MARTIN SWIENTEK

HIGH-PERFORMANCE NEAR-TIME PROCESSING OF BULK DATA

Doctor of Philosophy, January 2015

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 1.0

ABSTRACT

HIGH-PERFORMANCE NEAR-TIME PROCESSING OF BULK DATA

MARTIN SWIENTEK

Enterprise Systems like customer-billing systems or financial transaction systems are required to process large volumes of data in a fixed period of time. Those systems are increasingly required to also provide near-time processing of data to support new service offerings. Common systems for data processing are either optimized for high maximum throughput or low latency.

This thesis proposes the concept for an adaptive middleware, which is a new approach for designing systems for bulk data processing. The adaptive middleware is able to adapt its processing type fluently between batch processing and single-event processing. By using message aggregation, message routing and a closed feedback-loop to adjust the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios.

The relationship of end-to-end latency and throughput of batch and message-based systems is formally analyzed and a performance evaluation of both processing types has been conducted. Additionally, the impact of message aggregation on throughput and latency is investigated.

The proposed middleware concept has been implemented with a research prototype and has been evaluated. The results of the evaluation show that the concept is viable and is able to optimize the end-to-end latency of a system.

The design, implementation and operation of an adaptive system for bulk data processing differs from common approaches to implement enterprise systems. A conceptual framework has been development to guide the development process of how to build an adaptive software for bulk data processing. It defines the needed roles and their skills, the necessary tasks and their relationship, artifacts that are created and required by different tasks, the tools that are needed to process the tasks and the processes, which describe the order of tasks.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

API	Application Programming Interface
CDR	Call Detail Records
CPU	Central Processing Unit
CSV	Comma Separated Values
DB	Database
EIP	Enterprise Integration Pattern
ESB	Enterprise Service Bus
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
NFS	Network File System
NTP	Network Time Protocol
ORM	Object-relational mapping
PaaS	Platform as a Service
PID	Proportional Integral Derivative
PTP	Precision Time Protocol
PML	Process Modelling Language
QoS	Quality of Service
REST	Representional State Transfer
RUP	Rational Unified Process
SASO	Stable, Accurate, Settling Times, Overshoot
SCP	Secure Copy
SLA	Service Level Agreements
SOA	Service Oriented Architecture
SPEM	Software & System Process Modelling Metamodel
SQL	Structured Query Language
UML	Unified Modeling Language
UML4SPM	UML for Software Process Modelling
XML	Extended Markup Language

ACKNOWLEDGMENTS

This work presented in this thesis is the result of the last seven years. I wish to thank all the people who have supported and helped me during this long period of time. Without them this work would not have been finished sucessfully.

I would like to thank Prof. Dr. Bernhard Humm for his support, his brilliant ideas that pushed the thesis forward, his patience, when the progress has been slow and his advices.

I would also like to thank Prof. Dr. Udo Bleimann, for his advices regarding the research process in general and his support.

Additionally, I would like to thank Prof. Paul S. Dowland for his help to improve my academic writing skills and for his advice regarding the graduate school formalities.

I would sincerly like to thank my girlfriend Nadine for her endless support, love and for keeping things running. She has always believed in me und supported me when times were tough. Without her, this work would not have been possible.

Special thanks to Pumi and Mausi, my two cats, for always reminding me of the really important things in live: eating, napping, playing and having a clean litter box.

Last but not least, I would like to thank my family for their love and their support.

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

1 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.

1.1 SYSTEMS FOR BULK DATA PROCESSING

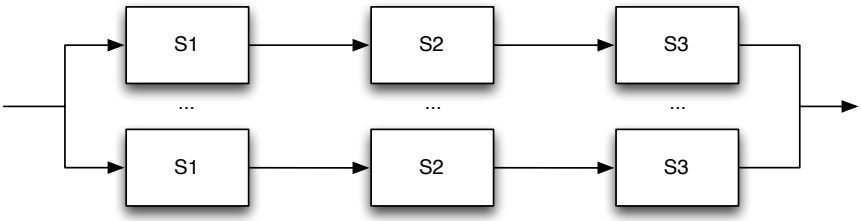
A distributed system for bulk data processing considered in this research consists of several subsystems running on different nodes that together form a processing chain, that is, the output of subsystem S1 is the input of the next subsystem S2 and so on (see Figure 1a).

To facilitate parallel processing, the system can consist of several lines of subsystems with data beeing distributed among each line.

1.1.1 *An Example: Billing Systems for Telecommunications Carriers*

An example of a system for bulk data processing is a billing system of a telecommunications carrier. 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 2).

(a) Single processing line



(b) Parallel processing lines

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

The performance requirements of such a billing system are high. It has to process more than 1 million records per hour and the whole batch run needs to be finished in a limited timeframe to comply with service level agreements with the print service provider. Since delayed invoicing causes direct loss of cash, it has to be ensured that the bill arrives at the customer on time.

Figure 2: Billing process

1.1.2 *Near-Time Processing of Bulk Data*

A new requirement for systems for bulk data processing is near-time processing. Near-time processing aims to reduce the end-to-end latency of a business process, that is, the time that is spent between the occurrence of an event and the end of its processing. In case of a billing system, it is the time between the user making a call and the complete processing of this call including mediation, rating, billing and presentment.

The need for near-time charging and billing for telecommunications carriers is induced by market forces, such as the increased advent of mobile data usage and real-time data services (Cryderman, 2011). Carriers want to offer new products and services that require real-time or near-time charging and billing. Customers want more transparency, for example, to set their own limits and alerts for their

data usage, which is currently only possible for pre-paid accounts. Currently, a common approach for carriers is to operate different platforms for real-time billing of pre-paid accounts and traditional batch-oriented billing for post-paid accounts. To reduce costs, carriers aim to converge these different plattforms.

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. A system is therefore either optimized for low latency or high maximum throughput. 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.

1.2 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:** What is an appropriate middleware design that is able to miminize the end-to-end latency for different load scenarios?

- **RQ4:** What software processes are needed to design, implement and operate an adaptive system for near-time processing of bulk data?

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

A. Performance evaluation of batch and messaging systems regarding maximum throughput and end-to-end latency.

B. Development of a concept for an adaptive middleware that delivers low latency while providing high throughput.

C. Implementation of a research prototype used for demonstrating the practicability of the concept.

D. Specification and conduction of appropriate performance tests to evaluate the developed approach.

E. 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.

1.3 RESEARCH METHODOLOGY

An evaluation of current approaches to optimize the performance of service-oriented systems has been conducted to get an understanding of the field and the involved problems.

To better understand the performance characteristics of batch and message-based single-event processing two prototypes of each processing style 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 end-to-end latency and maximum throughput. The message-based prototype has been extended to study the impact of data granularity on end-to-end latency and maximum throughput.

Based on the results of the performance evaluation of the batch and message-based prototype, the concept of an adaptive middleware has been developed. The message-based prototype has been extended to implement the concepts of the adaptive middleware. Using this prototype, a performance evaluation has been conducted to evaluate the proposed concepts for an adaptive middleware for bulk data processing.

1.4 CONTRIBUTIONS

This thesis makes the following contributions to the field:

- **A Performance evaluation of batch and messaging systems regarding maximum throughput and end-to-end latency**

  The performance evaluation compares the performance of batch and message-based systems. It analyses the impact of different processing styles, that is batch and message-based processing, on throughput and latency. It shows 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.

- **A concept and prototype implementation of a feedback-controlled adaptive middleware for near-time processing of bulk data**

  The adaptive middleware is able to adapt its processing type fluently between batch processing and single-event processing. By using message aggregation, message routing and a closed feedback-loop to adjust the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios.

- **A Conceptual Framework to Guide the Development of Feedback-Controlled Bulk Data Processing Systems**

  The design, implementation and operation of an adaptive system for bulk data processing differs from common approaches to implement enterprise systems. The conceptual framework describes the development process of how to build an adaptive software for bulk data processing. It defines the needed roles and their skills, the necessary tasks and their relationship, artifacts that are created and required by different tasks, the tools that are needed to process the tasks and the processes, which describe the order of tasks.

1.5 OUTLINE OF THE THESIS

The thesis is organized as follows:

1. **Introduction**

   This chapter sets the context for the conducted research. It specifies the aims and objectives and methodology of this research, followed by a brief description of the contributions.

2. **Background**

   This chapter further defines the context of this research and introduces core concepts and terms that are used throughout the thesis. It starts with an introduction to batch and message-based processing paradigms and analyzes the relationship between maximum throughput and end-to-end latency. Additionally, it describes performance issues of an Service Oriented Architecture (SOA) middleware and discusses current approaches for performance optimizations of such a middleware.

3. **Related Work**

   This chapter describes related work and demarcates it to the work of the research presented in this thesis. It starts with a summary of work related to the performance of service-oriented systems and approaches for performance optimizations. The chapter introduces the concept of self-adaptive software and presents related work in the context of self-adaptive middleware.

4. **Performance Evaluation of Batch and Message-based Systems**

   This chapter compares the performance of a batch and message-based system. It introduces the batch and message-based prototype systems that have been imple- mented and describes the performance evaluation to compare the performance characterics of the two different processing types. The impact of data granularity on throughput and latency of the messaging prototype is evaluated and discussed. In addition, the chapter gives an overview of other work related to the content of this chapter.

5. **An Adaptive Middleware for Near-Time Processing of Bulk Data**

   This chapter presents the design, implementation and evaluation of the adaptive middleware which is able to adapt its processing type fluently between batch processing and single-event processing. It starts with a description of the core concepts, such as message aggregation, message routing, and monitoring and control, followed by a discussion of important design aspects such as service design, transports, error handling and controller design. The design and implementation of a prototype of the adaptive middleware is outlined. A performance evaluation of the prototype is described. It is shown that the concept of an adaptive middleware for bulk data processing is able to minimize the end-to-end latency of a system. Additionally, the chap-

ter gives an overview of other work related to feedback-control of computing systems.

6. **A Conceptual Framework to Guide the Development of Feedback-Controlled Bulk Data Processing Systems**

   This chapter presents a framework that guides the practitioner to design, implement and operate a system based on the adaptive middleware for bulk data processing. It starts with a description of the metamodel of the framework, followed by a description of the entities of the process model, such as phases, roles, tasks, artifacts and tools. It is discussed how the conceptual framework can be used with other software development methodologies. The chapter gives an overview of other work related to the content of this chapter.

7. **Conclusion**

   This chapter concludes the thesis with a summary of the key objectives and contributions. It discusses the limitations of this research and identifies future work.

2

BACKGROUND

This chapter further defines the context of the conducted research and introduces core concepts and terms that are used throughout the thesis. It starts with an introduction of batch and message-based processing. The terms latency and throughput are defined and their relationship is analyzed. The term SOA is a briefly introduced, followed by a short description of Enterprise Service Bus (ESB) middleware technology. Additionally, it describes EIPs, that can be used to improve the performance of messaging systems. The chapter describes performance issues of an SOA middleware and discusses common approaches for performance optimizations of such a middleware.

2.1 BATCH PROCESSING

The traditional operation paradigm of a system for bulk data processing is batch processing (see Figure 3). 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 application reads a record from the input file, processes it and writes the record to the output file.

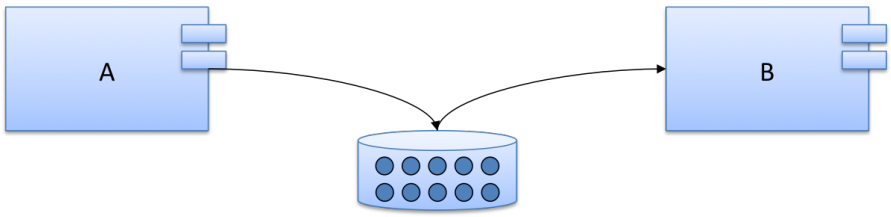


Figure 3: Batch processing

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. This timeframe is commonly called the batch window.

- **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.

2.1.1 *Integration Styles*

Batch processing systems use different styles to integrate with their outside environment or the integration of their subcomponents, with file-based and database-based integration being the most common. A combination of both styles is also possible.

- **File transfer**
  With a file-based integration, the batch system or its subcomponents read the data from the input file, processes it and writes the output to the output file. The file is transported to the next (sub-) system using File Transfer Protocol (FTP), Secure Copy (SCP) or other protocols for file transfer. The transfer is usually started by a specific job. Alternatively, a shared filesystem, such as Network File System (NFS) can be used. The system also needs to be notified when new input data is ready for processing, for example by actively monitoring a certain folder or by getting notified from the previous system in the processing chain.

- **Shared database**
  The batch system or its subcomponents read and write the input and output data to a database, which is shared among all (sub-) systems.

2.1.2 *Batch Performance Optimisations*

A batch-oriented system can be highly optimized for high throughput:

- **Data formats**
  Using optimized data formats such as binary formats or Comma Separated Values (CSV) data formats, that are optimized for reading and writing. Additionally, input and output data can be compressed to reduce data transfer times.
- **Database transactions**
  Database transactions introduce a major performance cost when processing large volumes of data. Batch processing system therefore minimize the amount of transactions by encapsulating a whole batch in one transaction, not every single record.
- **Database design and technologies**
  The database access can optimized by using pre-computed views especially designed for batch processing, such flattenend relations. Additionally, other database concepts than relational databases like No-SQL or in-memory databases can be used to further optimize the persistence layer.
- **Optimization depending on data semantics or technical properties**
  When data is processed in batches and is sorted accordingly to some business or technical rules, the processing algorithm can easily make assumptions about the data and optimize its processing. For example, when a batch only contains flat rate Call Detail Records (CDR) or CDR of a specific product or customer segment, the corresponding reference data can be pre-loaded.
- **Caching**
  Often used data such as reference data, for example products and tariffs in case of a billing system, can also be cached in memory, prior to processing.

2.1.3 *Demarcation to Big Data*

Big Data is a term that generally describes large amounts of data, that is too big to process with traditional data processing applications, such as Structured Query Language (SQL)-based databases. In the context of *Big Data*, a data processing application is defined as a system, that "answers questions based on information that was acquired in the past" (cf. Merz and Warren (2014)). These systems are commonly used in the realm of analytics to get better insights on the business.

Technologies for implementing such as a system, like Apache Hadoop (*Apache Hadoop*, 2014) are not considered for near-time processing of data, due to their high latency (cf. Merz and Warren (2014)).

2.2 MESSAGE-BASED PROCESSING

Messaging facilitates the integration of heterogeneous applications using asynchronous communication. Applications are communicating with each other by sending messages (see Figure 4). A messaging server or message-oriented middleware handles the asynchronous exchange of messages including an appropriate transaction control (Conrad et al., 2006).

In the context of this thesis, messaging is a mean to implement single-event processing.

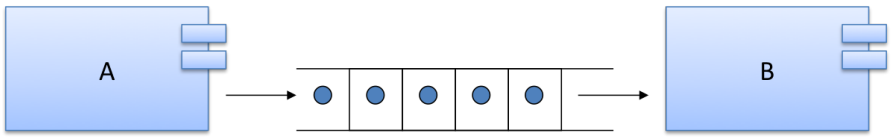


Figure 4: Message-based processing

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.

2.2.1 *Messaging Concepts*

There are two types of message channels (cf. Hohpe and Woolf (2003)):

- **Point To Point**
  A *Point To Point* channel is used to send messages to only one receiver. The messaging system ensures that a message is consumed only once. A *Point To Point* can also have multiple competing consumers, in this way, messages can be load-balanced among multiple consumers to scale the processing system.

- **Publish-Subscribe**
  A *Publish-Subscribe* channel is used to broadcast a message to multiple receivers. When a message is sent to input channel of the *Publish-Subsriber* channel, the messaging system copies the message to multiple output channels, one chanel for each receiver. Each subscriber gets the message only once.

The adaptive middleware presented in this thesis only uses *Point To Point* message channels with competing consumers.

Additionally, there are two important concepts for the transmission of messages (cf. Hohpe and Woolf (2003)):

- **Send and forget**
  The sending system sends the message to the message channel. The messaging system transmits the message in the background, the sender does not have to wait until the receiving system reads the message.

- **Store and forward**
  When the sending system sends the message to the message channel, the messaging system stores the message on the system of the sender and forwards it to the receiver by storing it to the receiving system. This can be repeated until the receiver receives the message.

In the context of this thesis, only *Send and forget* is considered.

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 proto-

cols and routed to the appropriate receiving system. Section 2.7 contains a detailed discussion of performance issues of message-based service-oriented middleware.

2.3 LATENCY VS. THROUGHPUT

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

- **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.

2.3.0.1 *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.

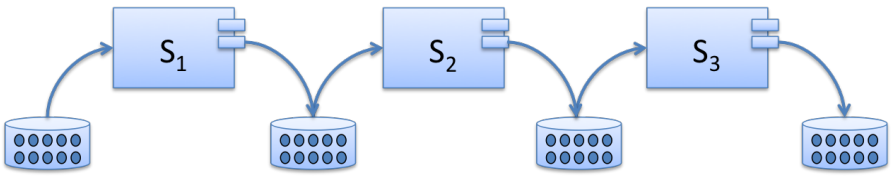


Figure 5: Batch processing system comprised of three subsystems

Assuming the system S_{Batch} which is comprised of N subsystems $S_1, S_2, \ldots, S_N$ (see Figure 5 for an example with $N = 3$):

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

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 is can be achieved by using message-based processing instead of batch processing.

2.3.0.2 *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 $MQi + 2$ and so on (see Figure 6).

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.

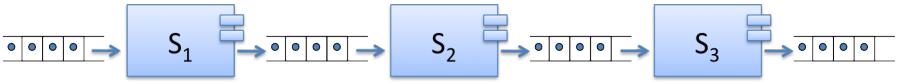


Figure 6: Message-based system comprised of three subsystems

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.

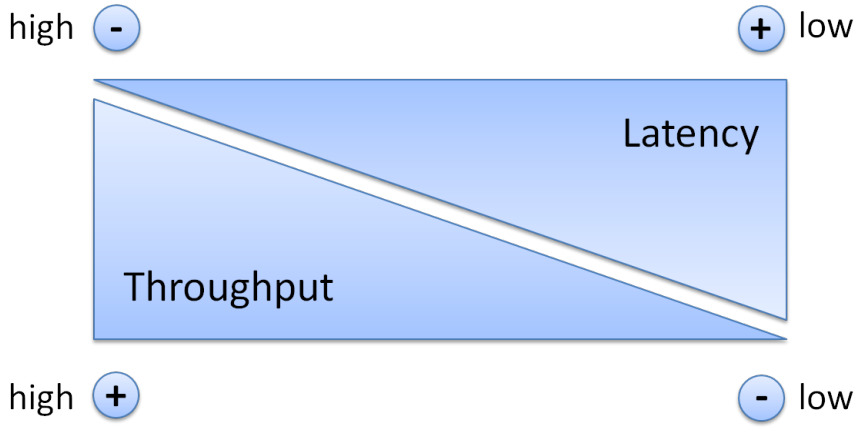


Figure 7: Latency and throughput are opposed to each other

From this follows that latency and throughput are opposed to each other (see Figure 7). 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.

2.4 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 plattform 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.

2.5 ENTERPRISE SERVICE BUS

An 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 services 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 busi-

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)

ness or technical components that are connected through the bus. The set of all service containers therefore constitutes 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.

2.6 ENTERPRISE INTEGRATION PATTERNS

Enterprise Integration Patterns (EIPs) describe a set of proven design patterns in the context of enterprise integration and messaging systems (cf. Hohpe and Woolf (2003)). The adaptive middleware presented in this thesis is based on common EIP, such as *Aggregator* and *Message Router*.

2.6.1 *Performance relevant EIPs*

The following EIPs are relevant for improving the performance of a message-based system and are used in the further course of the research presented in this thesis.

2.6.1.1 *Aggregator*

The *Aggregator* is a stateful filter that correlates multiple received messages, aggregates them, and writes them as single message to its output channel (see Figure 8).

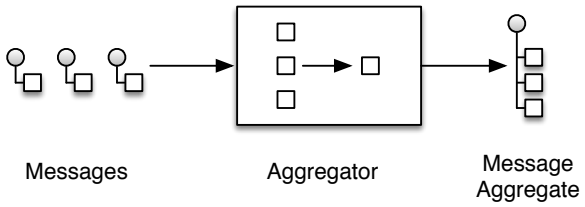


Figure 8: Aggregator (Hohpe and Woolf, 2003)

It is defined by the following properties:

- **Correlation**
  Defines which messages should be correlated with each other.

- **Completeness Condition**
  Defines when a set of messages is ready to be written to the output channel.

- **Aggregation Algorithm**
  Defines how the received messages should be aggregated to a single message.

Hohpe and Woolf (2003) describe the following most common strategies for completeness conditions:

- **Wait for All**
  The aggregation is completed, when all messages are received.

- **Timeout**
  The aggregation is completed when a defined timout occurs.

- **First Best**
  The *Aggregator* waits until the first message is received.

- **Timeout with Override**
  The *Aggregator* waits until a defined timeout occurs or until a message with a special content is received.

- **External Event**
  The aggregation is completed by an external event, for example the end of a business day.

Additionally, the authors describe the following strategies to aggregate messages into a single message (Hohpe and Woolf, 2003):

- **Select the best answer**
  Only the "best" messages is passed to the output channel, all other messages are dismissed, for example the lowest bid for an item.

- **Condense data**
  The message data is aggregated into a single value, for example computing an average or a sum of a numerical value.

- **Collect data for later evaluation**
  Messages are simply combined into a single message. The decision how to aggregate can be done later by another component.

2.6.1.2 *Message Router*

The *Message Router* reads messages from an input message channel and sends it to different output channels, depending on a set of conditions defined in the *Message Router* (see Figure 9).

A *Message Router* can implement different types of message routing:

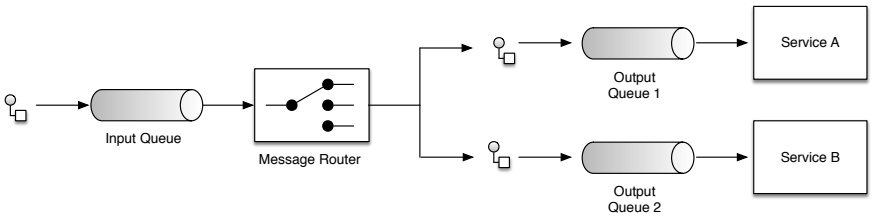


Figure 9: Message Router (Hohpe and Woolf, 2003)

- **Content-based routing**
  The routing is based on the properties of a message, for example the message type or some business specific rules.
- **Context-based routing**
  The routing is based on conditions of the environment. This is used for example for load-balancing or failover strategies.
- **Dynamic routing**
  The routing is based on a dynamic rule base, which can be adapted at run-time.

2.6.1.3 *Content Filter*

A *Content Filter* removes or simplifies unneeded data items from a message, only needed data items are left (see Figure 10).

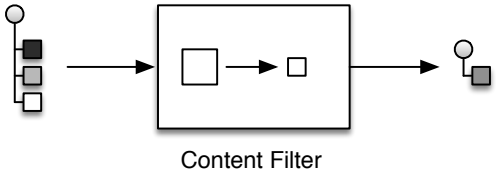


Figure 10: Content Filter (Hohpe and Woolf, 2003)

2.6.1.4 *Claim Check*

Since messaging adds an additional overhead to the processing of each message, for example by serializing and deserializing of data, it may be inefficient to send large volumes of data over a messaging system (cf. Hohpe and Woolf (2003)). The *Claim Check* pattern can used to mitigate this problem. It stores the payload of a message in a persistent data store and passes a unique identifier, the claim check, to the next components. Using this identifier, a component can retrieve the message payload from the data store and process the message (see Figure 11).

