413–416. (Cited on page 14.)
- Romano, P. and Leonetti, M. (2012). Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning, *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pp. 786–792. (Cited on page 85.)
- Royce, W. W. (1987). Managing the Development of Large Software Systems: Concepts and Techniques., *ICSE* pp. 328–339. (Cited on pages 137 and 144.)
- Sachs, K., Kounev, S., Bacon, J. and Buchmann, A. (2009). Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark, *Perform. Eval.* **66**(8): 410–434. (Cited on pages 23 and 62.)
- Sachs, K., Kounev, S., Carter, M. and Buchmann, A. (2007). Designing a Workload Scenario for Benchmarking Message-Oriented Middleware, *Proceedings of the 2007 SPEC Benchmark Workshop, SPEC*. (Cited on pages 23 and 62.)
- Salehie, M. and Tahvildari, L. (2009). Self-Adaptive Software: Landscape and Research Challenges, *ACM Trans. Auton. Adapt. Syst.* **4**(2): 1–42. (Cited on page 27.)
- Schulte, R. (2002). Predicts 2003: Enterprise Service Buses Emerge, Gartner. (Cited on page 14.)
- Schwaber, K. and Sutherland, J. (2013). The Scrum Guide, <http://www.scrumguides.org/scrum-guide.html>. [retrieved: November 2014]. (Cited on pages 137 and 140.)
- Scilab (2014). <http://www.scilab.org>. [retrieved: November 2014]. (Cited on page 136.)
- SimPy (2014). <https://pypi.python.org/pypi/simpy>. [retrieved: November 2014]. (Cited on page 136.)
- Smith, C. U. (1990). *Performance Engineering of Software Systems*, 1st edn, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. (Cited on page 30.)
- Smith, C. U. and Williams, L. G. (2003). Best practices for software performance engineering, *Int. CMG Conference*, pp. 83–92. (Cited on page 31.)
- SOAP Specification [online]. 2007. Available from: <http://www.w3.org/TR/soap> [cited January 2008]. (Cited on page 16.)
- Spring Batch (2013). <http://static.springsource.org/spring-batch/>. [retrieved: July 2014]. (Cited on page 41.)

- Suzumura, T., Takase, T. and Tatsubori, M. (2005). Optimizing Web Services Performance by Differential Deserialization, *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, IEEE Computer Society, Washington, DC, USA, pp. 185–192. (Cited on page 24.)
- SystemC (2014). <http://www.accellera.org/downloads/standards/systemc>. [retrieved: November 2014]. (Cited on page 136.)
- Tekli, J., Damiani, E., Chbeir, R. and Gianini, G. (2012). Soap processing performance and enhancement, *Services Computing, IEEE Transactions on* 5(3): 387–403. (Cited on page 24.)
- Textor, A., Schmid, M., Schaefer, J. and Kroeger, R. (2009). SOA Monitoring Based on a Formal Workflow Model with Constraints, *QUA-SOSS '09: Proceedings of the 1st international workshop on Quality of service-oriented software systems*, ACM, New York, NY, USA, pp. 47–54. (Cited on page 30.)
- The Perl Programming Language (2014). <http://www.perl.org>. [retrieved: November 2014]]. (Cited on page 136.)
- Ueno, K. and Tatsubori, M. (2006). Early capacity testing of an enterprise service bus, *Web Services, 2006. ICWS '06. International Conference on*, pp. 709–716. (Cited on page 63.)
- Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S. and Leymann, F. (2009). Monitoring and Analyzing Influential Factors of Business Process Performance, *EDOC '09: Proceedings of the 2009 IEEE International Enterprise Distributed Object Computing Conference (edoc 2009)*, IEEE Computer Society, Washington, DC, USA, pp. 141–150. (Cited on page 30.)
- Wichaiwong, T. and Jaruskulchai, C. (2007). A Simple Approach to Optimize Web Services' Performance, *NWESP '07: Proceedings of the Third International Conference on Next Generation Web Services Practices*, IEEE Computer Society, Washington, DC, USA, pp. 43–48. (Cited on pages 24 and 31.)
- Woodall, P., Brereton, P. and Budgen, D. (2007). Investigating service-oriented system performance: a systematic study, *Softw. Pract. Exper.* 37(2): 177–191. (Cited on page 22.)
- Woodside, M., Franks, G. and Petriu, D. (2007). The future of software performance engineering, *Future of Software Engineering, 2007. FOSE '07*, pp. 171–187. (Cited on page 31.)
- Wu, B., Liu, S. and Wu, L. (2008). Dynamic reliable service routing in enterprise service bus, *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pp. 349–354. (Cited on pages 69 and 84.)

- Xia, C. and Song, S. (2011). Research on real-time esb and its application in regional medical information exchange platform, *Biomedical Engineering and Informatics (BMEI)*, 2011 4th International Conference on, Vol. 4, pp. 1933–1937. (Cited on page 25.)
- Zhang, R., Lu, C., Abdelzaher, T. F. and Stankovic, J. A. (2002). ControlWare: a middleware architecture for feedback control of software performance, *Distributed Computing Systems*, 2002. Proceedings. 22nd International Conference on, IEEE Comput. Soc, pp. 301–310. (Cited on page 86.)
- Zhuang, Z. and Chen, Y.-M. (2012). Optimizing jms performance for cloud-based application servers, *Cloud Computing (CLOUD)*, 2012 IEEE 5th International Conference on, pp. 828–835. (Cited on page 25.)
- Ziyaeva, G., Choi, E. and Min, D. (2008). Content-based intelligent routing and message processing in enterprise service bus, *Convergence and Hybrid Information Technology*, 2008. ICHIT '08. International Conference on, pp. 245–249. (Cited on pages 69 and 84.)

PUBLICATIONS

- Swientek, M., Bleimann, U. and Dowland, P. (2008). Service-Oriented Architecture: Performance Issues and Approaches, in P. Dowland and S. Furnell (eds), *Proceedings of the Seventh International Network Conference (INC2008)*, University of Plymouth, Plymouth, UK, pp. 261–269.
- Swientek, M., Humm, B., Bleimann, U. and Dowland, P. (2009). An SOA Middleware for High-Performance Communication, in U. Bleimann, P. Dowland, S. Furnell and V. Grout (eds), *Proceedings of the Fifth Collaborative Research Symposium on Security, E-learning, Internet and Networking (SEIN 2009)*, University of Plymouth, Plymouth, UK.
- Swientek, M., Humm, B., Bleimann, U. and Dowland, P. (2014). An Adaptive Middleware for Near-Time Processing of Bulk Data, *ADAPTIVE 2014, The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications*, Venice, Italy, p. 37 to 41.