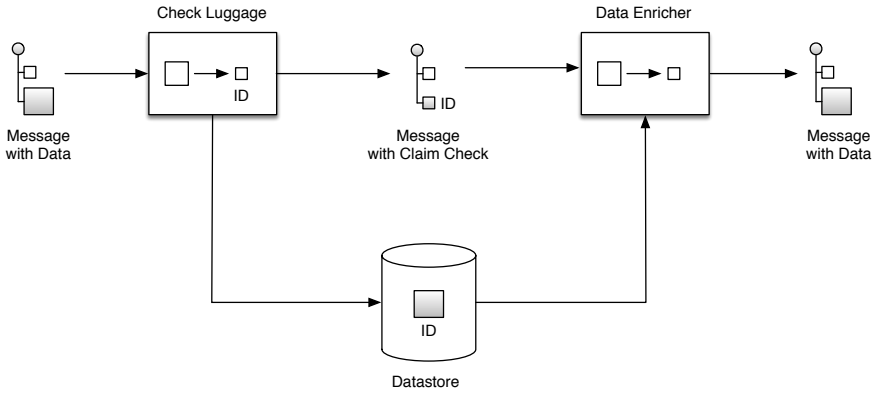


Figure 11: Claim Check (Hohpe and Woolf, 2003)

2.7 PERFORMANCE ISSUES OF SERVICE-ORIENTED MIDDLEWARE

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

2.7.1 *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.

2.7.2 *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 map-

ping 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)).

2.7.3 *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 become 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 providers 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.

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

2.8 CURRENT APPROACHES FOR IMPROVING THE PERFORMANCE OF AN SOA MIDDLEWARE

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

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

2.8.2 *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.

2.8.3 *Service Granularity*

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

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”.

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).

2.8.4 *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 inventory 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.

2.8.5 *Scaling*

Scalability describes the "ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement" (Bondi, 2000).

Weinstock and Goodenough (2006) define scalability as:

1. The ability to handle increased workload (without adding resources to a system).

2. The ability to handle increased workload by repeatedly applying a cost-effective strategy for extending a system's capacity.

- **Horizontal scaling**
  Horizontal scaling involves adding more nodes to system, for example adding more servers to a distributed system and using a load-balancer to distribute the work between them.

- **Vertical scaling**
  Vertical scaling involves adding more resources to single node, such as additional Central Processing Units (CPUs) or memory.

When a system is faced with infrequent load spikes, static scaling can lead to an overprovisioning of resources. The system is optimized to handle the load spikes, but is idle during the rest of the time.

2.8.6 *Dynamic Scaling*

A solution to prevent overprovisioning and 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.). This is also called elasticity in the context of cloud computing (cf. Herbst et al. (2013)).

While scaling is a common approach to improve the performance of a system, it also leads to additional operational and possible license costs. The solution presented in this thesis can be combined with these auto-scaling approaches to further increase the performance of the system.

2.9 SUMMARY

Systems for bulk data processing are traditionally implemented using batch processing, for example billing systems for telecommunication providers. The performance requirements for such systems are high. They have to process millions of records in a fixed timeframe to comply with service level agreements.

Message-oriented middleware facilitates the integration of applications using asynchronous messages. 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. In the context of an SOA, an Enterprise Service Bus is a common messaging middleware combining messaging, web services, data transformation and intelligent routing.

Common approaches to improve the performance of message-based systems try to reduce the transmission time by compressing messages, to adjust the service granularity to form more coarse-grained services or to adjust the degree of loose coupling to reduce the communication overhead. Scaling is a common approach to optimize the performance of of data processing system. Dynamic scaling is another solution to handle infrequent load spikes.

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.

Systems are currently either optimized for bulk data processing or low latency. Alternatively, they use different components for batch and real-time processing. The proposed approached in this thesis, a middleware that is able to adapt its processing style fluently based on the current load of the system is a new approach which has not be considered so far by the state of art to the best of the author's knowlegde.

3 RELATED WORK

This chapter gives an overview of work related to the research presented in this thesis. It starts with work that addresses the performance of Service-Oriented systems in general. Performance optimizations are discussed in the context of transport optimization, middleware optimizations and message batching.

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.

The proposed middleware uses a feedback-control loop to control the message aggregation of the system. The chapter gives a brief overview of feedback-control of computing systems.

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.

This chapter discusses work generally related to the conducted research and the adaptive middleware presented in Chapter 5.1. Related work specific to performance monitoring and modelling, which is relevant for the performance evaluation of batch and messaging systems (Section 4) is discussed in Section 4.5. Related work specific to Software Process Modelling, which is relevant for the conceptual Framework (Section 6) is discussed in Section 6.10.

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 plattforms**
  The interoperability of services on different platforms is realized 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, for example 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 OPTIMIZATION

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

3.2.1 *Transport Optimization*

In particular, various approaches have been proposed to optimize the performance of SOAP, the standard protocol for Web Service communication. This includes approaches for optimizing 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 a data layer, this facilitates the use of special data transfer methods such 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, Grasselt, 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 maximize 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

In contrast, the optimization approach presented in this thesis is aimed at the integration layer of messaging system, which allows further optimizations, such as dynamic message batching and message routing.

3.2.2 *Middleware Optimizations*

Some research has been done to add real-time capabalities to ESB or messaging middleware. Garces-Erice (2009) proposes an architecture for a real-time messaging middleware based on an ESB. 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 bandwith allocation. The proposed system is able to dynamically allocate bandwidth according to business requirements.

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.

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 with stringent requirements of low latency with high throughput. The main principle of Tempo is to use an independent queuing system for each topic. Ressources 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.

In contrast to this approaches, the approach presented in this thesis is based on a standard middleware and can be used with several integration technologies, such as JMS or SOAP.

3.2.3 *Message Batching*

Aggregating or batching of messages is a common approach for optimizing performance and has been applied to several domains. TCP Nagle's algorithm is a well-known example of this approach (Nagle, 1984).

Message batching for optimizing the throughput of Total Ordering Protocols (TOP) have first been investigated by Friedman and Renesse (1997). In their work, the authors have compared the throughput and latency of four different Total Ordering Protocols. They conclude that "batching messages is the most important optimization a protocol can offer".

Bartoli et al. (2003) extend the work of Friedman and Renesse (1997) with a policy for varying the batch level automatically, based on dynamic estimates of the optimal batch level.

Romano and Leonetti (2012) present a mechanism for self-tuning the batching level of Sequencer-based Total Order Broadcast Protocols (STOB), that combines analytical modelling an Reinforcement Learning (RL) techniques.

Didona et al. (2012) propose a self-tuning algorithm based on extremum seeking optimization principles for controlling the batching level of a Total Order Broadcast algorithm. It uses multiple instances of extremum seeking optimizers, each instance is associated with a distinct value of batching b and learns the optimal waiting time for a batch of size b.

Friedman and Hadad (2006) describe two generic adaptive batching schemes for replicated servers, which adapt their batching level automatically and immediately according to the current communication load, without any explicit monitoring of the system.

The approach presented in this research applies the concept of dynamic message batching to minimize the end-to-end latency of a message-based system for bulk data processing.

3.3 SELF-ADAPTIVE SOFTWARE SYSTEMS

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) provide 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.

3.3.1 *Reference Architectures for Self-Adaptive Software Systems*

Several reference architetures for self-adaptive software systems have been proposed. We discuss the three most common: *Kramer's Three Layer Architecture Model for Self-Management* (Kramer and Magee, 2007), *Anderson's Reflection Reference Model* (Andersson et al., 2009) and *IBM's MAPE-K* (IBM Group, 2005).

3.3.1.1 *Three Layer Architecture Model for Self-Management*

Kramer and Magee (2007) describe a *Three Layer Architecture Model for Self-Management* which is based on Gat's three-layered architecture (cf. Gat (1998)) (see Figure 12). It consists of the following three layers:

- **Component Control**
  The bottom layer consists of a set of interconnected components that implement the application function of the system. It also contains facilities to report the current status of components to higher layers and capabilities to support component creation, deletion and interconnection.

- **Change Management**
  The middle layer is responsible for effecting changes to the underlying layer in response to new states reported by the underlying layer or in response to new objectives required by the layer above.

- **Goal Management**
  This layer produces change management plans in response to request from the layer below and in response to the introduction of new goals.

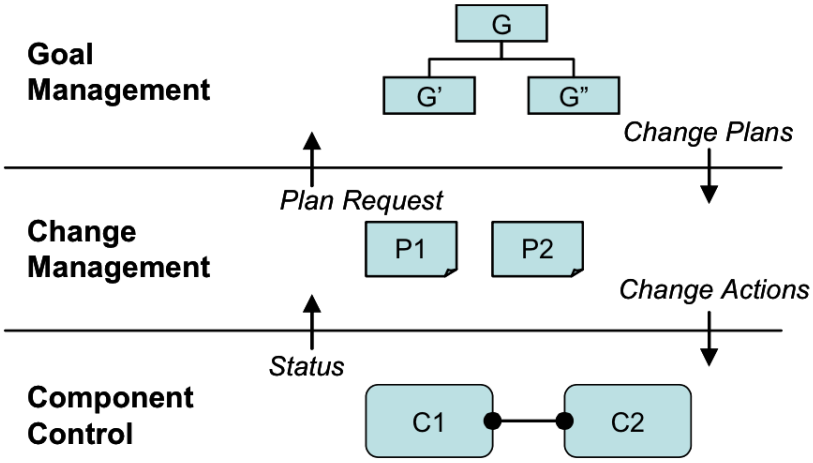


Figure 12: Three Layer Architecture Model for Self-Management (Kramer and Magee, 2007)

3.3.1.2 *Reflection Reference Model*

Andersson et al. (2009) propose a reference model for reflection (see Figure 13). The model consists of two parts, the *meta-level* and the *base-level*. The *base-level* provides the functionality of the system and contains a computation part and a domain model. The *meta-level* provides the reflective capability and consists of two parts, *meta-computation* and *meta-model*. The *meta model* is the self-representation of the system. *Meta-computation* is the logic dealing with the changes in the *meta model*.

3.3.1.3 *Mape-K*

IBM's *Mape-K* (*Monitor - Analyze - Plan - Execute - Knowlegde*) describes a reference architecture for adaptation control loops (IBM Group, 2005). It consists of the following elements:

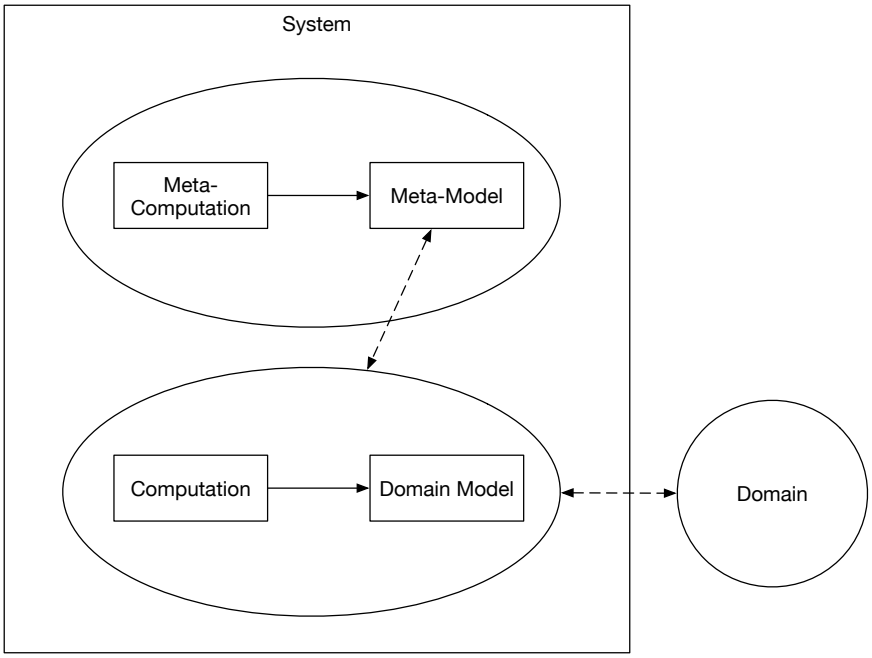


Figure 13: Reflection Reference Model (Andersson et al., 2009)

- **Monitor**
  The monitor function collects the details from the managed resources, via touchpoints, and correlates them into symptoms that can be analyzed.
- **Analyze**
  The analyze function provides the mechanisms to observe and analyze situations to determine if some change needs to be made.
- **Plan**
  The plan function creates or selects a procedure to enact a desired alteration in the managed resource.
- **Execute**
  The execute function provides the mechanism to schedule and perform the necessary changes to the system.
- **Knowledge**
  The four functions (monitor, analyze, plan, execute) share data in the *Knowledge Source*, which includes topology information, historical logs, metrics, symptoms and policies.

3.4 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". Adap-

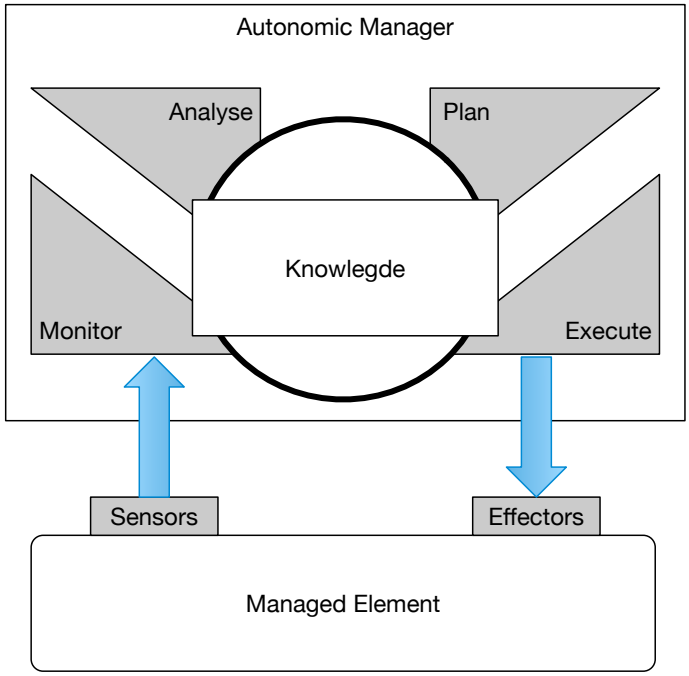


Figure 14: Mape-K Reference Model (IBM Group, 2005)

tion 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 optimizes 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 SLAs 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. It 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.

3.4.1 *Adaption in Service-Oriented Architectures*

Several adaption methods have been proposed in the context of service-based applications. In their survey, Kazhamiakin et al. (2010) describe the following adaption methods:

- **Adaption by Dynamic Service Binding**
  This adaption method relies on the ability to select and dynamically substitute services at run-time or at deployment-time. Services are selected in such a way that the adaption requirements are satisfied in the best possible way.

- **Quality of Service (QoS)-Driven Adaption of Service Compositions**
  The goal of this adaption approach is to select the best set of services available at run-time, under consideration of process constraints, end-user preferences and the execution context.

- **Adaption of Service Interfaces and Protocols**
  The goal of this adaption approach is to mediate between two services with different signatures, interfaces and protocols. This includes signature-based adaption, ontology-based adaption or behavior-based adaption.

3.4.2 *Adaptive ESB*

Research on messaging middleware currently focusses on 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).

The DRESR (Dynamic Reconfigurable ESB Service Routing), proposed by Bai et al. (2007), allows the routing table to be changed dynamically at run-time based on service selection preferences, such as response time. It defines mechanisms to test and evaluate the availability and performance of a service and to select services based on its testing results and historical data.

Ziyaeva et al. (2008) propose a framework for content-based intelligent routing. It evaluates the service availability and selects services based on its content and properties.

Jongtaveesataporn and Takada (2010) propose a load balancing mechanism that distributes requests to services of the same *service type*, having the same function and signature, and enables the dynamic selection of the target service.

Work to manage and improve the 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 adress 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.

In contrast to this solutions, the approach presented in this thesis uses dynamic message aggregation and message routing as adaption mechanism to optimize the end-to-end latency of messaging system for different load scenarios.

3.5 FEEDBACK CONTROL OF COMPUTING SYSTEMS

Control Engineering Methodologies have been identified as a promising solution to implement self-adaptive software systems (Patikirikorala 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 purpose of a controller is called control objective, the most common control objectives are (Abdelzaher et al., 2008):

- **Regulatory control**
  Ensure that the measured output is equal to (or near) the reference input.
- **Disturbance rejection**
  Ensure that disturbances acting on the system do not significantly affect the measured output.
- **Optimization**
  Obtain the best value of the measured output.

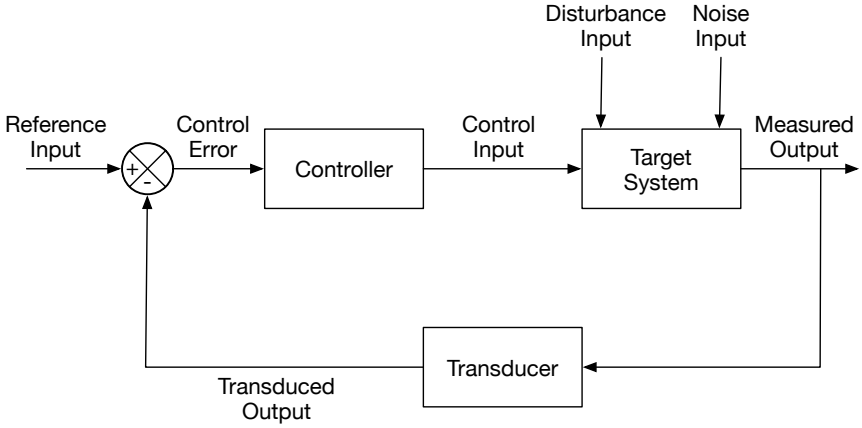


Figure 15: Block diagram of feedback control system (Hellerstein et al., 2004)

Figure 15 shows the essential elements of a single-input, single-output (SISO) control system (Hellerstein et al., 2004):

- **Control error**
  The difference between the reference input and the measured output.
- **Control input**
  The parameter that affects the behaviour of the controlled system.

- **Controller**
  The controller determines the setting of the control input needed to achieve the reference input.
- **Disturbance input**
  Any change that affects the way in which the control input influences the measured output.
- **Measured output**
  The measurable parameter of the controlled system.
- **Noise input**
  Any effect that changes the measured output produced by the controlled system.
- **Reference Input**
  The desired value of the measured output.
- **Target system**
  The computing system to be controlled.

Feedback-control has been applied to several different domains of computing systems since the early 1990s, including data networks, operating systems, middleware, multimedia and power management (cf. Hellerstein et al. (2004)). Feedback-control of middleware systems include application servers, such as the Apache http-Server, database management systems, such as IBM Universal Database Server, and e-mail servers, such as the IBM Lotus Domino Server. Hellerstein et al. (2004) describe 3 basic control problems in this context:

- Enforcing service level agreements
- Regulate resource utilization
- Optimize the system configuration

Additionally, feedback-control has been applied recently to web environments, such as web servers and web services, application servers, including data flow control in J2EE servers, Repair Management in J2EE servers and improving the performance of J2EE servers and cloud environments (cf. Gullapalli et al. (2011)).

According to Patikirikorala et al. (2012), approaches for feedback-control of computing systems use the following control mechanisms:

- **Fixed-gain control**
  Fixed-gain control uses static model parameters and gains, for example Proportional Integral Derivative (PID)-controllers.
- **Adaptive control**
  Adaptive control dynamically estimates the model parameters and gains of the controller at runtime.

- **Linear Quadratic Regulator (LQR)**
  LQR is an optimal control strategy particular useful in the MIMO control systems design. It uses a cost function representing a quadratic formula involving the control error and control effort. The goal is to mimimize the cost function so that the error is minimized with a small control effort.

- **Model predictive control (MPC)**
  MPC control algorithms aim to optimize the future behavior by computing the trajectory of the control inputs.

- **Gain scheduling**
  Gain scheduling uses a predefined logic/rule based evaluation to change the controller online. These rules are implemented in the gain scheduling component and depend on the prior knowledge about the performance variables, disturbances and conditions.

- **Cascaded (nested) control**
  The objective of cascaded control is to change the set point of the inner loop by an outer loop.

A fixed-gain controller that is widely used in exisiting approaches is a PID-Controller because of its robustness against modelling errors, disturbance rejection capabilities and simplicity (cf. Patikirikorala et al. (2012)). It 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.

The *Adaptive Middleware* presented in this thesis utilizes a closed-feedback loop to control the aggregation size of the processed messages, depending on the current load of the system to minimize the end-to-end latency of the system. This is a novel approach that has not previously been investigated.

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 Agreements (SLA) violations. It is itself implemented as a 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 analyze, manage and optimize business operations based on business goals. Optimizations 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 optimization 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 SUMMARY

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 optimize 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 us-

ing 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 coBRA framework (Irmert et al., 2008).

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).

Feedback-control is a common technique to implement the adaptive behaviour of software systems. In this thesis, a closed feedback-loop is used to control aggregation size of messaging system to minimize the end-to-end latency of the system.

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.

This thesis proposes an adaptive middleware to optimize the end-to-end latency of a system for bulk data processing by dynamically adapting its processing style between batch and single-event processing, based on the current load of the system. This is a novel approach which has not yet been discussed in current literature.

While the research presented in this thesis is based on previous work in the fields of self-adaptive middleware and feeback-control of computing systems, the work discussed in this chapter does not offer a solution for the research question stated in Section 1.2: How to achieve high-performance near-time processing of bulk data?

Part II

CONTRIBUTIONS

4

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 over- head for each processed message. Therefore, message-based processing is ususally 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,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 aggregatation 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 characterics of the two processing 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 for a telecommunications carrier that have been built to evaluate the performance of batch and message-based processing.

A billing system for telecommunications carrier is a distributed system consisting of several sub components that process the different billing sub processes like mediation, rating, billing and presentment (see Figure 16).

Figure 16: Billing process

The mediation components receive usage events from delivery systems, like telecommunication 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 17 shows the components of the billing prototype:

- **Event Generator**
  The *Event Generator* generates the calling events, i.e. the CDR that are processed by the billing application.

- **Mediation**
  The *Mediation* component checks wether the calltime of the call detail record exceeds the minimal billable length or if it belongs to a flat rate 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 call detail 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.

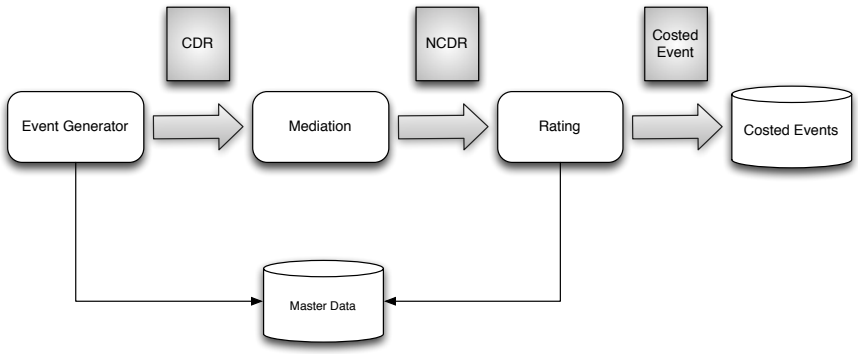


Figure 17: Components of the billing application prototype

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.

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

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.

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 UML component diagram as shown in Figure 18):

- **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 UML class diagram as shown in Figure 19):

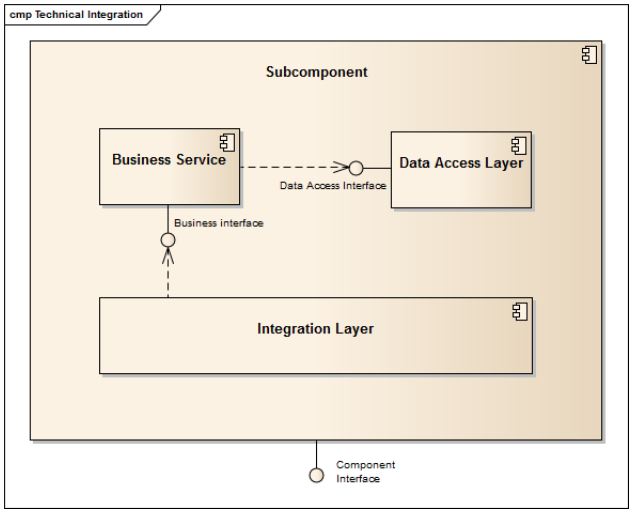


Figure 18: The prototypes share the same business components, database and data-access layer.

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

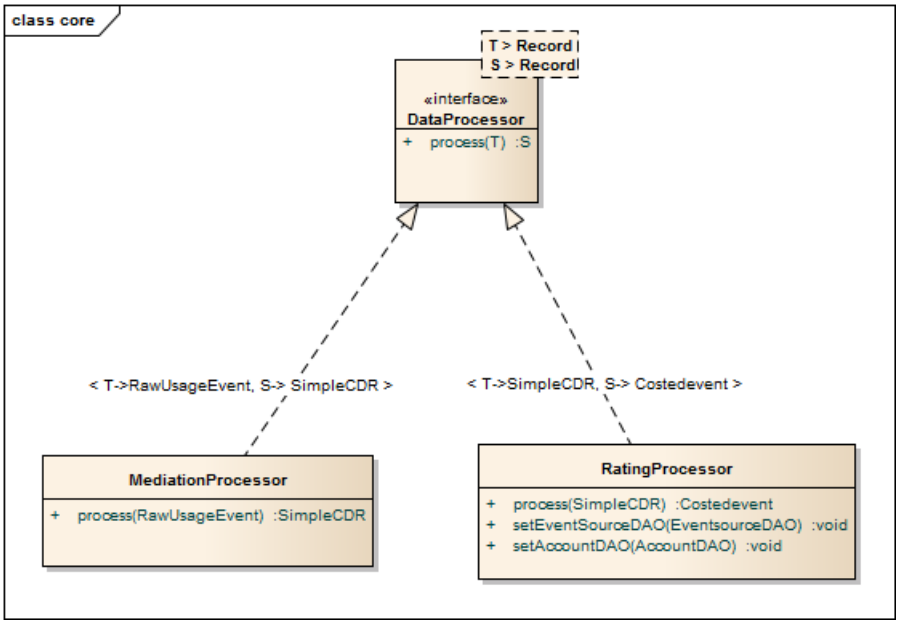


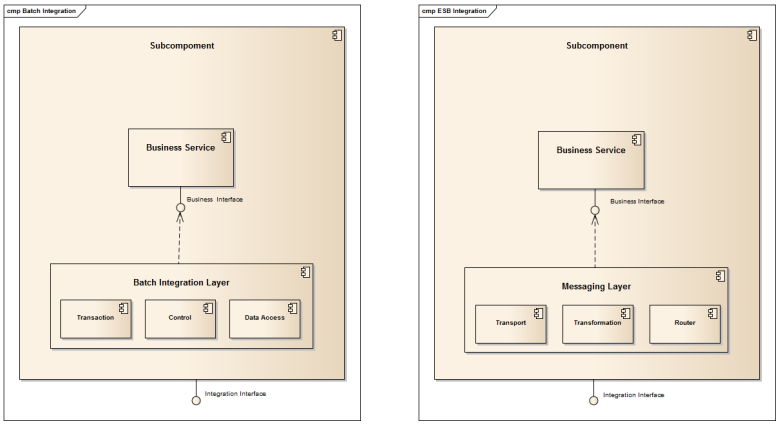
Figure 19: Business services

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 (see Figure 20a).

The messaging prototype uses a messaging middleware for exchanging messages (see Figure 20b). The messaging middleware provides components for the transport, transformation and routing of messages.



(a) Batch integration (b) Message-based integration

Figure 20: UML component diagram: The prototypes use different integration layers.

4.2.1.3 *Data model*

The prototypes use a common data model as shown in Figure 21. 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.

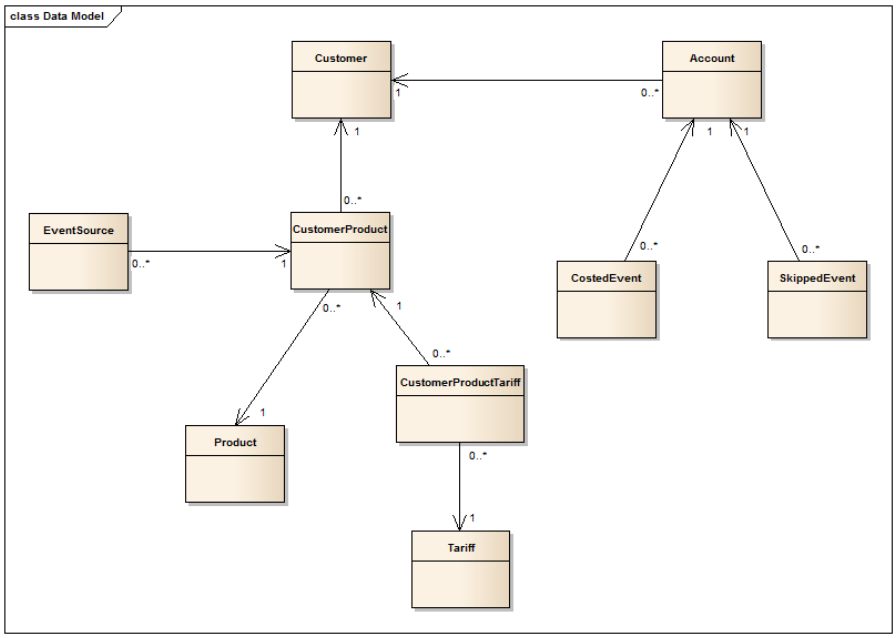


Figure 21: Logical data model of the prototype

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 22 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 FTP.

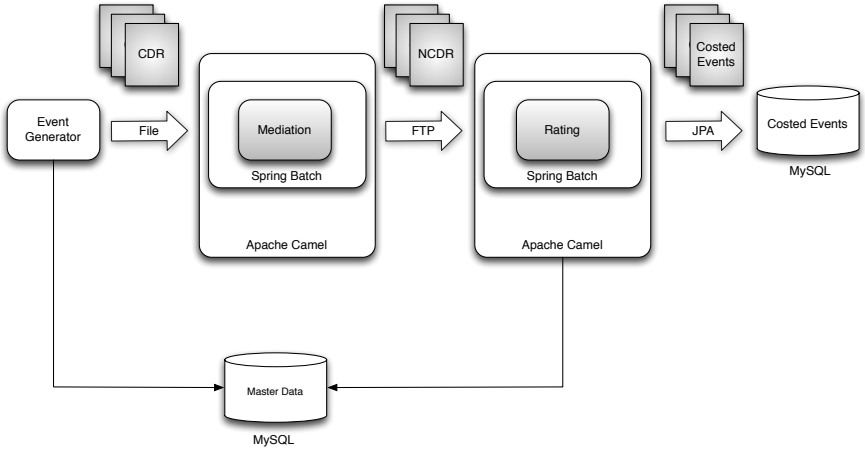


Figure 22: Batch prototype

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 transfered 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 23).

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

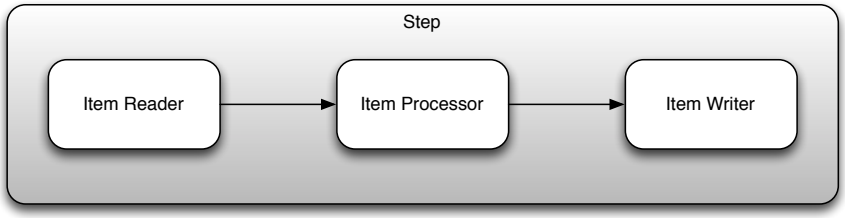


Figure 23: A Step consists of an item reader, item processor and item writer

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

Listing 4.1: 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 4.1 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 4.2 shows the definition of the mediation batch route.

Listing 4.2: 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 4.3 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 4.3: 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 24 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.

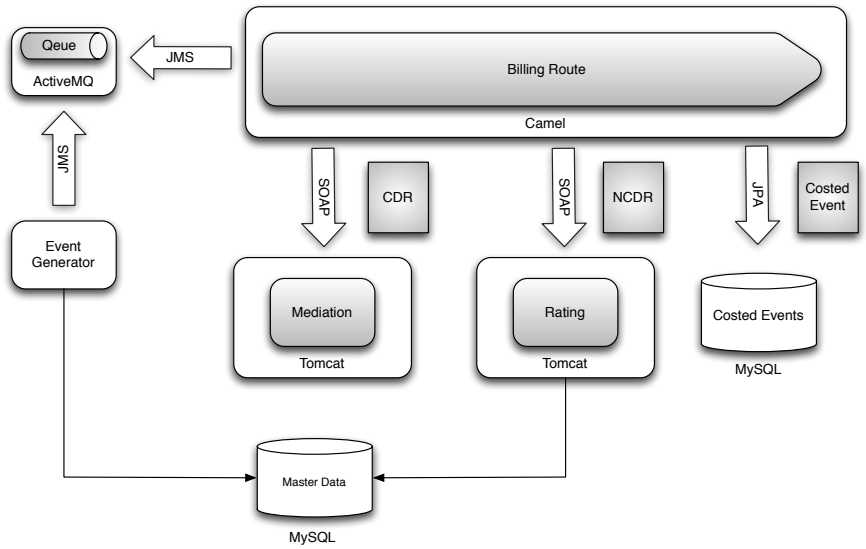


Figure 24: Message-based prototype

Listing 4.4 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 ist 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 processsing of an event, it is written to an error JMS queue (line 3).

Listing 4.4: Billing route definition

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

4.3 PERFORMANCE EVALUATION

To compare the performance characterics 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 25 and 26 show the measuring points of the batch prototype and the messaging prototype.

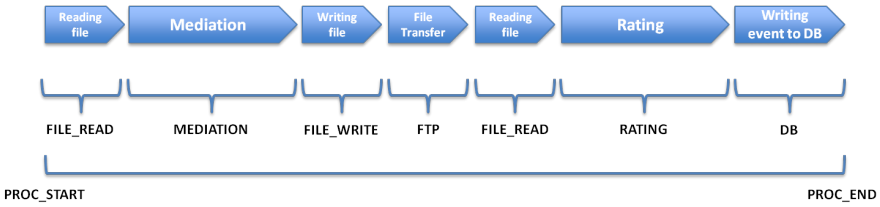


Figure 25: Measuring points of the batch prototye

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

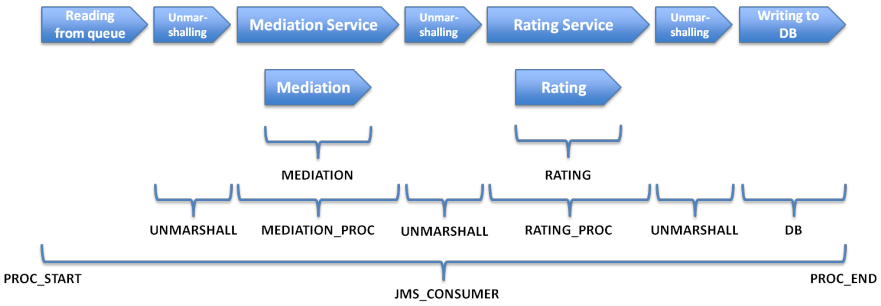


Figure 26: Measuring points of the messaging prototype

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 characterics 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 27 shows the UML deployment diagramm of the Batch prototype.

4.3.3.2 *Messaging Prototype*

The messaging prototype consists of three EC2 nodes, as shown in the UML deployment diagram in Figure 28. The *Master Node* hosts the *ActiveMQ Server* which runs the JMS queue containining 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.

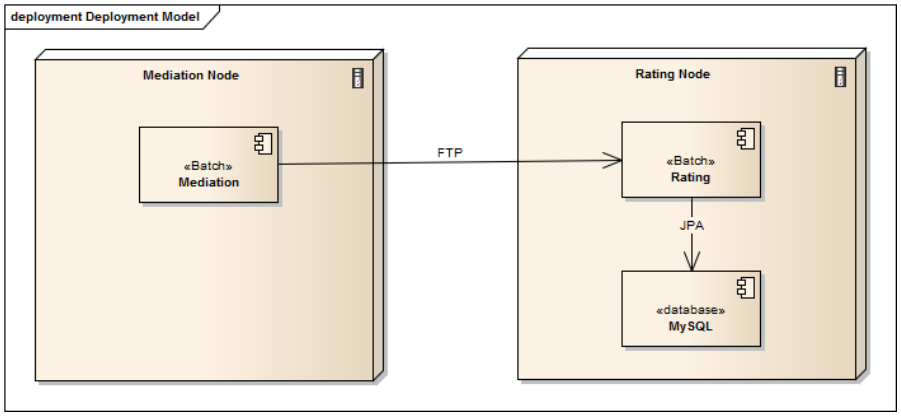


Figure 27: Batch prototype deployment on EC2 instances

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 x 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.

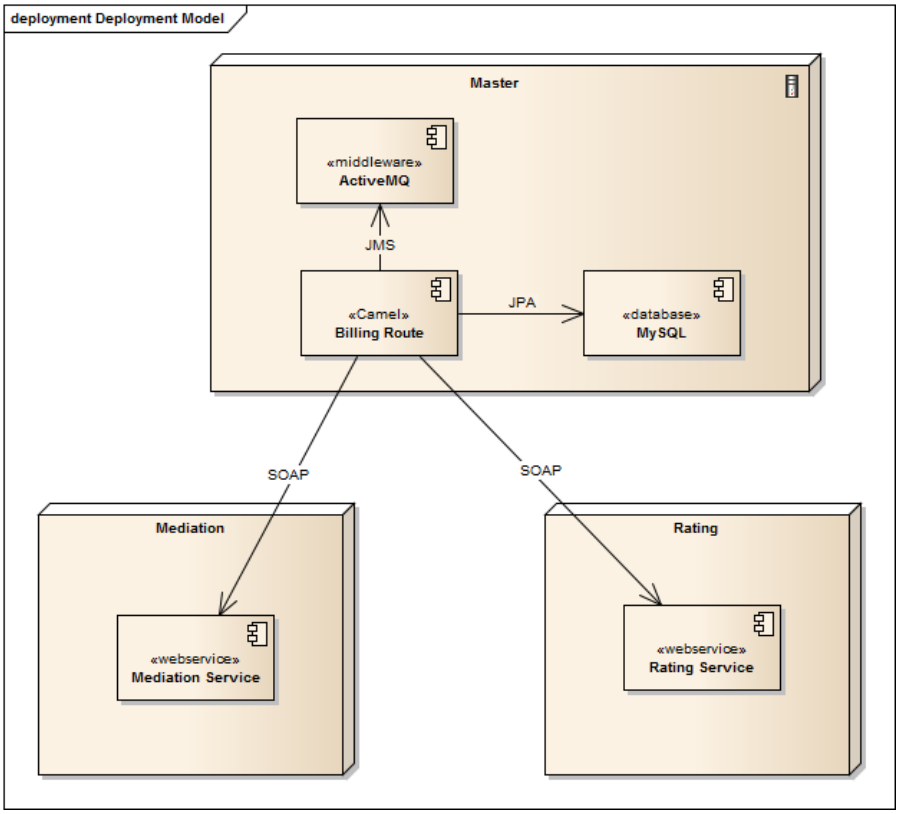


Figure 28: Messaging prototype deployment on EC2 instances

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 genarotor. 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**
   Processsing 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 29 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 380 records per second which is reached with an input of 1.000.000 records.

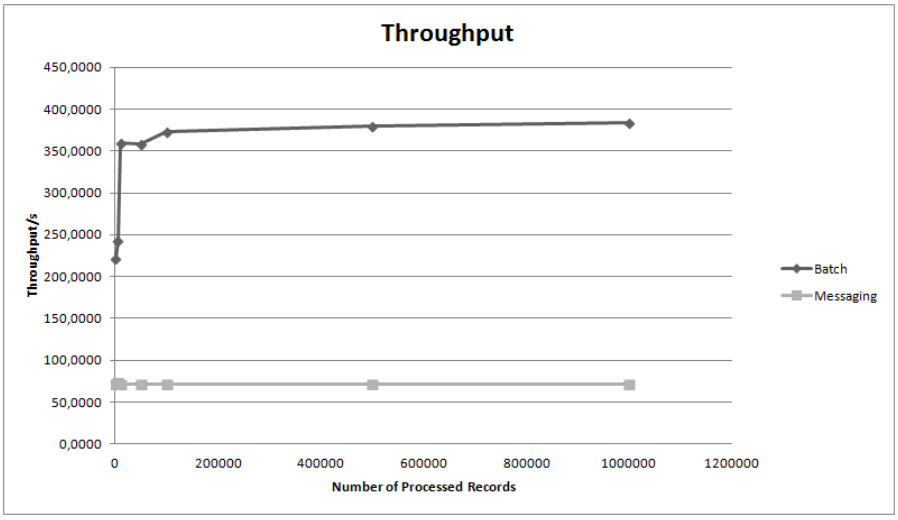


Figure 29: Throughput

4.3.6.2 *Latency*

Figure 30 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 processd 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.

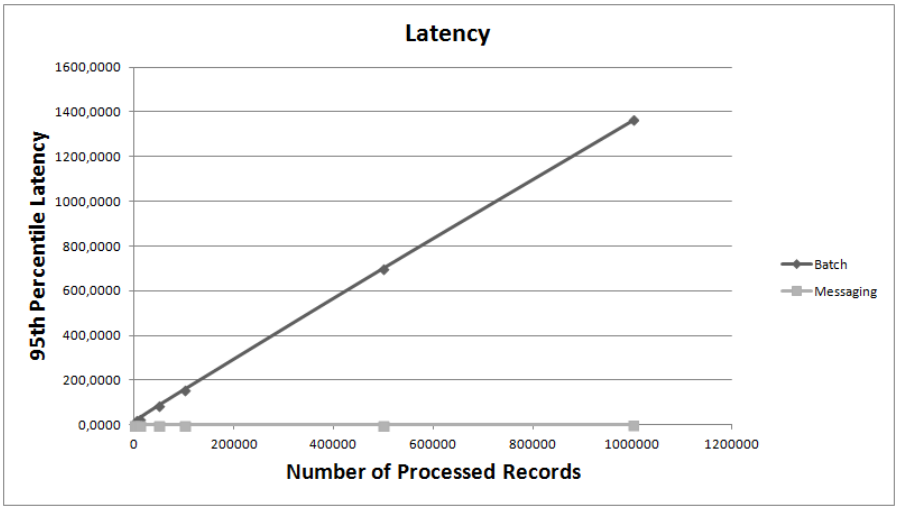


Figure 30: Latency

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 31. 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.

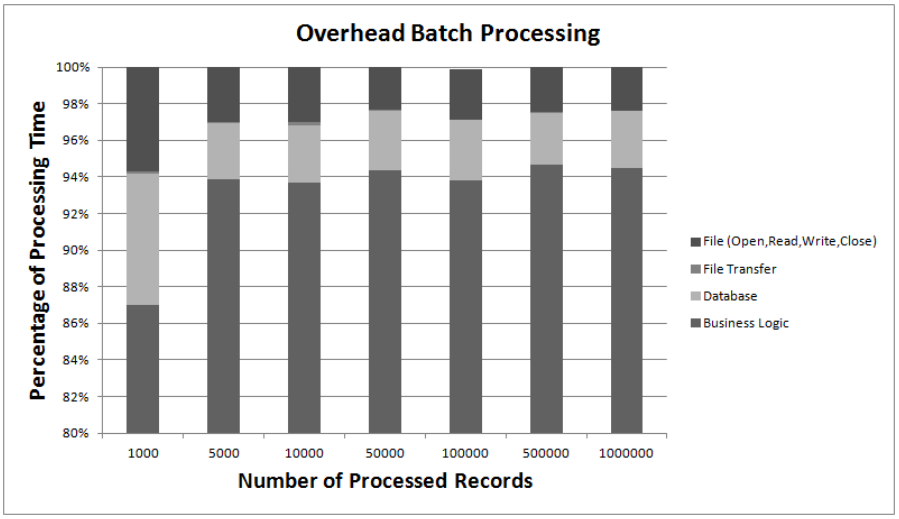


Figure 31: Overhead batch prototype

On the contrary, the overhead of the messaging prototype is about 84% of the total processing time (see Figure 32). 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.

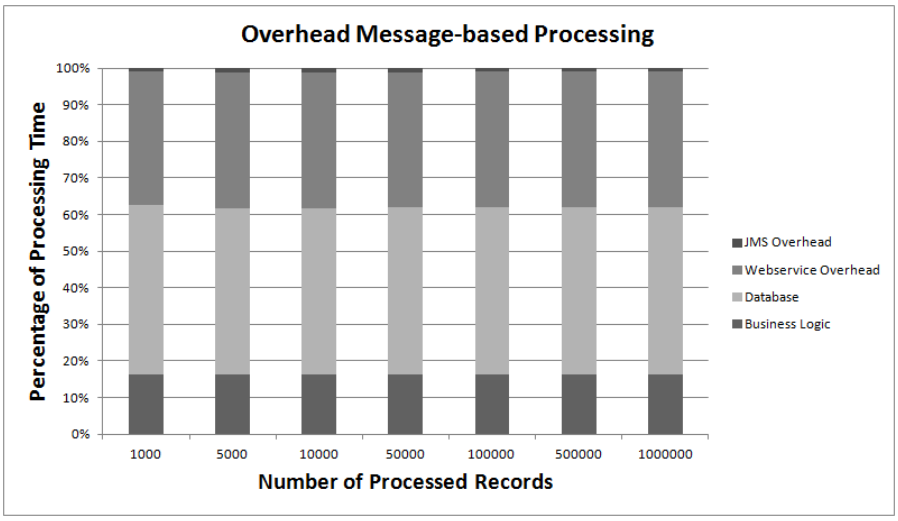


Figure 32: Overhead messaging prototype

4.3.6.4 *System utilisation*

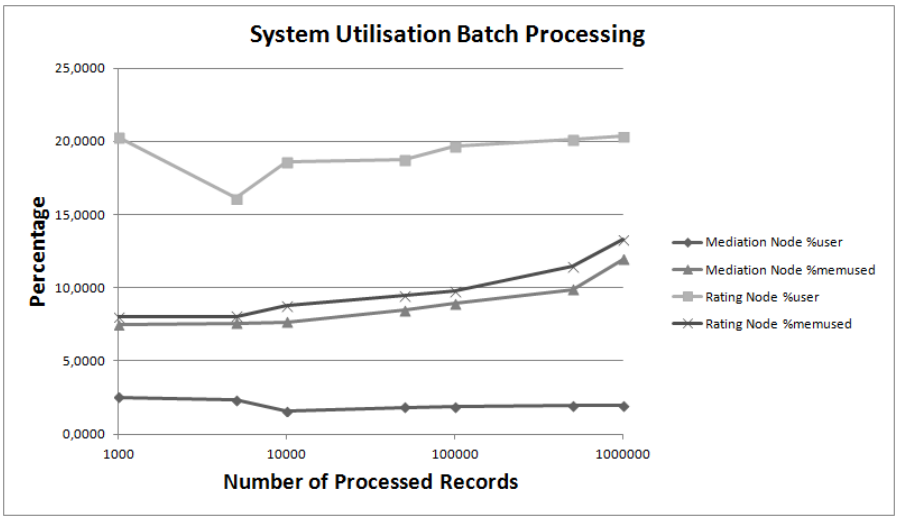


Figure 33: System utilisation batch prototype

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

ure 33 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 34 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.

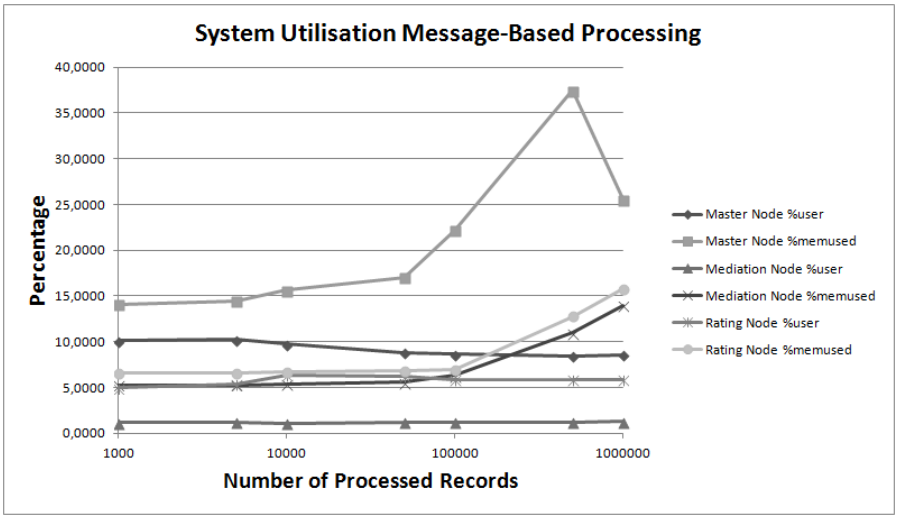


Figure 34: System utilisation messaging prototype

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-

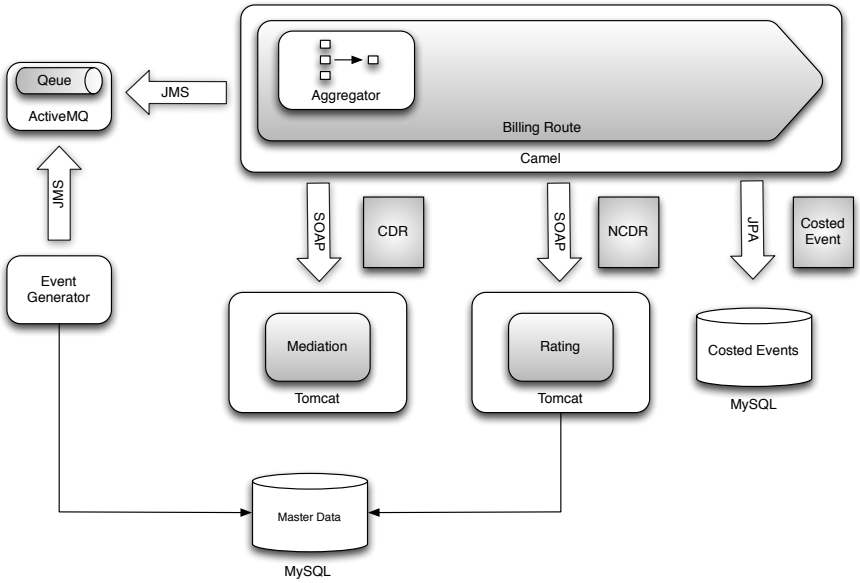


Figure 35: The data granularity is controlled by an aggregator

saging route. In case of the messaging prototype, messages are not corelated 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 corelate messages by specific data, for example an account number or by a business rule.

Listing 4.5 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 4.5: 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()
            ).completionSize(completionSize).completionTimeout(
            completionTimeout).parallelProcessing()
8       .to("cxf:bean:mediationEndpoint?dataFormat=POJO&
            headerFilterStrategy=#dropAllMessageHeadersStrategy&
            defaultOperationName=processEvents")
9       .process(new ProcessEventsPostProcessor())
10      .to("cxf:bean:ratingEndpoint?dataFormat=POJO&
            headerFilterStrategy=#dropAllMessageHeadersStrategy&
            defaultOperationName=processCallDetails")
11      .process(new ProcessCallDetailsPostProcessor())
12      .process(costedEventsProcessor);
13  }
```

Figure 36 shows the impact of different aggregatation sizes on the throughput of the messaging prototype. For each test 100.000 events have been processed. The throughput increases constantly for

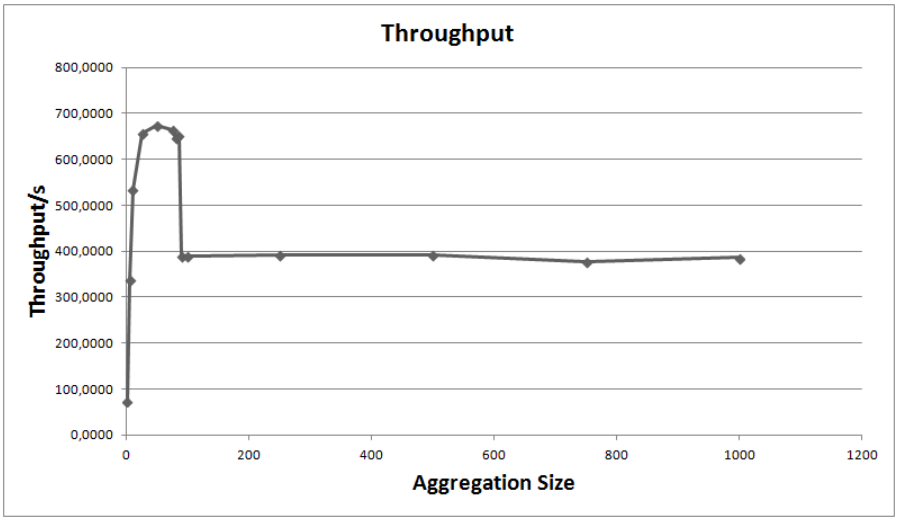


Figure 36: Impact of different aggregation sizes on throughput

$1 < aggregation\_size \leqslant 50$ with a maximum of 673 events per second with $aggregation\_size = 50$. 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 37. 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 $aggregation\_size > 50$.

The increased throughput achieved by increasing the aggregation size comes with the cost of a higher latency. Figure 38 shows the

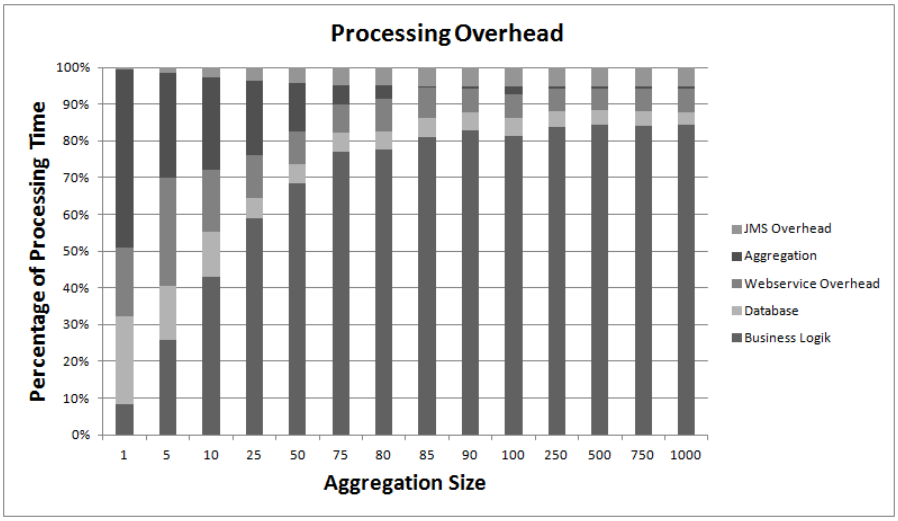


Figure 37: Impact of different aggregation sizes on processing overhead

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

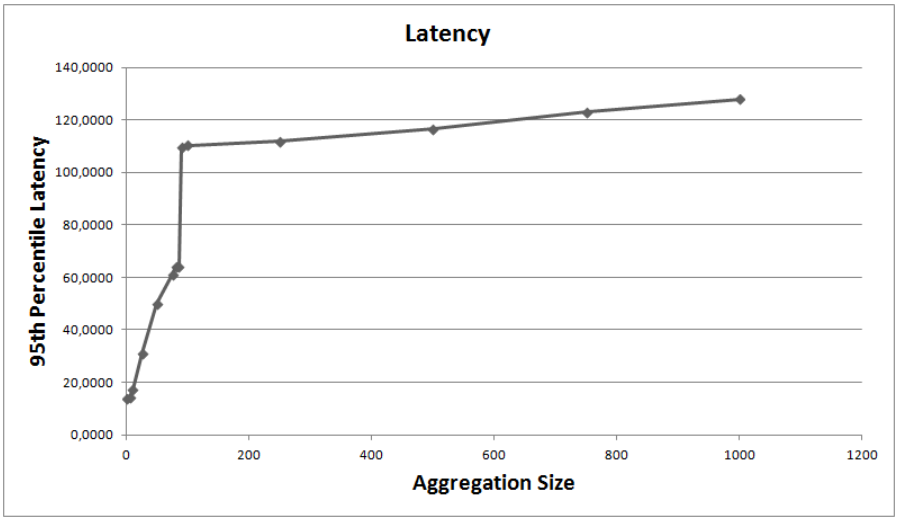


Figure 38: Impact of different aggregation sizes on latency

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 39 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 $aggregation\_size \geqslant 90$ 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, beeing less than 2% and 8%, respectively.

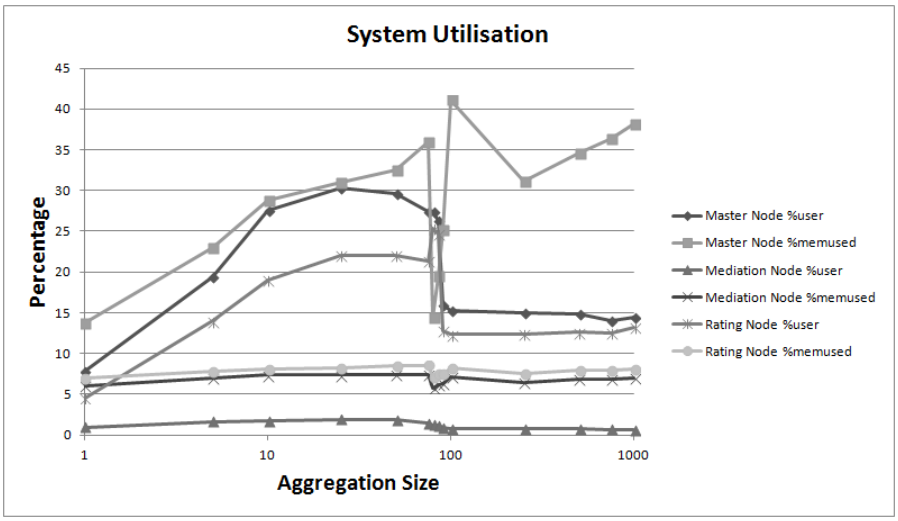


Figure 39: Impact of different aggregation sizes on system utilisation

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 protoype 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*

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

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 tournaround 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. (2006) 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 usuable 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 2.3, latency and throughput are opposed performance metrics of a system for bulk data processing. Batch processing, while providding high throughput, leads to high latency, which impedes near-time processing. Message-based 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.

5

AN ADAPTIVE MIDDLEWARE FOR NEAR-TIME PROCESSING OF BULK DATA

5.1 INTRODUCTION

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. Finally, Section 5.8 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 middlware: (1) mes-

sage 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 disadadvantes 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 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 routing depends on how messages are aggegrated. 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 Messaging 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:

Table 7: Properties of different aggregation strategies

Aggregation Strategy	Pro	Con
No correlation	– Simple solution – Even distribution of events	– optimization is not or hardly possible
Business correlation	– Optimization is possible	– Analysation of processed data needed – No even distribution of data (depending on correlation rule)
Technical correlation	– Optimization is possible	– 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	– Aggregation size	Routing is based on the technical properties of a message aggregate.
Content-based routing	– 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.

- **Static routing:**
  Static routing uses static routic rules, that are not changed automatically.
- **Dynamic routing:**
  Dynamic routing adjusts the routing rules automatically at runtime, for example depending on 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*

In order to optimize the end-to-end latency of the system, the middleware needs to constantely monitor the load of the system and control the aggregation size accordingly (see Figure 40).

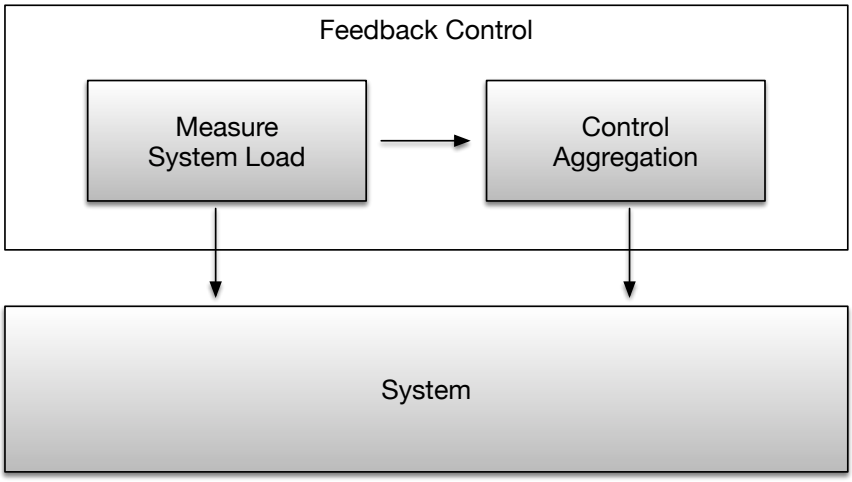


Figure 40: Monitoring and Control

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

To control the level of message aggregation at runtime, the adaptive middleware uses a closed feedback loop as shown in Figure 41, with 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.

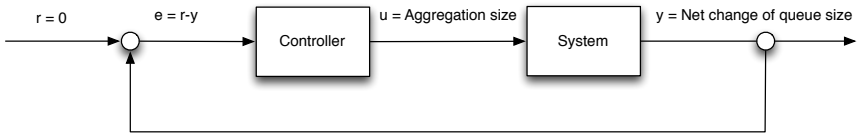


Figure 41: Feedback loop to control the aggregation size

5.4 MIDDLEWARE COMPONENTS

Figure 42 shows the components of the middleware, that are based on the Enterprise Integration Patterns described by Hohpe and Woolf (2003). A description of these components can be found in Table 9.

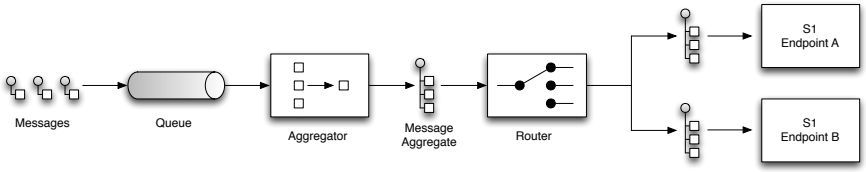


Figure 42: Middleware components

5.5 DESIGN ASPECTS

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

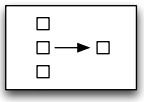
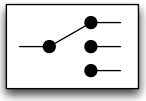
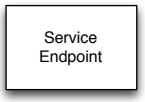
5.5.1 *Service Design*

The services that implement the business functionality of the system need to be explicitely 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.

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.

- 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.

Listing 5.1 shows the interface of a service offering different operations for batch processing (line 6) and single-event processing (line 10).

Listing 5.1: 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 }
```

5.5.2 *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 2.1 and 2.2 for a detailed description of each processing style).

When using high aggegration 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 2.6 for a detailed description of this pattern). Table 10 summarizes the transport options for low and high aggregation sizes.

Table 10: Transport options for high and low aggregation sizes

Aggregation Size	**Transport Options**
High	– Database – File-based (e.g. FTP) – Claim Check EIP
Low	– 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).

5.5.3 *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 availaible 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 beeing written to the error qeue, especially when using high aggregation sizes.

5.5.4 *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 using curve fitting to create a model of the system

- Black-box modeling

- Modeling using stochastic approaches, especially queuing theory

- Modeling using special purpose representations, for example the first principles analysis

For pratical reasons, 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. Develop the control system

5.5.4.1 *Control Problem*

The control problem is defined as follows:

- Minimize the end-to-end latency of the system by controlling the message aggregation size.
- The aggregation size used by the messaging system should depend on the current load of the system.
- When the system faces high load, the aggregation size should be increased to maximize the maximum throughput of the system.
- When the sytem faces low load, the aggregation size should be decreased to miminize the end-to-end latency of the system.

5.5.4.2 *Input/Output Signals*

Janert (2013) describes the following criteria for selecting input control signals:

- **Availability**
  It should be possible to influence the control input directly and immediately.
- **Responsiveness**
  The system should respond quickly to a change of the input signal. Inputs whose effect is subject to latency or delays should be avoided when possible.
- **Granularity**
  It should be possible to adjust the control input in small increments. If the control input can only be adjusted in fixed increments, then it could be necessary to consider this in the controller or actuator implementation.
- **Directionality**
  How does the control input impact the control output? Does an increased control input result in increased or decreased output?

Additionally, the following criteria should be considered for selecting output control signals:

- **Availability**
  The quantity must be observable without gaps and delays.
- **Relevance**
  The output signal should be relevant for the behaviour of the system that should be controlled.
- **Responsiveness**
  The output signal should reflect changes of the state of the system quickly without lags and delays.
- **Smoothness**
  The output signal should be smooth and does not need to be filtered.

With regard to these criteria, the following input and output control signals have been choosen

- **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.

5.5.4.3 *Control Strategy*

SIMPLE CONTROLLER

A simple non-linear control strategy could be implemented as follows (cf. Janert (2013)):

- When the tracking error is positive, increase the aggregation size by 1
- Do nothing when the tracking error is zero.
- Periodically decrease the aggregation size to test if a smaller queue size is able to handle the load.

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.

A PID-controller seems not a good fit for the problem at hand since the aggregation size can not be controlled continously. It is always a whole positive integer.

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.

The prototype extends the messaging prototype with the following components (see Figure 43):

- **Performance Monitor**
  The *Performance Monitor* manages the feedback-control loop by periodically calling the *Sensor* and updating the *Controller*. Additionally, it calculates the current throughput and end-to-end latency of the sytem.

- **Sensor**
  The *Sensor* is responsible for getting the current size of the message queue using Java Monitoring Extensions (JMX).

- **Controller**
  The *Controller* calculates the new value for the aggregation size base on the setpoint and the current error.

- **Actuator**
  The *Actuator* is responsible for setting the new aggregation size of the *Aggregator* calculated by the *Controller*.

5.6.1 *Aggregator*

The message aggregator uses the same *AggregationStrategy* as the messaging prototype as described in Section 4.4, as shown in Listing 5.2:

- The *aggregate* method, which is called by the aggregator for each message, takes two arguments: *oldExchange* contains the already aggregated messages, *newExchange* contains the new arrived message (line 4).

- If there are not yet any aggregated messages stored in the aggregator (line 5):
  - The message body (*rawUsageEvent*) is read from the new arrived message (line 6).
  - A new *usageEventsList* is generated (line 8).

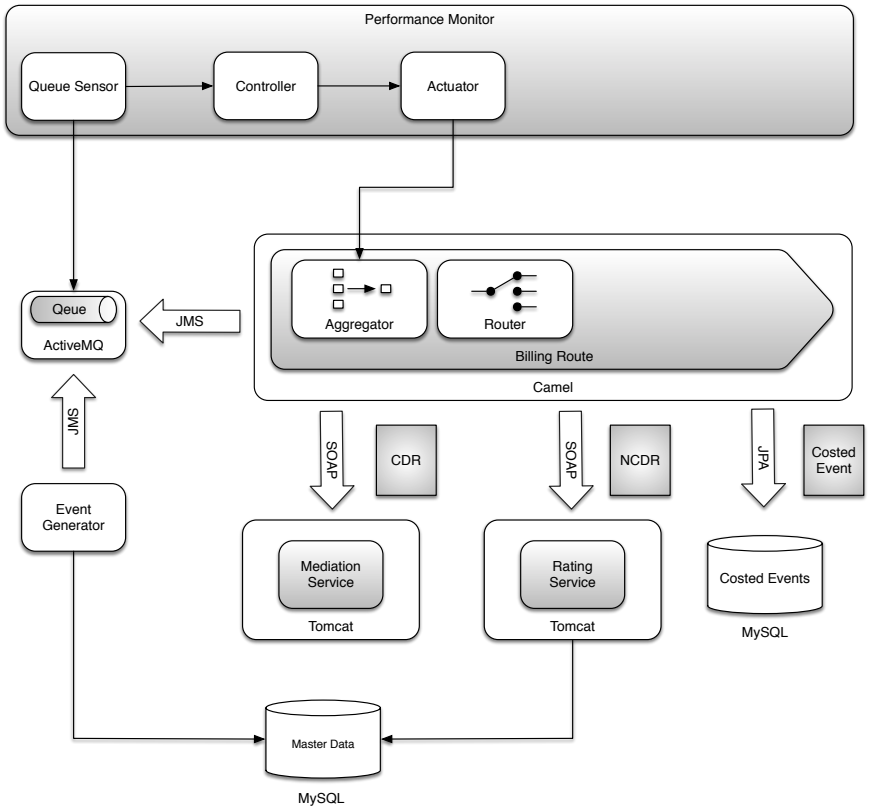


Figure 43: Components of the prototype system

  - The message body is added to the list and the list is added to the incoming message, which now becomes the new message aggregate.

- Otherwise, the message body of the new message is added to list of the aggregated messages (line 22).

Listing 5.2: 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 }
```

The aggregator is configured to dynamically use the aggregation size (*completionSize*) set by a message header, as shown in Listing 5.3

(line 2). This message header is set by the *Actuator* (see Section 5.6.2.3), which is controlled by the *Controller* (see Section 5.6.2.2).

Listing 5.3: 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 44 shows the components of the feedback-control loop.

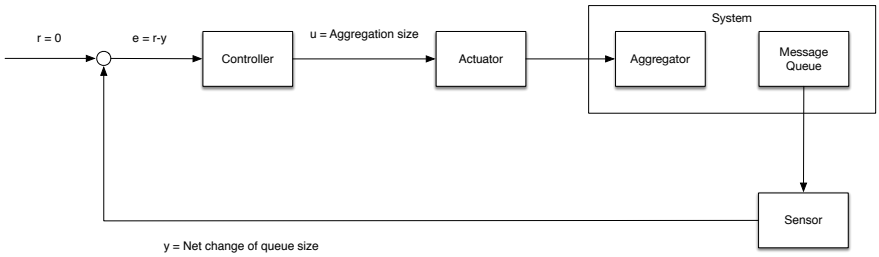


Figure 44: Components of the feedback-control loop

5.6.2.1 *Sensor*

The *JmxSensor* implements the *Sensor* interface (see Figure 45). It reads the current length of the input queue of the *ActiveMQ* server instance using JMX.

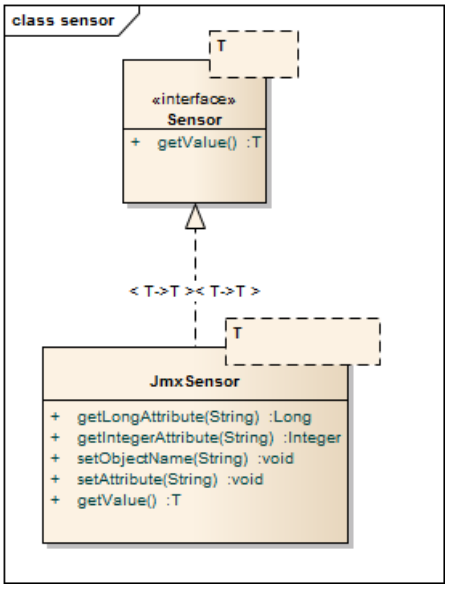


Figure 45: UML classdiagram showing the sensor classes

5.6.2.2 *Controller*

A *Controller* has to implement the *Controller* interface. The following implementations of the *Controller* interface have been implemented (see Figure 46):

- **BasicController**
  Implements a generic controller. The control strategy is provided by an implementation of the *ControllerStrategy*.
- **TestController**
  A controller used for testing the static behaviour of the system.

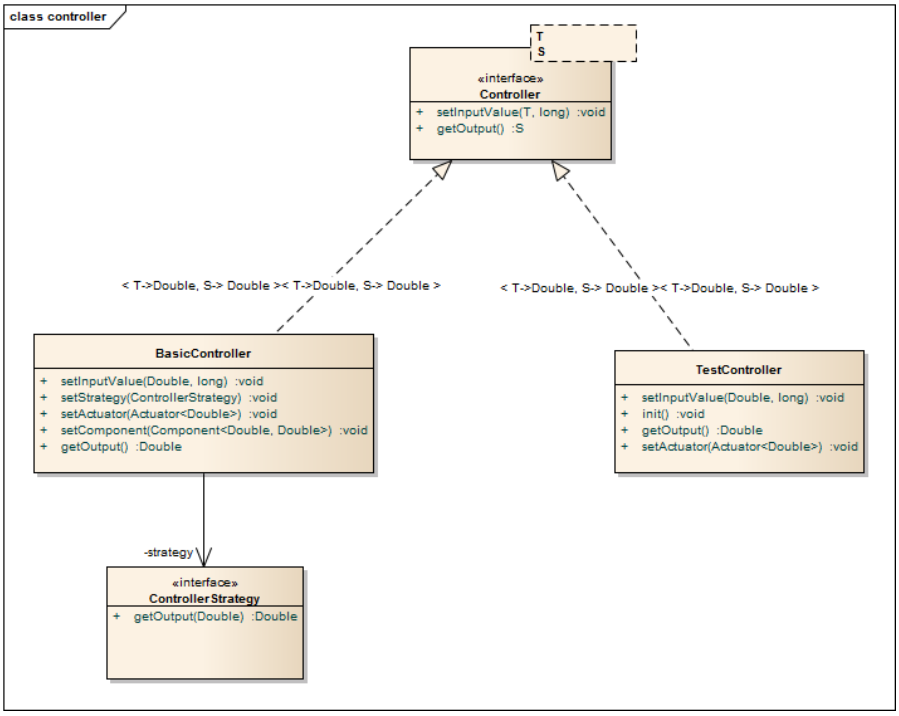


Figure 46: UML classdiagram showing the controller classes

The strategy of the controller is implemented by a controller strategy which implements the *ControllerStrategy* interface (see Listing 5.4).

Listing 5.4: ControllerStrategy Interface

```
1 package com.jswiente.phd.performance.controller;
2
3 public interface ControllerStrategy {
4   public Double getOutput(Double error);
5 }
```

Figure 47 shows the available implementations of the *ControllerStrategy*.

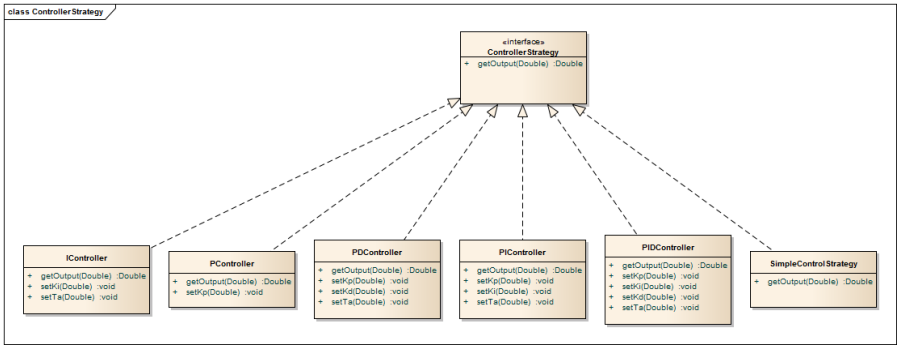


Figure 47: UML classdiagram showing the controller strategy classes

SIMPLE CONTROLLER

Listing 5.5 shows the implementation of the simple control strategy, as described in Section 5.5.4.3:

- If the queue size increases, increase the aggregation size (line 10-13).
- Otherwise, do not change the aggregation size (line 22).
- Periodically decrease the aggregation size by one (line 17-20).

The controller uses two different timers depending on the previous action.

Listing 5.5: Implementation of the simple control strategy

```
1  public class SimpleControlStrategy implements ControllerStrategy
       {
2
3    @Value("${simpleController.period1}")
4    private int period1;
5    @Value("${simpleController.period2}")
6    private int period2;
7    private int timer = 0;
8
9    public Double getOutput(Double error) {
10     if (error > 0) {
11       timer = period1;
12       return +1.0;
13     }
14
15     timer--;
16
17     if (timer == 0) {
18       timer = period2;
19       return -1.0;
20     }
21
22     return 0.0;
```

```
23    }
24
25 }
```

PID CONTROLLER

The implementation of the *PID Controller*, as described in Section 5.5.4.3 is straight forward, as shown in Listing 5.6.

Listing 5.6: 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 * (
           error - previousError)/ta);
21     previousError = error;
22     return output;
23   }
24
25   //Setter methods removed for simplification...
26
27 }
```

5.6.2.3 *Actuator*

The *AggregateSizeActuator* is responsible for setting the aggregation size of the *Aggregator* and is controlled by the *Controller* (see Figure 48).

It *AggregateSizeActuator* implements the *Actuator* interface (see Listing 5.7).

Listing 5.7: Actuator Interface

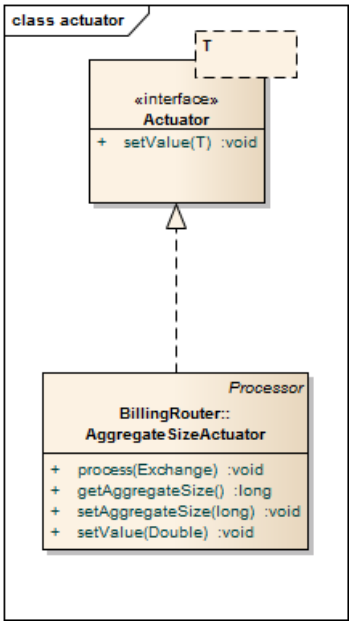


Figure 48: UML classdiagram showing the actuator classes

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

The *AggregateSizeActuator* sets the aggregation size (*completionSize*) by setting a specific header in the currently processed message, as shown in Listing 5.8 (line 15).

Listing 5.8: 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,
           aggregateSize);
16   }
```

```
17
18    @ManagedAttribute
19    public long getAggregateSize() {
20      return aggregateSize;
21    }
22
23    @ManagedAttribute
24    public void setAggregateSize(long aggregateSize) {
25      logger.debug("Setting aggregateSize to: " + aggregateSize);
26      this.aggregateSize = aggregateSize;
27    }
28
29    @Override
30    public void setValue(Double value) {
31      logger.debug("Actuator: Setting aggregateSize to: " + value);
32      long aggregateSize = Math.round(value);
33      this.setAggregateSize(aggregateSize);
34    }
35
36  }
```

5.6.2.4 *Performance Monitor*

The *Performance Monitor* manages the feedback-control loop by periodically calling the *Sensor* and updating the *Controller*. Additionally, it calculates the current throughput and end-to-end latency of the sytem using the *StatisticsService* (see Figure 49).

5.6.3 *Load Generator*

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

Figure 50 shows the UML class diagram of the load generator.

- **DataGenerator**
  This is the main class of the *DataGenerator*.
- **Writer**
  The *Writer* interface defines methods for writing the generated events. There are two implementations available, the *FileWriter*, which is used to write the generated to a file and the *JmsWriter*, which is used to write the events to a JMS queue.
- **Generator**
  The *Generator* interface defines methods for generating events.
- **Distribution**
  The *Distribution* interface represents an event distribution used

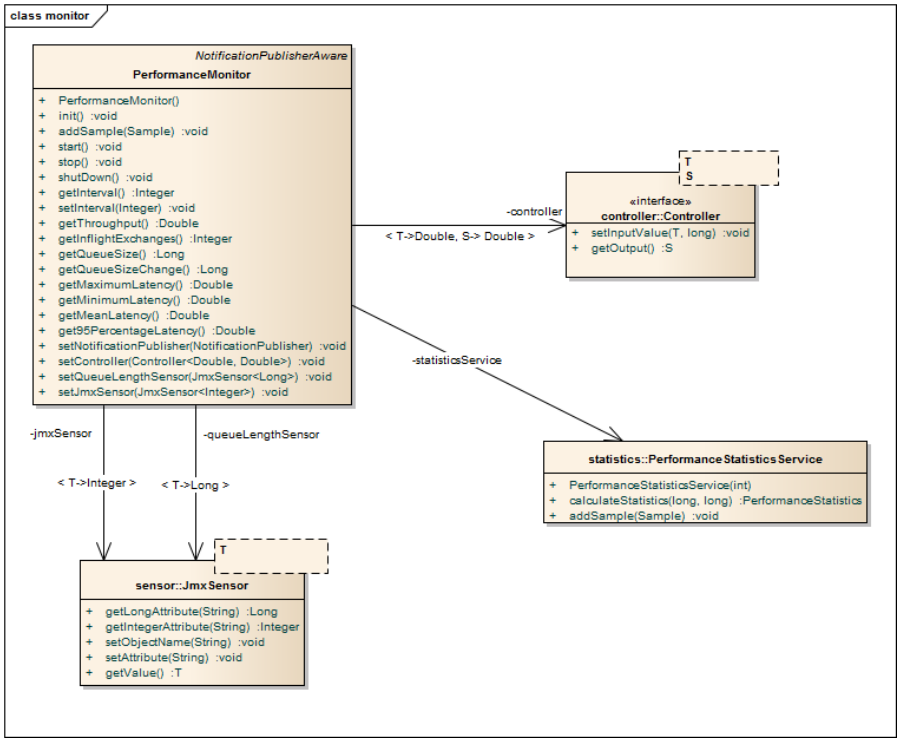


Figure 49: UML classdiagram showing the *PerformanceMonitor*

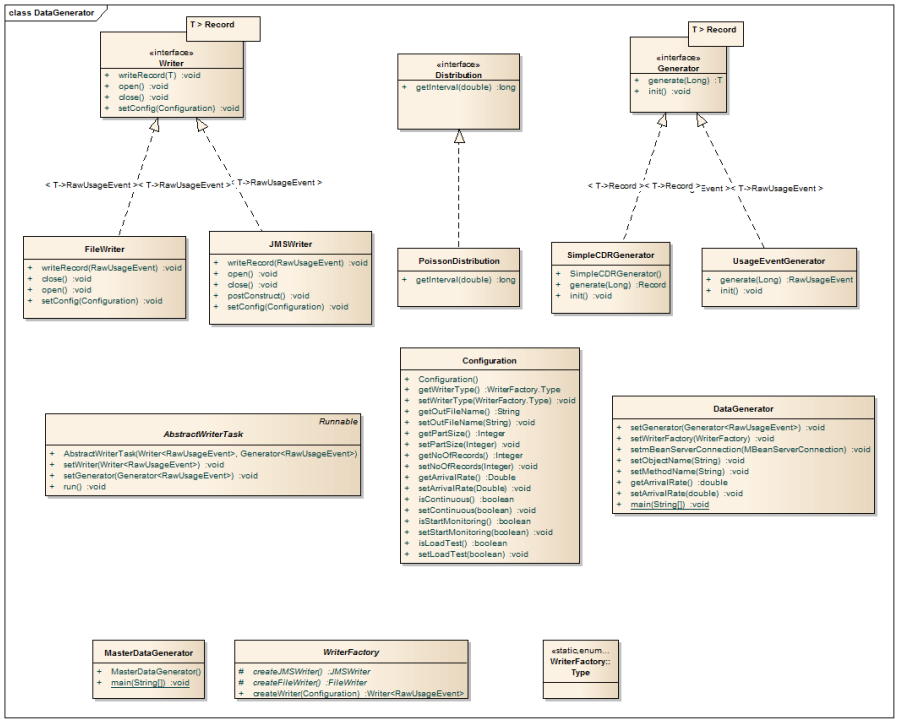


Figure 50: UML class diagram of the *Load Generator*

by the *DataGenerator*. The *PoissonDistribution* is the single implementation of this interface.

- **Configuration**
  This class holds a specific set of configuration parameters used at run-time.

The *DataGenerator* uses a *Poisson Process* to simulate the load of the system. Events occur continuously and independently of each other with exponentially distributed inter-arrival times.

5.7 EVALUATION

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

5.7.1 *Test Environment*

The tests have been run on a development machine to decrease the development-build-deploy cycle, as described in Table 11.

Table 11: Test environment

Memory	3 GiB
CPU	Intel Core i5 M520 @ 2,40 GHz
Architecture	32-bit
Disk Drive	150 GB SSD
Operating System	Windows 7
Database	MySQL 5.5.24
Messaging Middleware	Apache ActiveMQ 5.6.0

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 (**S**table, **A**ccurate, **S**ettling times, **O**vershoot):

- **Stability**
  The system should provide a bounded output for any bounded input.

- **Accuracy**
  The measured output of the control system should converge to the reference input.
- **Settling time**
  The system should converge quickly to its steady state.
- **Overshoot**
  The system should achieve its objectives in a manner that does not overshoot.

5.7.3 *Static Tests*

To test the relationship between the input and output variables of the control-loop, aggregation size and change of queue size, the following static tests have been performed:

- The *TestController* has been configured to periodically increase the aggregation size after 100 time steps (1 time step equals 1 second).
- The test has been repeated with different load of the system, that is, using different arrival rates for the *DataGenerator*.

Figure 51 shows the queues size of the system in relationship to the aggregation size, for differrent arrival rates.

- The system is not able to handle the load with an $aggregationsize < 5$ and an $arrivalrate = 50$. With an $aggregationsize \geqslant 5$, the system is able to process the events faster than they occur.
- With an $arrivalrate = 100$, the system is not able to handle the load with an $aggregationsize < 15$. With an $aggregationsize \geqslant 15$, the system is able to process the events faster than they occur.
- With an $arrivalrate = 150$, the system is not able to handle the load with an $aggregationsize < 25$. With an $aggregationsize \geqslant 25$, the system is able process the events faster than they occur.

The change of queue size between each time step is shown in Figure 52.

5.7.4 *Step Test*

To measure the dynamic response of the system, the following step test have been performed:

- The *TestController* has been configured to increase the aggregation size from 1 to 50.

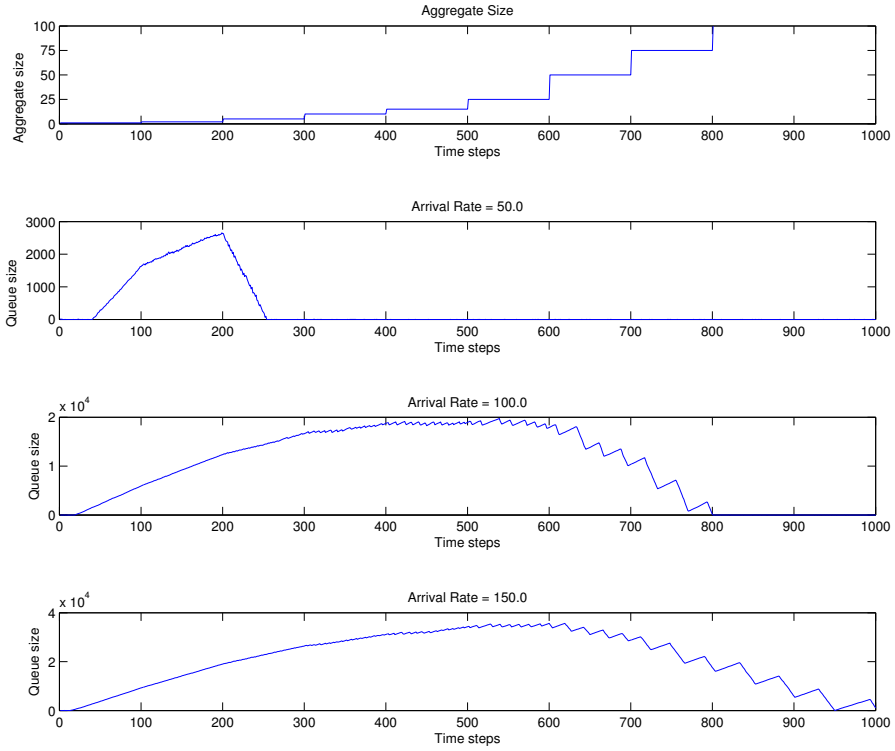


Figure 51: Static test: queue sizes

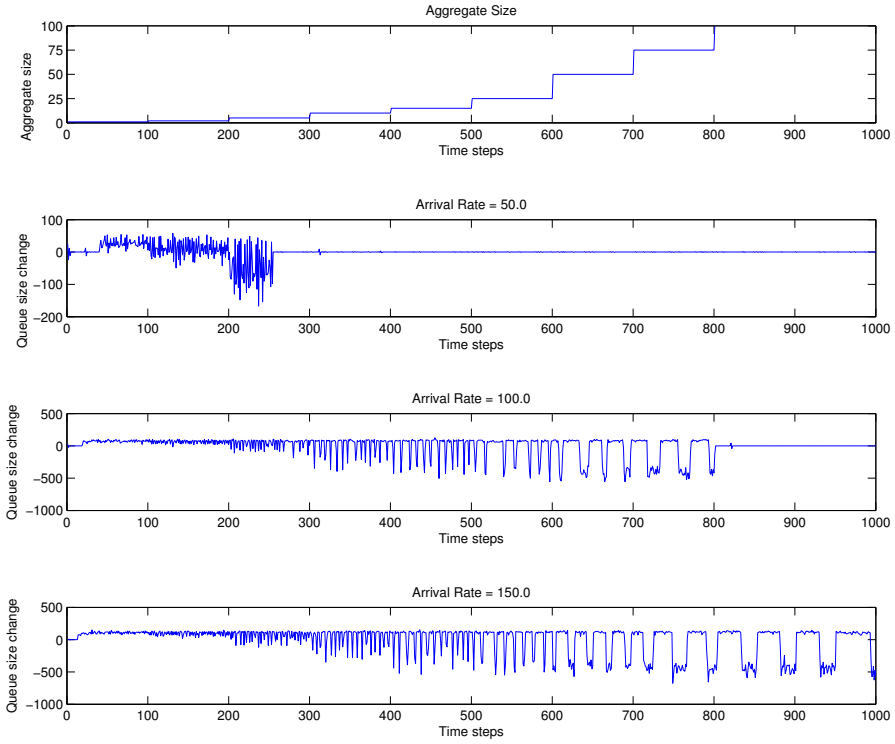


Figure 52: Static test: queue size changes

- Messages occur with an arrival rate of 150.

Figure 53 shows the result of the step test:

- With an aggregation size of 1, the system is not able to handle the load. The queue length is costantely increasing.
- When the aggregation size is set to 50 at timestep 100, the queue size is directly decreased, without a noticable delay.

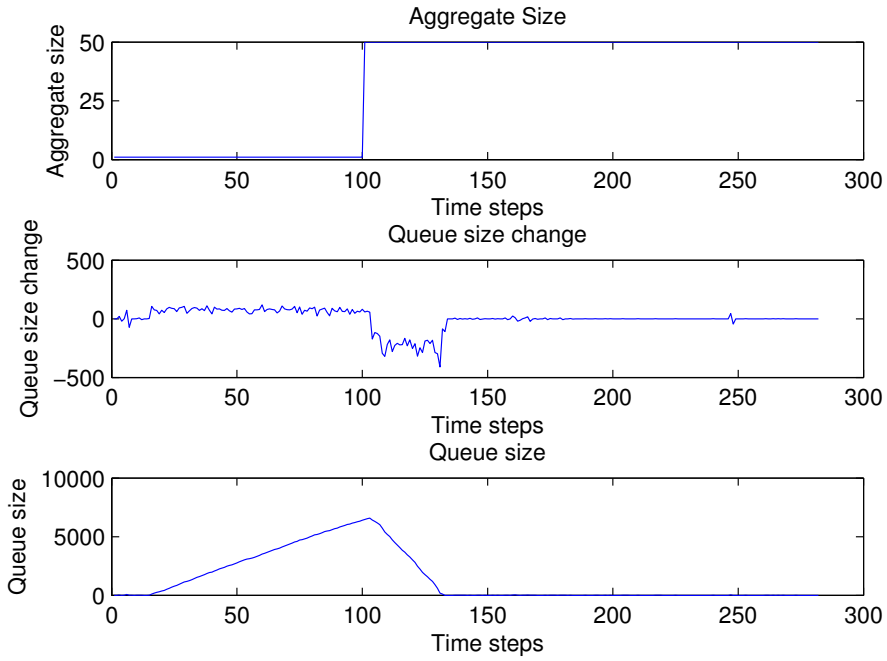


Figure 53: Step test

5.7.5 *Controller Tests*

The following test has been performed to evaluate the performance of the *Simple Controller* and the *PID-Controller*:

- Events are generated with an $\mathtt{arrival\ rate} = 50.0$ for 100 timesteps.
- At $\mathtt{timestep} = 100$, the arrival rate is set to 150.0 for another 100 timesteps.
- At $\mathtt{timestep} = 200$, the arrival rate is set back to 50.0.

Figure 54 shows the results of this test using a proportional controller. The *PID-Controller* has been configured with the following gains:

- $K_p = 1.0$, $K_i = 0.0$, $K_d = 0.0$

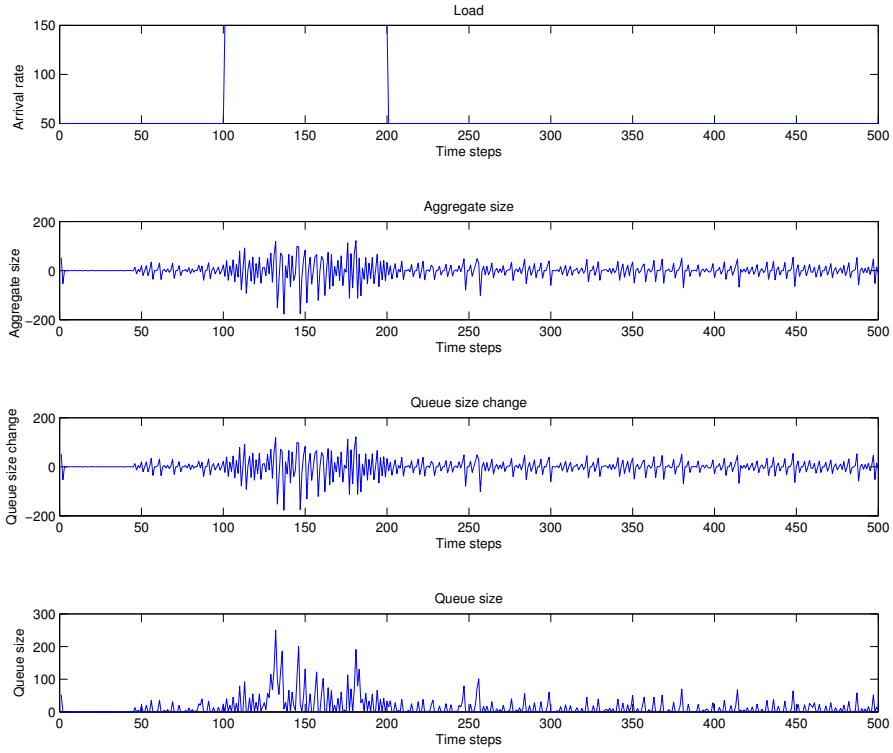


Figure 54: Proportional control

The controller is able to control the queue size but it leads to an significant oscillation of the aggregation size.

A better solution can achieved when using the *Simple Control* strategy. Figure 55 shows the results of the test using the *Simple Control* strategy:

- The controller is reasonably able to control the size of the queue. At $\mathtt{timestep} = 100$, it increases the aggregate size to a maximum value of 36.
- At $\mathtt{timestep} = 200$, the controller starts to decrease the aggregation size. At $\mathtt{timestep} = 375$, the aggregation size is back at 3.

5.7.6 *Results*

Summarizing the results of the evaluation, the proposed concept for the adaptive middleware is a viable solution to optimize the end-to-end latency of data processing system. The results show that using a closed-feedback loop is a feasible technique for implementing the dynamic control of the aggregation size. Using the queue size change to measure the system load is also shown to be appropriate.

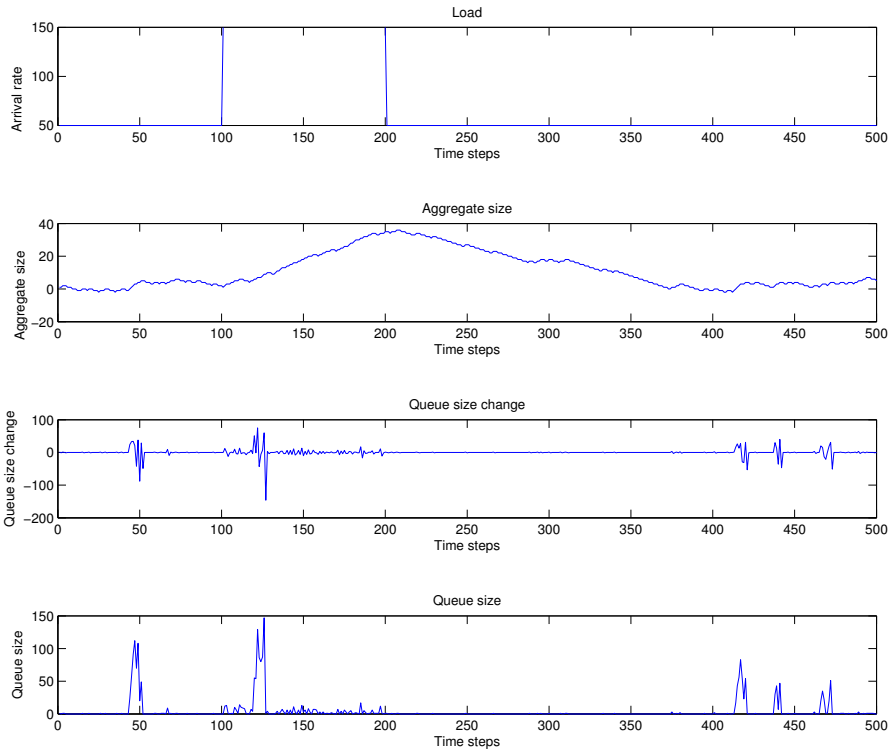


Figure 55: Simple control strategy

5.8 SUMMARY

In this chapter, a novel concept of middleware for near-time processing of bulk data has been presented 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 messsage, the middleware routes a message to appropriate service endpoints, which are optimized for either single-event or batch processing.

Additionally, several design aspects have been described that should be taken into account when designing and implementing an adaptive system for bulk data processing, such as how to design the service interfaces, the integration and transport mechanisms, the error-handling and controller design.

The solution is based on standard middleware, messaging technologies and standards and fully preserves the benefits of an SOA and messaging middlware, such as:

- Loose coupling
- Remote communication

- Platform language Integration
- Asynchronous communication
- Reliable Communication

To evaluate the proposed middleware concepts, a prototype system has been developed based on the message-based prototype described in Section 4. The tests show that the proposed middleware solution is viable and is able to optimize the end-to-end latency of a data processing system for different load scenarios.

During the implementation of the prototype of the adaptive middleware, it became apparent that the design and implementation of such a system differs from common approaches to implement enterprise software systems. The next chapter describes a conceptual framework that guides the design, implementation and operation of a system for bulk data processing based on the adaptive middleware.

6

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, development and operation 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 56 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.

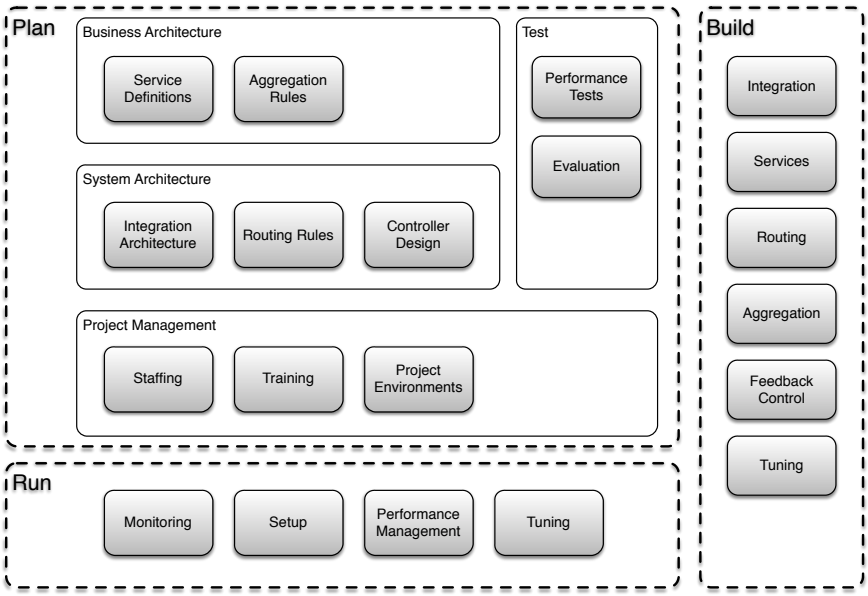


Figure 56: Overview of Conceptual Framework

The conceptual framework uses UML 2.0 notation elements and diagrams, such as class diagrams, activity diagrams and use-case diagrams to describe the developtment process. It consists of the following packages, as shown in Figure 57:

- **Metamodel**
  Contains elements and class-diagrams for describing the metamodel 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 framwork.
- **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.

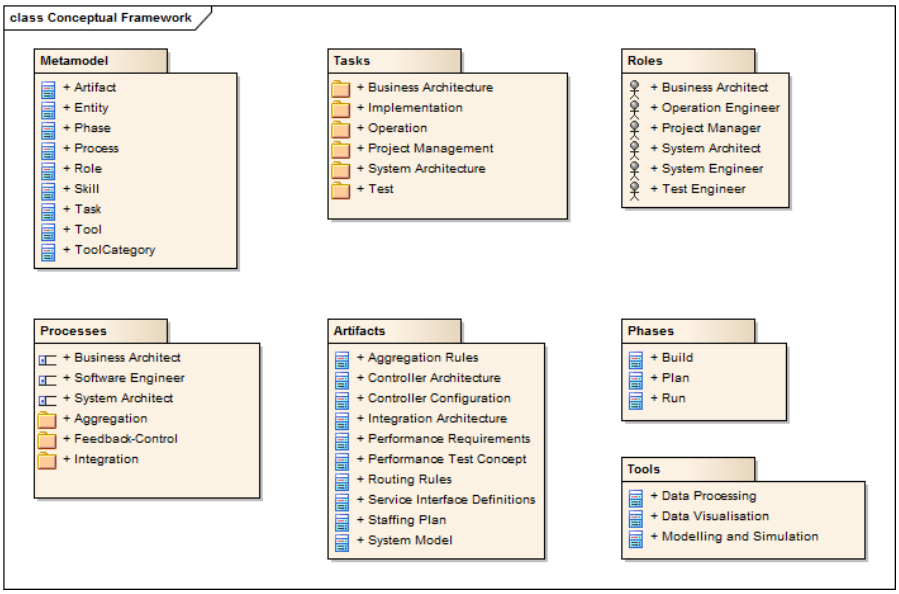


Figure 57: Package structure

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 softwar 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 58:

- **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.

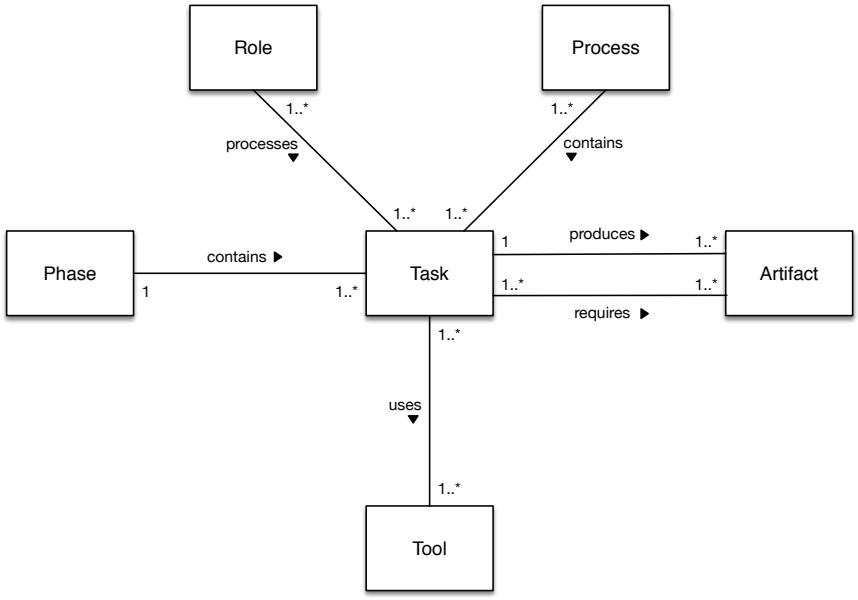


Figure 58: Metamodel

6.3 PHASE

Phases are the top-level entitis 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 59:

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

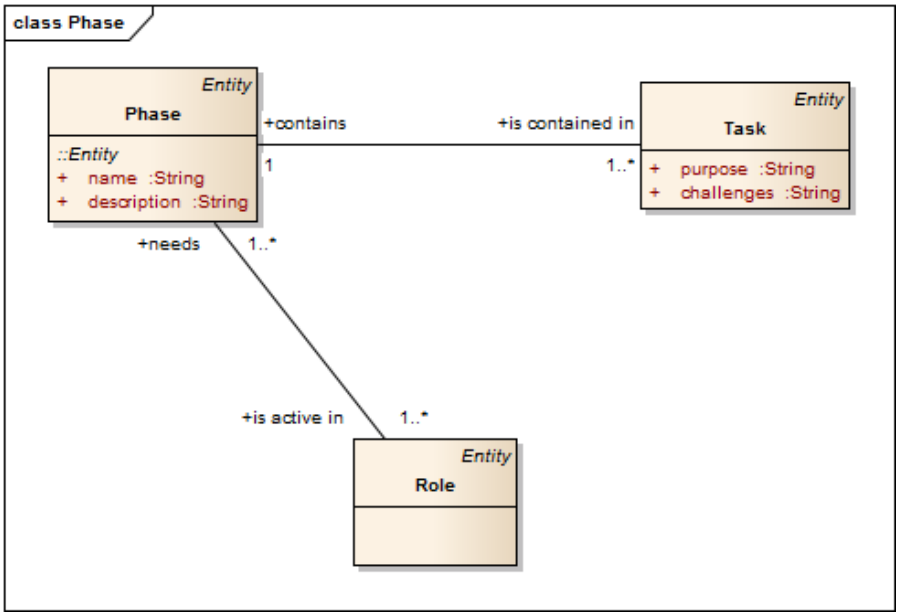


Figure 59: Attributes of a phase

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 theses 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	• 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

- 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 reponsibilities 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 60:

- **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 ist responsible for implementing the software system.

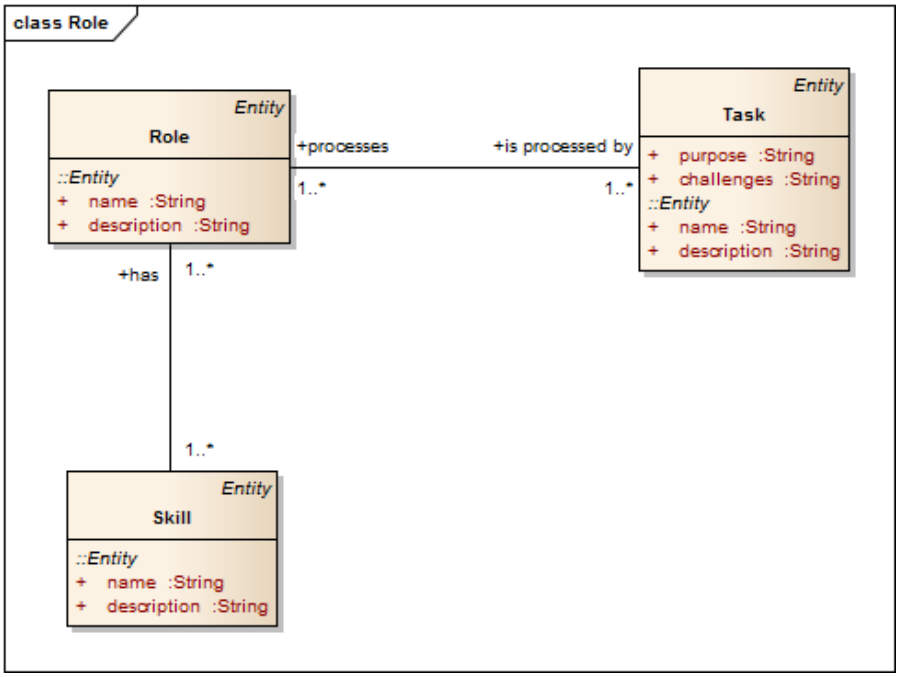


Figure 60: Attributes of a role

- **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*

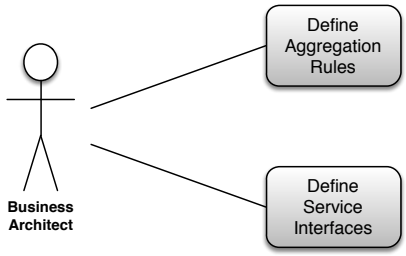


Figure 61: Role: Business Architect

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	• Define Service Interfaces • Define Aggregation Rules
Needed skills	• Integration styles and patterns, e.g. SOA • Concepts of the Adaptive Middleware for Bulk Data Processing • Business domain knowledge

6.4.2 *System Architect*

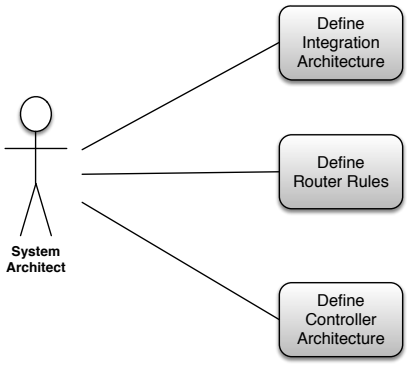


Figure 62: Role: System Architect

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	• Define Integration Architecture • Define Controller Architecture
Needed skills	• 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*

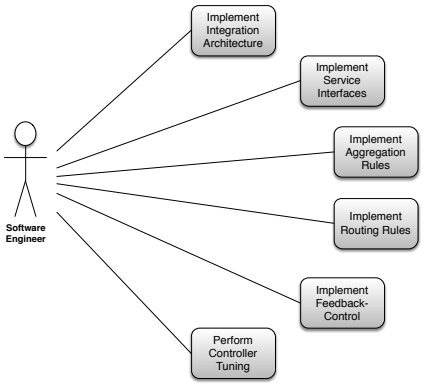


Figure 63: Role: Software Engineer

Table 17: Software Engineer

Role	Developer
Description	The Developer is responsible for the implementation of the system, including the implementation and tuning of the feeback-controll loop.
Tasks	• Implement Integration Architecture • Implement Service Interfaces • Implement Aggregation Rules • Implement Routing Rules • Implement Feedback-Control • Perform Controller Tuning
Needed skills	• 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

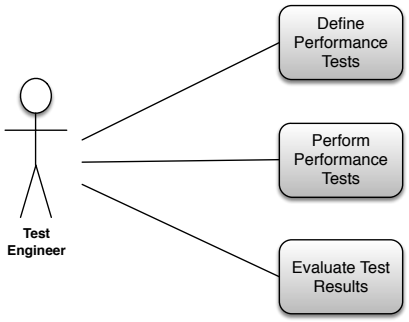


Figure 64: Role: Test Engineer

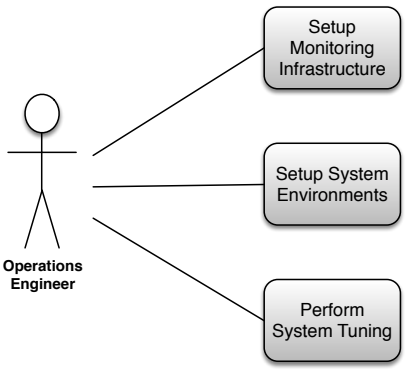


Figure 65: Role: Operations Engineer

6.4.6 *Project Manager*

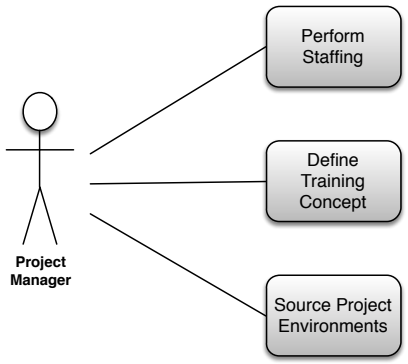


Figure 66: Role: Project Manager

Table 20: table Project Manager

Role	Project Manager
Description	The Project Manager is responsible for the project coordination, including the staffing and planing of the required environments.
Tasks	• Perform Staffing • Define Training Concept • Source Project Environments
Needed skills	• 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 67 shows an overview of the tasks grouped by the different phases of the Conceptual Framework.

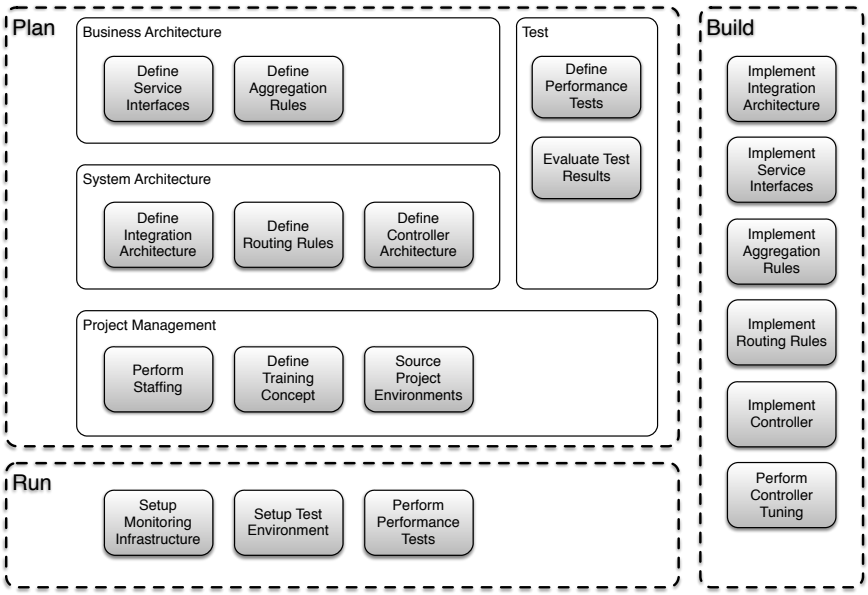


Figure 67: Overview of tasks

Tasks are organised in different packages, as shown in Figure 68:

- **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.

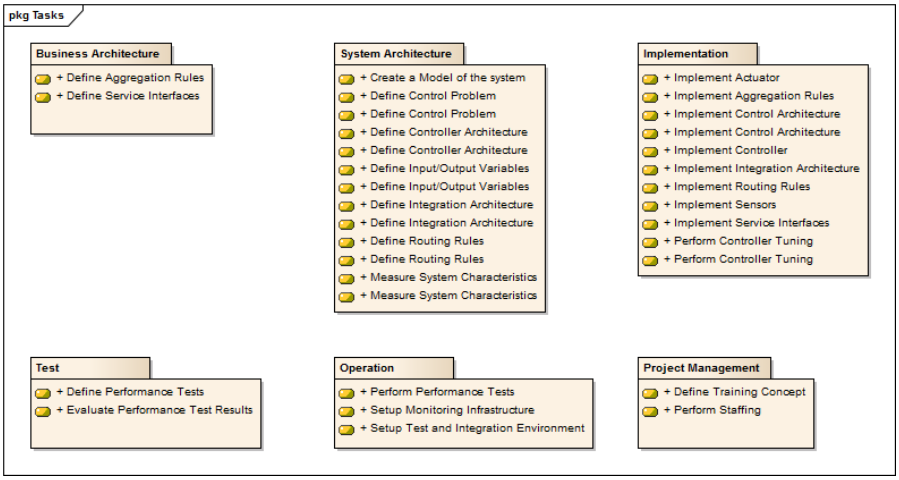


Figure 68: Subpackages of the Tasks package

A Task is described by the following attributes, as shown in Figure 69:

- **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 artificats 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

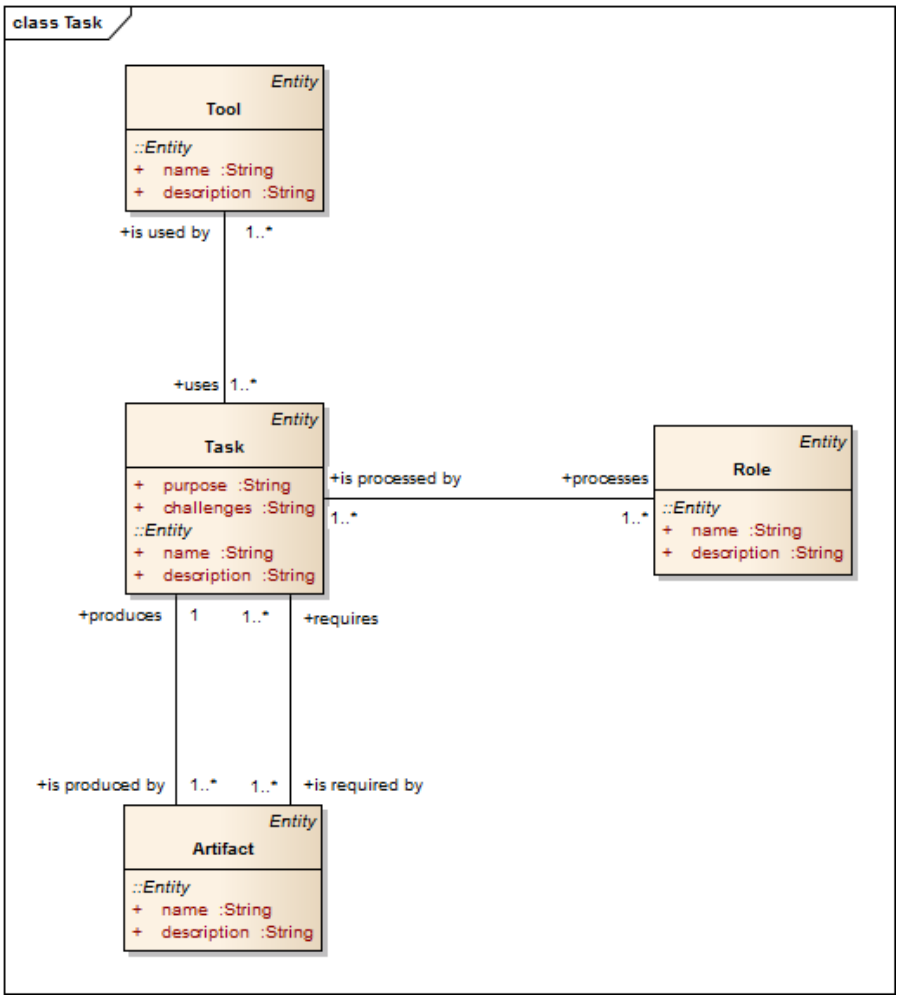


Figure 69: Attributes of a task

  - 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.

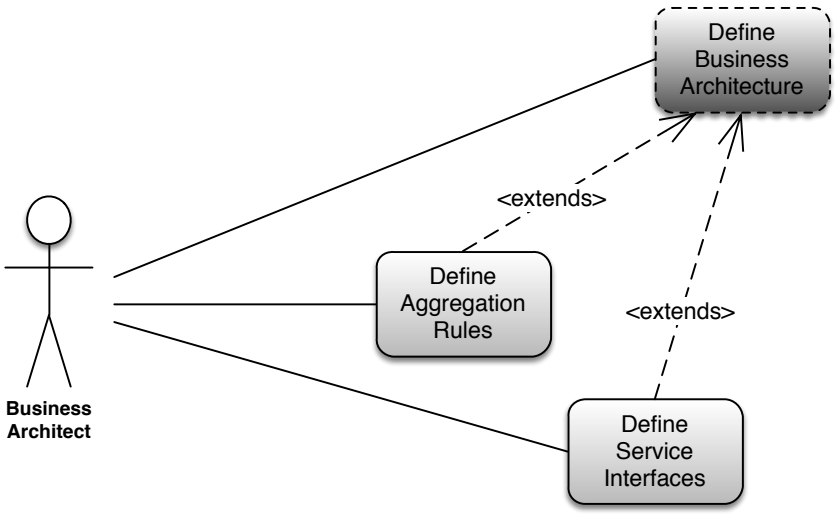


Figure 70: Tasks extending the definition of the business architecture

6.5.1.1 *Define Performance Requirements*

Table 21: Define Performance Requirements

Task	Define Performance Requirements
What	This task is concerned with the definition of the performance requirements of the system, including • Definition of workload scenarios • Definition of the adaptive features of the system • Definition of requirements regarding throughput and latency of the system – 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	• Busines Analyst • System Architect

Output	Performance Requirements
Challenges	The performance requirements must be explicitely 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	This task is concerned with the definition of the service interfaces, that together implement the business functionality of the system, including: • 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 5.5.1 for details). – 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	• 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	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): • **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	• Business Architect • System Architect
Output	Aggregation Rules
Challenges	• 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.

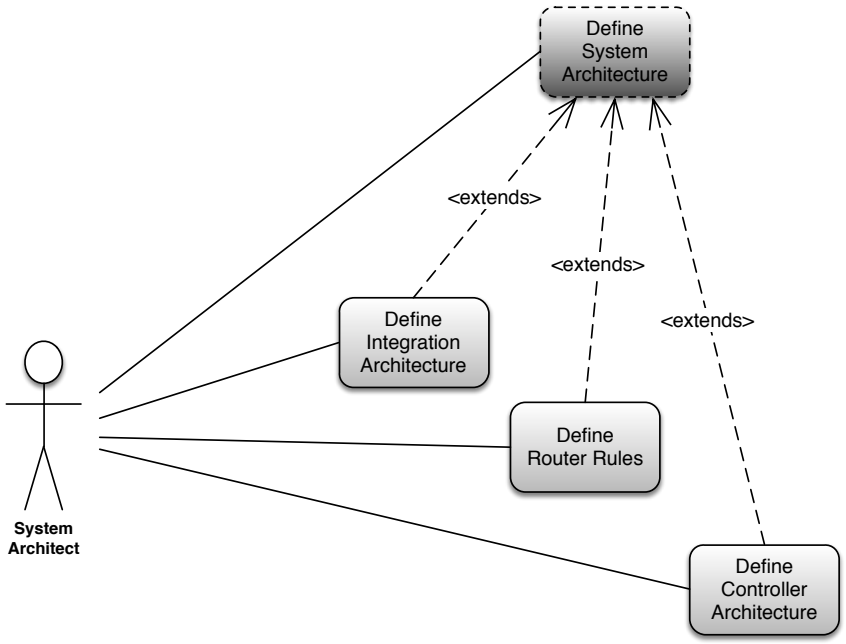


Figure 71: Tasks extending the definition of the system architecture

6.5.2.1 *Define Integration Architecture*

Table 24: Define Integration Architecture

Task	Define Integration Architecture

What	This task is concerned with the definition of the integration architecture of the system, including • Definition of communication styles, such as – Sychronous communication – Asynchronous communication • Choosing a middleware technology or product • Definition of transports, for example – JMS – SOAP – REST – FTP – DB • Different transports and integration patterns need to considered for different aggregation sizes (see Section 5.5.2 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	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 tasks includes • 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	This task is concerned with the definition of the controller architecture, including • Defining the controller type, for example – PID Controller – Fuzzy Controller • Defining sensors and actuators and their distribution architecture • Defining filters and additional components of the control-loop. • Depends on the *Control Problem* and the system dynamics (linear, non-linear).
Why	The *Controller Architecture* is the basis for the implementation of the feedback-control loop to control the message aggregation size at run-time
Who	*System Architect*
Input	• *Integration Architecture* • *Control Problem*

Output	*Controller Architecture*
Challenges	Finding the right *Controller Architecture* 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 *Controller Architecture* beforehand.

6.5.2.4 *Define Control Problem*

Table 27: Define Control Problem

Task	Define Control Problem
What	This task is concerned with the definition of the *Control Problem*, including • 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 *Control Problem* defines the goal of the feedback-control.
Who	*System Architect*
Output	*Control Problem*
Challenges	The *Control Problem* is not in all cases obvious and needs to be derived from the *Performance Requirements* of the system.

6.5.2.5 *Define Input/Output Variables*

Table 28: Define Input/Output Variables

Task	Define Input/Output Variables

What	This task is concerned with the definition of the input and output variables used by the controller, for example • Number of messages in the system • Input queue length • Current end-to-end latency • Current throughput
Why	The *Input/Output Variables* are needed for the implementation of the controller.
Who	*System Architect*
Input	*Control Problem*
Output	*Input/Output Variables*
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	This task is concerned with the implementation of the *Controller Architecture*, including • 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	*Software Engineer*
Input	*Controller Architecture*
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 5.7.3 for details.
Why	The static behaviour of the system is needed to determine the characteristics of the system.
Who	*System Architect*
Output	*Static Test Results*
Tools	• 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.
Why	The dynamic behaviour of the system is needed for building a model of the system and to tune the controller.
Who	*System Architect*
Output	*Step Test Results*
Tools	• 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	*System Architect*
Input	Static and dynamic behaviour of the system
Output	*System Model*
Tools	Tools for system modelling and system identification

Challenges	The *Software Engineer* needs to have a profound knowlegde 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	This task is concerned with the tuning of the implemented controller. • 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 *Controller Architecture*.
Why	The Controller needs to be adjusted to the system characteristics.
Who	*Software Engineer*
Input	*Controller Architecture*
Output	*Controller Configuration*
Tools	Tools for Simulation
Challenges	The software engineer needs to have a profound knowlegde 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	This task is concerned with the implementation of the business services, including • Implementation of batch operations • Implementation of single-event operations

Why	The services implement the business functionality of the system.
Who	*Software Engineer*
Input	*Service Interface Definitions*
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	This task is concerned with the implementation of the message aggregation, including • Implementation and configuration of the Aggregator component. • Implementation of the aggregation rules. The *Aggregation Rules* should be configurable during run-time or configuration-time and should not be hard-coded.
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	*Software Engineer*
Input	*Aggregation Rules*
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	This task is concerned with the implementation of the message routing, including • Implementation and configuration of the Router component. • Implementation of the routing rules. The *Routing Rules* should be configurable during run-time or configuration-time and should not be hard-coded.
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 *Adaptive Middleware* (see Section 5.3.2 for details).
Who	*Software Engineer*
Input	*Routing Rules*
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	This task is concerned with the definition of the performance tests, including • 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	*Test Engineer*
Output	*Performance Test Concept*
Tools	• 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 *Define Performance Tests*.
Why	The performance test evaluation is conducted to understand the performance characteristics of the system.
Who	• *Test Engineer* • *System Engineer*
Input	*Performance Test Result*
Output	*Performance Test Evaluation*
Tools	• 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 infrastructuree
What	Setting up the monitoring infrastructure, including • 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	*Operations Engineer*
Input	*System Architecture*
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 • 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	*Operations Engineer*
Input	*Performance Test Concept*

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 *Define Performance Tests*.
Why	The performance tests are necessary to assure that the system meets the performance requirements.
Who	*Test Engineer*
Input	*Performance Test Concept*
Output	*Performance Test Results*
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 ressources 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 • Definition of target audience, for example *Operation Engineers*, *System Engineers* • Definition of training content, for example – 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	• The operation engineers need to have the knowlegde to operate and tune the system in production. • Additionally, the team members also need to have the knowlegde to design and implement the system.
Who	*System Architect*
Input	• *Business Architecture* • *System Architecture* • *Controller Architecture*
Output	*Training Concept*
Challenges	The *Training Concept* should consider the respective audience and its existing knowlegde.

6.5.6.2 *Perform Staffing*

Table 43: Staffing

Task	Perform Staffing

What	This task is concerned with the staffing of the project. There are special skills needed for staffing the project, for example • Know how about the adaptive middleware concepts as introduced in Chapter 5. • Controller design, implementation, and tuning
Why	The *Staffing Plan* is needed to get the appropriate team members with the needed skills.
Who	*Project Manager*
Output	*Staffing Plan*
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 72

Figure 73 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 74

Figure 75 shows the UML activity diagram of the process.

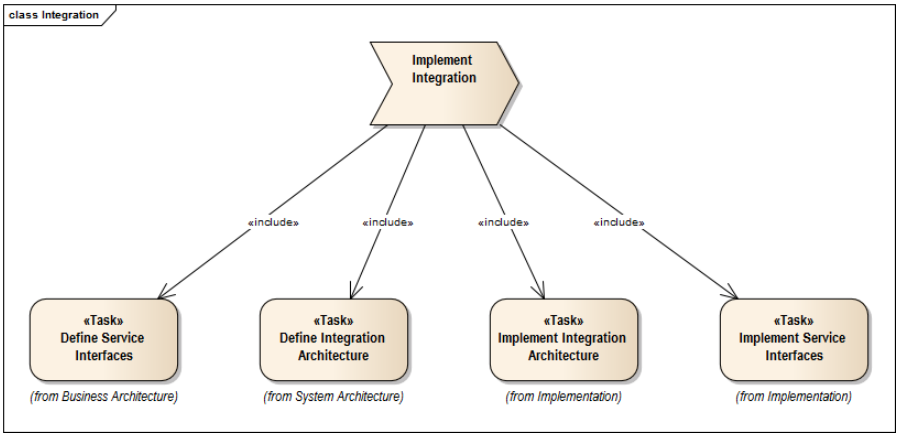


Figure 72: Tasks of the process Implement Integration

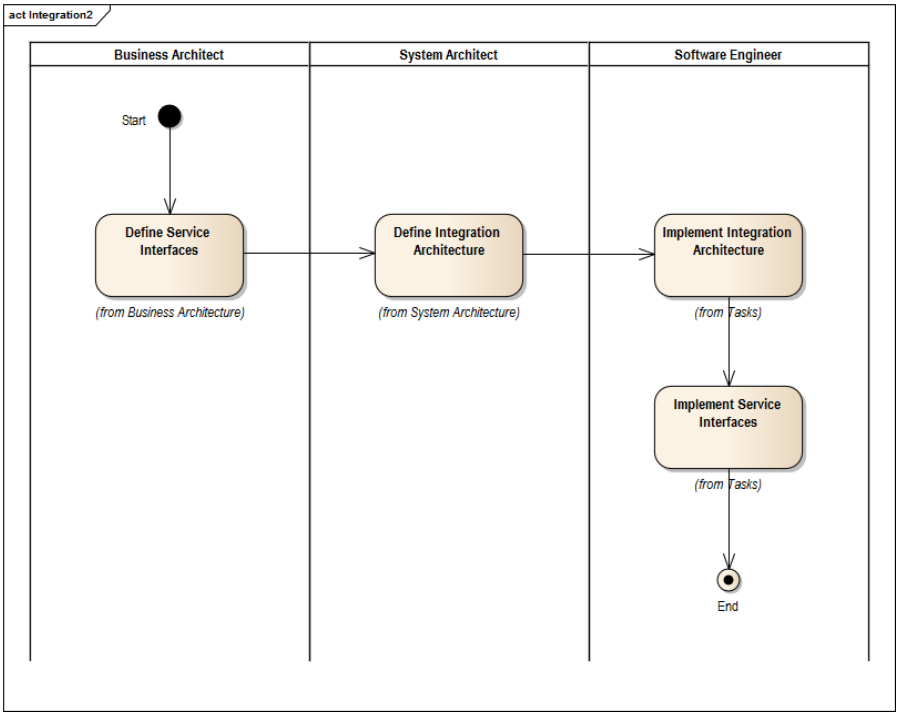


Figure 73: UML Activity Diagram: Implement Integration

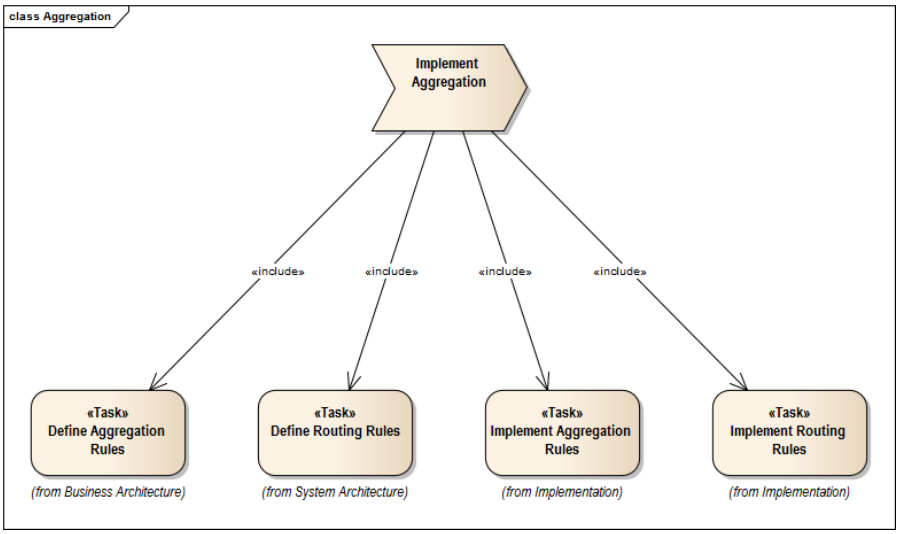


Figure 74: Tasks of the process Implement Aggregation

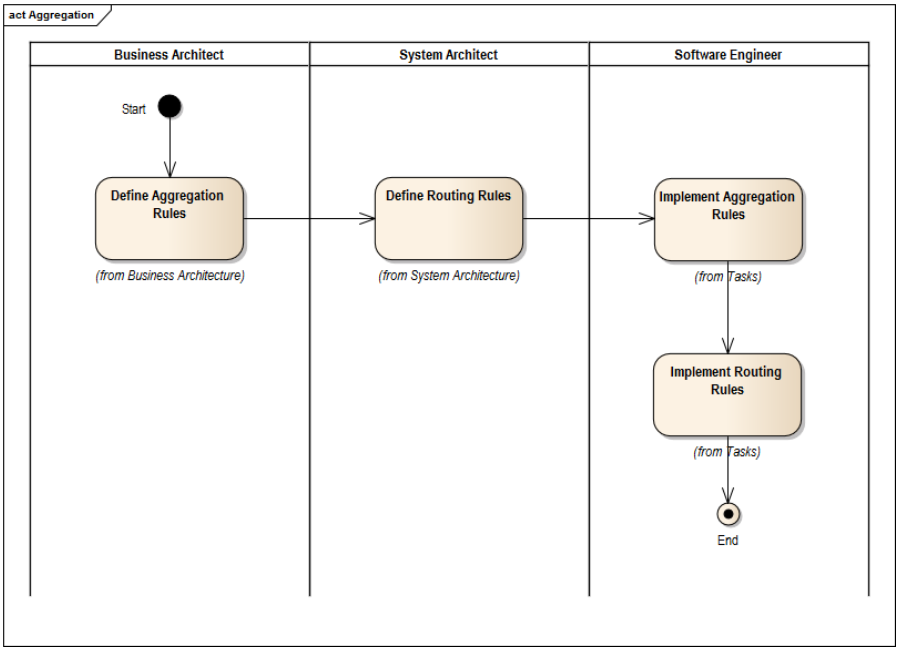


Figure 75: UML Activity Diagram: Implement Aggregation

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 76.

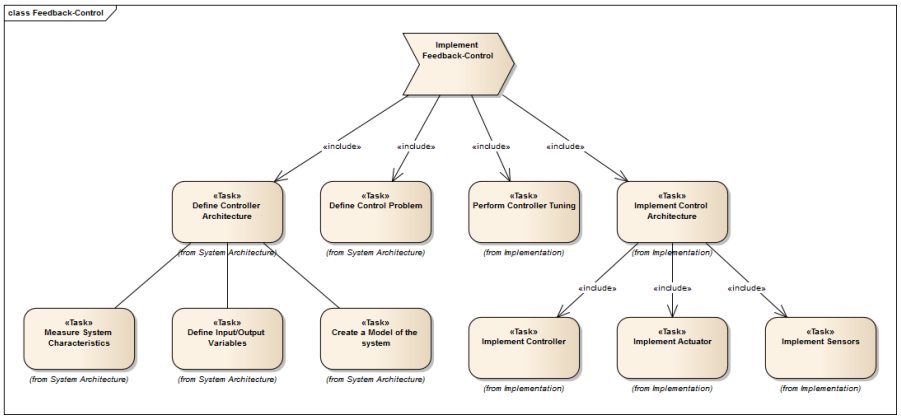


Figure 76: Tasks for implementing the feedback-control loop

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 77.
- 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 78.

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 69:

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

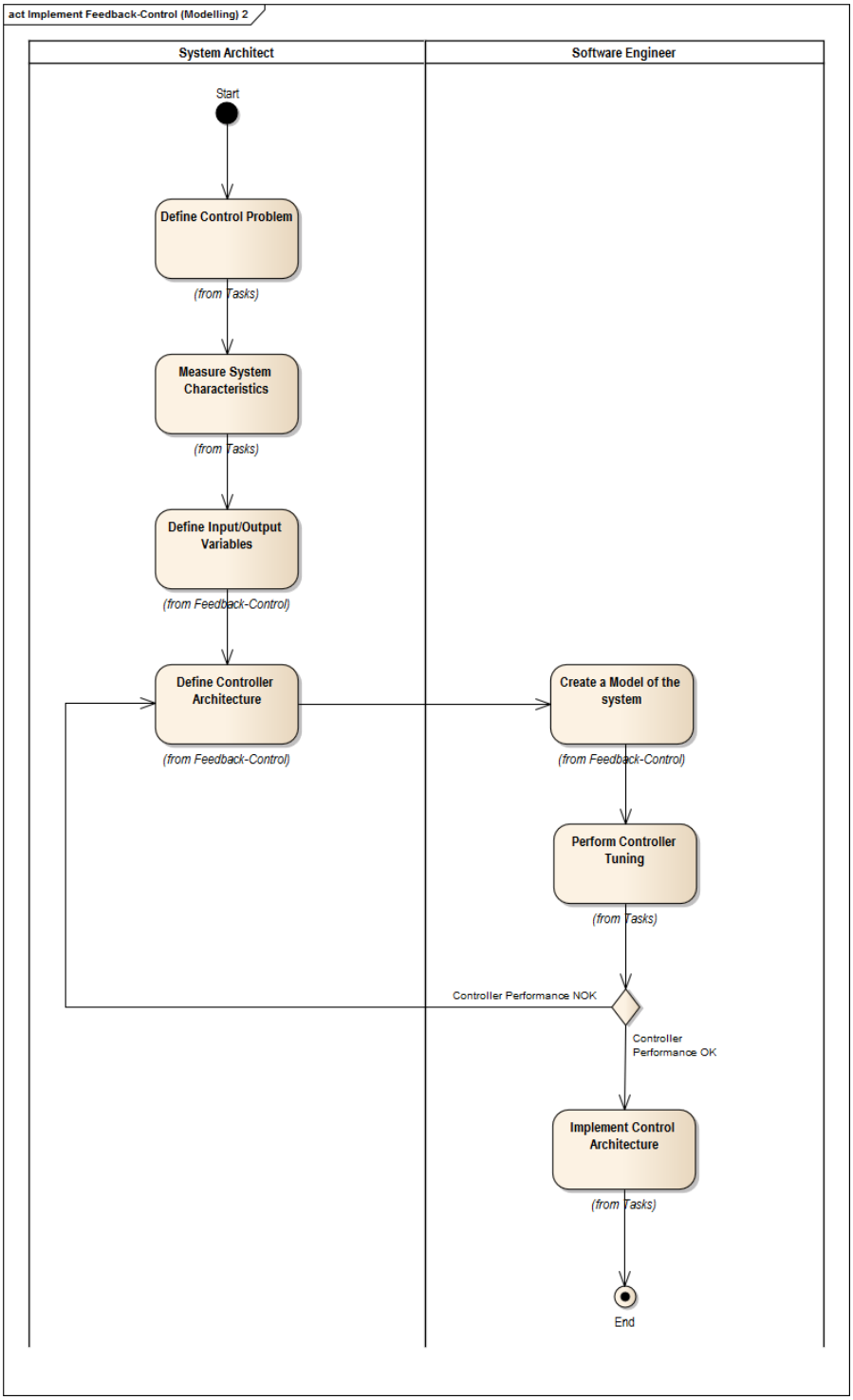


Figure 77: UML Activity Diagram: Implement Feedback-Control Loop using a model

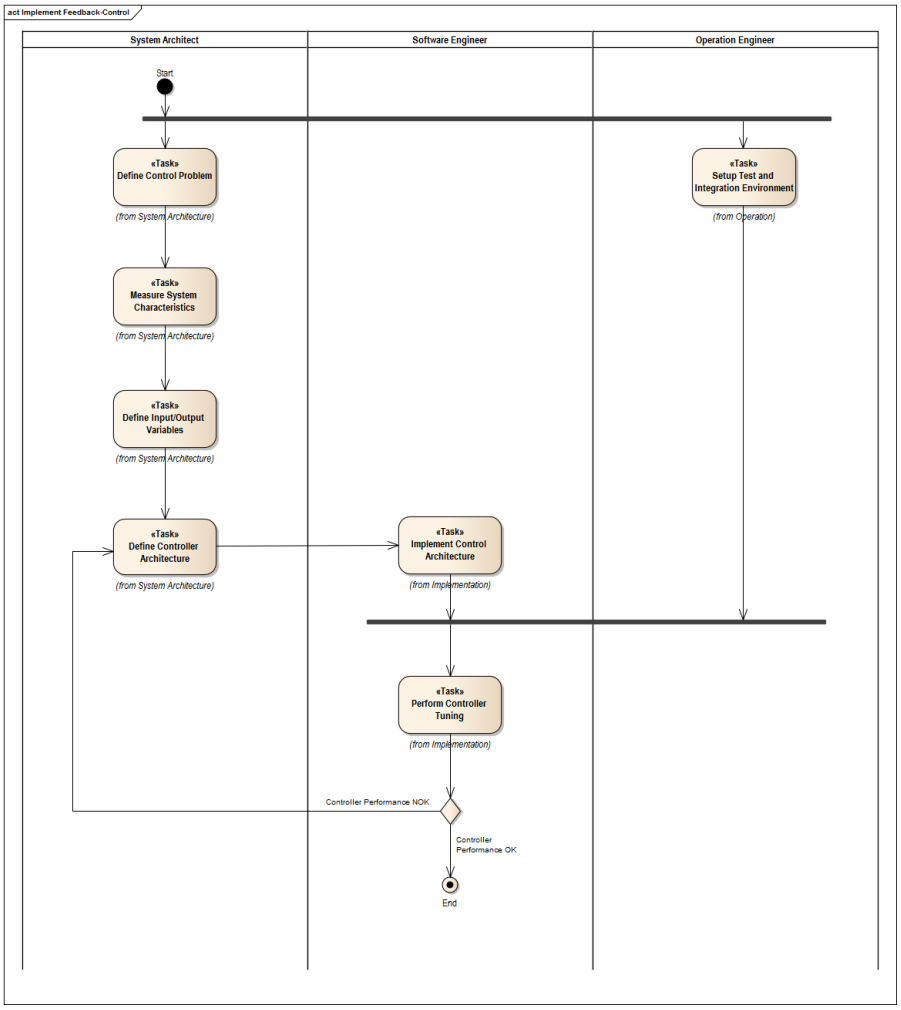


Figure 78: UML Activity Diagram: Implement Feedback-Control Loop without using a model

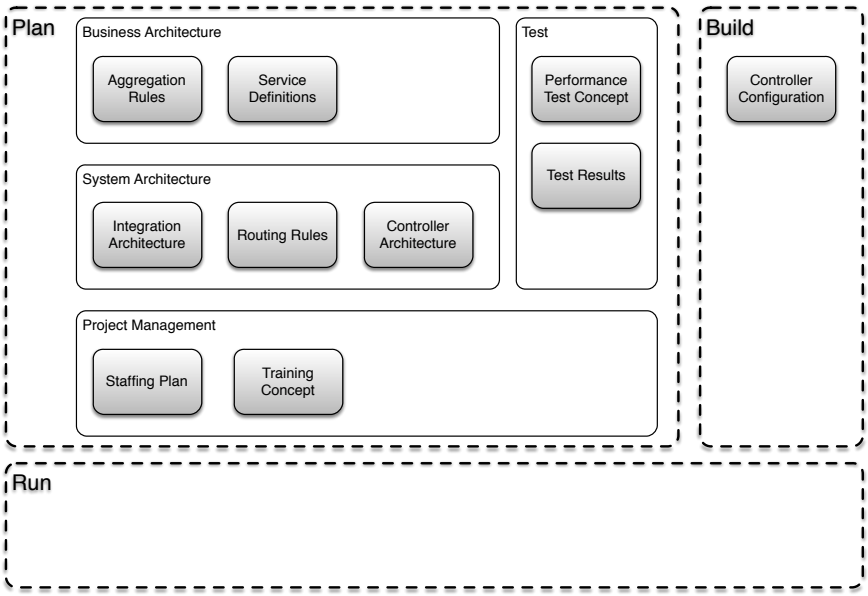


Figure 79: Artifacts

- **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	• 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	• Business Architect • System Architect

6.7.2 *Service Interface Definition*

Table 45: Service Interface Definition

Artifact	Service Interface Definition
Description	• 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	Defines the technical integration of the business services, including • 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	*Define Routing Rules*
Role	*System Architect*

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	*Perform Controller Tuning*
Role	*Sofware Engineer*

6.7.8 *Training Concept*

Table 51: Training Concept

Artifact	Training Concept
Description	• 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	*Define Training Concept*

Role

- *Project Manager*
- *System Architect*

6.7.9 *Staffing Plan*

Table 52: Training Concept

Artifact	Staffing Plan
Description	The staffing plan contains • 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	*Perform Staffing*
Role	*Project Manager*

6.8 TOOLS

The design and implementation of the adaptive middleware requires the use of some specific tools. A tool is decribed by the following attributes, as shown in Figure 81:

- **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

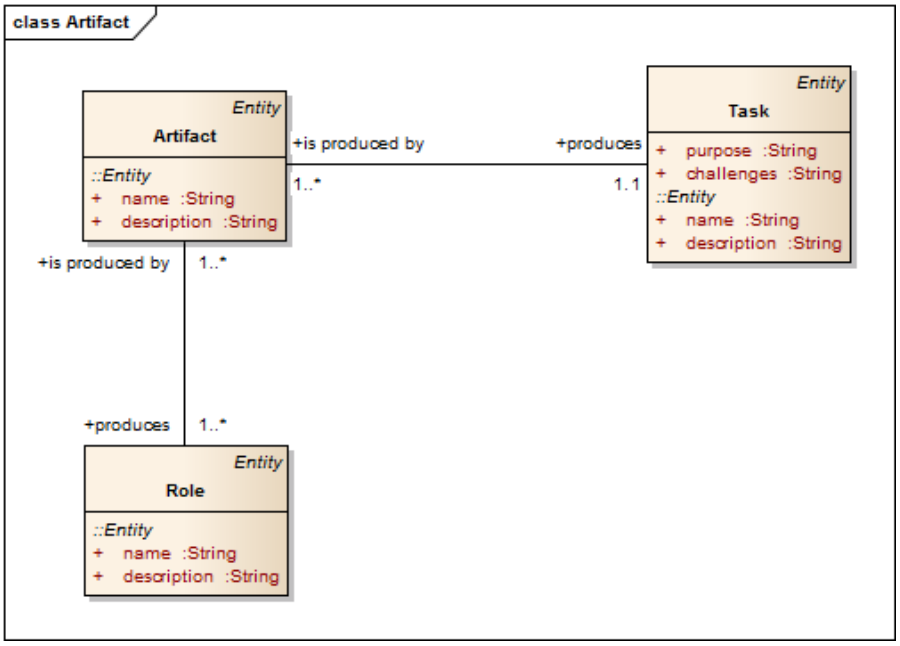


Figure 80: Attributes of an artifact

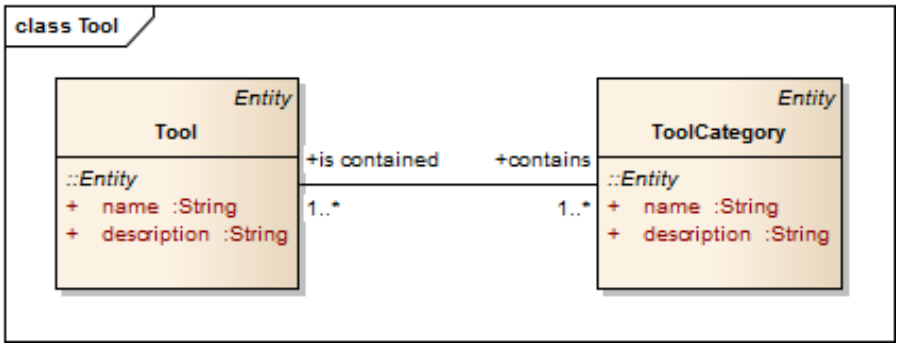


Figure 81: Attributes of a tool

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

The implementation of the feedback-controll 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 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 asssigning 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 artificats produced by the software development process.
  - **Environment**
    Provisioning the software development organisation with the software development environment.

Figure 82 shows the core workflows of the RUP and when they are conduction during the different phases.

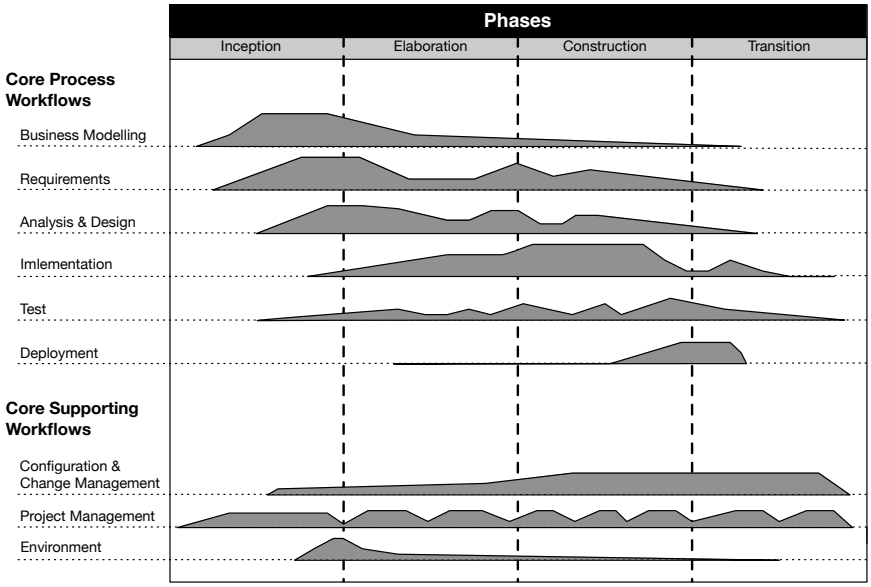


Figure 82: Core process workflows (Kruchten and Royce, 1996)

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	• Define Service Interfaces • Define Aggregation Rules
Requirements	• Define Performance Requirements
Analysis & Design	• Define Integration Architeture • Define Routing Rules • Define Controller Architecture
Implementation	• 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 constantely 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 adapative 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) oder 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 Derniame (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 prescripte 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 previousley performed by software engineers are now performed by the system itself. In a way, the role of the human software engineer is to some extend 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 explicitely reflected in software process models, since they are not independet from off-line activities. In addition to on-line activities, on-line roles and work products also need to be adressed 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 constrast 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 coresponding 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 artificats. 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 occurences of the inner adaptive process. It consists of the activities *Aggregate Metrics*, *Analyze*, *Evolve Adaption Rules*, *Adjust and Synchronize*, and *Reflect*. Theses 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 facilates 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

7 CONCLUSION

This chapter concludes the thesis by summarizing the achievements of the research and discussing its limitations. Additionally, it presents possible directions for further research.

7.1 ACHIEVEMENTS OF THE RESEARCH

This research project is aimed to optimize the end-to-end latency of a system for bulk data processing.

The first objective of the research was to analyze the relationship of end-to-end latency and throughput of batch and message-based systems. A formal definition of the relationship of end-to-end latency and maximum throughput has been given in Section 2.3. To analyze the impact of different processing styles, that is batch and message-based processing, on throughput and latency, two prototypes of a billing system for each processing type have been built. A performance evaluation has been conducted to compare the prototypes with each other with the focus on throughput and latency. The evaluation showed 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.

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).

The results presented in Section 4.4 show that throughput and latency depend on the granularity of data that is being processed. Another finding is, 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 cause by a congestion in the aggregator.

Based on the results of the performance evaluation of the batch and message-based prototype, the concept of an adaptive middleware for near-time processing of bulk data has been developed (see Section 5). The adaptive middleware is able to adapt its processing type fluently between batch processing and single-event processing. By using message aggregation, message routing and a closed feedback-loop to adjust the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios. The concept also describes several design aspects that should be taken into account when designing and implementing an adaptive system for bulk data processing, such as how to design the service interfaces, the integration and transport mechanisms, the error-handling and controller design.

The message-based prototype has been extended to implement the concepts of the adaptive middleware, which is described in Section 5.6. Using this prototype, a performance evaluation has been conducted to evaluate the proposed concepts of the adaptive middleware for bulk data processing (see Section 5.7). The results show that the concept is generally viable and is able to minimize the end-to-end latency of a system for bulk data processing.

During the implementation of the prototype of the adaptive middleware, it became apparent that the design and implementation of such a system differs from common approaches to implement enterprise software systems. In order to guide the implementation of an adaptive system for bulk data processing, a conceptual framework has been developed (see Section 6). It defines artifacts, roles, tasks and their dependencies, and processes to describe the necessary steps for design, implementation and operation of such a system, including:

- 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.

Additionally, it has been described in Section 6.9 how the conceptual framework can be used with common software development methodologies.

Several aspects of the results achieved in this research project have been presented at refereed conferences and have received positive comments from reviewers and delegates.

7.2 LIMITATIONS

Despite having met the objectives of this research project, this research has some limitations, that are summarized below:

- The services that implement the business functionality of the system need to be explicitely designed to support the run-time adaption between single-event and batch processing, as described in Section 5.5.1. Therefore, existing services need to be changed in order to be integrated into the system. This can pose a problem when using off-the-shelf services or **SaaS!** (SaaS!). The integration of such services has not been considered in this research.

- The services integrated by the prototype do not implement any further optimizations for batch processing. They use the same implementation for batch and single-event processing. Thus, the impact of batch optimizations has not been investigated. This was not necessary to show the performance improvements of message aggregation on the maximum throughput of the messaging prototype.

- The adaption mechanisms of the *Adaptive Middleware* only uses message aggregation and message routing, depending on the aggregation size. Other mechanisms such as dynamic service composition and selection and load balancing have not been investigated.

- The prototype of the *Adaptive Middleware* only uses a single message queue, the integrated services are called synchronous, using a request/response pattern. This design was chosen, to simplify the dynamics of the system. Thus, the impact of using multiple message queues has been investigated in the evaluation.

- The impact of different controller architectures has not been exhaustivley analysed and researched. Only two controller architectures have been implemented and evaluated. Other controller designs, such as fuzzy control, have not been invistigated. Additionally, a formal analyzation of the feedback-control system has not been conducted, for example by creating a model of the system. Instead, an empirical approach has been taken to evaluate the viability of the proposed solution.

- The *Conceptual Framework* has not been validated, for example by qualitative research methods, such as expert interviews, or applied with real-life projects.

Despite these limitations, the research project has made valid contributions to knowledge and provided sufficient proof of concept for the proposed approaches.

7.3 FUTURE WORK

The research project has advanced the field of systems for bulk data processing. However, a number of areas for future work can be identified that build on the achieved results.

- The adaptive middleware uses a single aggregator, that aggregates messages after they have been read from the input message queue of the system. The aggregator is controlled by a closed feedback-loop that controls the aggregation size based on the current load of the system. It could be interesting to investigate how this approach scales for a system consisting of multiple sub-systems, each sub-system consisting of an input message queue and an aggregator. These aggregators need to be a combination of an aggegrator and splitter to decrease the aggegration size of messages that are aggregrated by preceding sub-systems. A question of interest is the type of control strategy that is needed for this kind of systems. Does a decentralized control strategy work, with every subsystem having its own independant control-loop or is central approach necessary?

- The aggregator used by the adaptive middleware uses static correlation rules to aggregate messages. Depending on the type of input data, it could be necessary to adapt these rules at runtime. For example to change the correlation rule from a simple correlation, where messages are aggregated in the order in which they occur to a more complex correlation rule based on business rules. It is thinkable that this adaption can be automatically performed by the system without manual changes.

- The proposed adaptive middleware uses dynamic message aggregation to optimize the end-to-end latency of a data processing system. The concept could be extended to use other adaption mechanisms, such as dynamic service composition and selection and load balancing. Additionally, further performance optimization techniques such as caching or dynamic scaling could be investigated by further research.

- The concept for the adaptive middleware has only been evaluated using a prototype using simulated data. No experience has been made so far using this approach in real-life projects. It could be of interest how this approach is transferrable to real enterprise systems.

A

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_prototype

A.1 PROJECT STRUCTURE

The source code is organized in the following modules:

- **BatchProcessor**
  Contains the implementation of the batch prototype based on Spring Batch.
- **BillingRouter**
  Contains the implementation of the messaging prototype based on Apache Camel.
- **CamelUtils**
  Contains common Apache Camel specific logging utilities
- **DataGenerator**
  Contains the load generator.
- **FunctionalCore**
  Contains the common business functionality of the system used by batch and messaging prototypes.
- **MediationBatchRoute**
  Contains the wrapper to integrate the mediation processor of the batch prototype with Apache Camel.
- **MediationService**
  Contains the webservice wrapper of mediation processor used by the messaging prototpye.
- **PerformanceMonitor**
  Contains the Feedback-Control framework to monitor, measure and control the messaging system.
- **RatingBatchRoute**
  Contains the wrapper to integrate the rating processor of the batch prototype with Apache Camel.

- **RatingService**
  Contains the webservice wrapper of the rating processor used by the messaging system.

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 42 and 103.)

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 page 42.)

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 33.)

Acuña, S. T. and Ferré, X. (2001a). Software process modelling., *ISAS-SCI (1)*, pp. 237–242. (Cited on pages 166, 167, and 168.)

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 167.)

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 57.)

Amazon EC2 Auto Scaling (n.d.). http://aws.amazon.com/autoscaling. [retrieved: March 2014]. (Cited on page 29.)

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 45.)

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 170.)

Andersson, J., de Lemos, R., Malek, S. and Weyns, D. (2009). Reflecting on self-adaptive software systems., *SEAMS* pp. 38–47. (Cited on pages xi, 36, 37, and 38.)

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 33.)

Apache Camel (2014). http://camel.apache.org. [retrieved: July 2014]. (Cited on pages 58 and 60.)

Apache Hadoop (2014). http://hadoop.apache.org. [retrieved: December 2014]. (Cited on page 14.)

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 29.)

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 41, 47, and 85.)

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 76.)

Bartoli, A., Calabrese, C., Prica, M., Di Muro, E. A. and Montresor, A. (2003). Adaptive Message Packing for Group Communication Systems, pp. 912–925. (Cited on page 35.)

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 34.)

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 167.)

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 167 and 168.)

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 168.)

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 34.)

Blair, G., Bencomo, N. and France, R. (2009). Models@ run.time, *Computer* **42**(10): 22–27. (Cited on page 170.)

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement., *IEEE Computer ()* **21**(5): 61–72. (Cited on pages 160 and 167.)

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance., *Workshop on Software and Performance* pp. 195–203. (Cited on page 28.)

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 76.)

Brun, Y., Serugendo, G., Gacek, C. and Giese, H. (2009). Engineering self-adaptive systems through feedback loops, *... Self-Adaptive Systems* . (Cited on page 42.)

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 41.)

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 46.)

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 pages 41 and 47.)

Chappell, D. (2004). *Enterprise Service Bus*, O'Reilly Media, Inc., Sebastopol, CA, USA. (Cited on pages xiii, 19, 20, 21, and 41.)

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 page 78.)

Conrad, S., Hasselbring, W., Koschel, A. and Tritsch, R. (2006). *Enterprise Application Integration: Grundlagen, Konzepte, Entwurfsmuster, Praxisbeispiele*, Elsevier, Spektrum, Akad. Verl. (Cited on page 14.)

Cryderman, J. (2011). Is Real-Time Billing and Charging a Necessity?, **7**(11). (Cited on page 4.)

Curtis, B., Kellner, M. I. and Over, J. (1992). Process modeling, *Communications of the ACM* **35**(9): 75–90. (Cited on page 167.)

D'Ambrogio, A. (2005). A WSDL Extension for Performance-Enabled Description of Web Services, *Lecture notes in computer science* **3733**: 371. (Cited on page 77.)

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 76.)

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 170.)

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 33.)

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 35.)

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 169.)

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 45.)

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 38.)

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 25.)

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 33.)

EXI Working Group [online]. 2007. Available from: http://www.w3.org/XML/EXI [cited January 2008]. (Cited on page 27.)

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 167.)

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 35.)

Friedman, R. and Renesse, R. V. (1997). Packing messages as a tool for boosting the performance of total ordering protocls, *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 35.)

Fuggetta, A. (2000). Software process: a roadmap., *ICSE - Future of SE Track* pp. 25–34. (Cited on pages 111, 166, 167, and 172.)

Gacek, C., Giese, H. and Hadar, E. (2008). Friends or foes?: a conceptual analysis of self-adaptation and it change management., *SEAMS* pp. 121–128. (Cited on page 171.)

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 34.)

Gat, E. (1998). Three-layer architectures, *in* D. Kortenkamp, R. P. Bonasso and R. Murphy (eds), *Artificial Intelligence and Mobile Robots*, MIT Press, Cambridge, MA, USA, pp. 195–210. Available from: http://dl.acm.org/citation.cfm?id=292092.292130. (Cited on page 36.)

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 39 and 47.)

Gnuplot (2014). http://gnuplot.info. [retrieved: November 2014]. (Cited on page 159.)

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 41.)

Guinea, S., Baresi, L., Spanoudakis, G. and Nano, O. (2009). Comprehensive Monitoring of BPEL Processes, *IEEE Internet Computing* **99**. (Cited on page 44.)

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 43.)

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 33 and 46.)

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 33.)

Haesen, R., Snoeck, M., Lemahieu, W. and Poelmans, S. (2008). On the definition of service granularity and its architectural impact, *in* Z. Bellahsène 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 27.)

Hellerstein, J. L. (2004). Challenges in control engineering of computing systems, *American Control Conference, 2004. Proceedings of the 2004*, pp. 1970–1979. (Cited on page 90.)

Hellerstein, J. L., Diao, Y., Parekh, S. and Tilbury, D. M. (2004). *Feedback Control of Computing Systems*, John Wiley & Sons. (Cited on pages xi, 42, 43, and 90.)

Henjes, R., Menth, M. and Zepfel, C. (2006). Throughput Performance of Java Messaging Services Using WebsphereMQ, *ICDCSW '06: Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*, IEEE Computer Society, Washington, DC, USA, p. 26. (Cited on page 78.)

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 page 77.)

Herbst, N. R., Kounev, S. and Reussner, R. (2013). Elasticity in Cloud Computing: What It Is, and What It Is Not., *ICAC* pp. 23–27. (Cited on page 29.)

Hess, A., Humm, B. and Voß, M. (2006). Regeln für serviceorientierte Architekturen hoher Qualität, *Informatik Spektrum* **29**(6): 395–411. (Cited on page 28.)

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, xiii, 14, 15, 21, 22, 23, 24, 58, 86, and 87.)

IBM Group (2005). *An architectural blueprint for autonomic computing*, IBM White paper. (Cited on pages xi, 36, 37, and 39.)

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 66.)

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 171.)

Inverardi, P. and Tivoli, M. (2008). The Future of Software: Adaptation and Dependability., *ISSSE* **5413**(Chapter 1): 1–31. (Cited on page 170.)

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 40 and 47.)

Janert, P. K. (2013). *Feedback Control for Computer Systems*, O'Reilly Media, Inc. (Cited on pages 91 and 92.)

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 pages 41 and 47.)

Josuttis, N. (2007). *SOA in practice*, O'Reilly, Sebastopol, CA, USA. (Cited on pages 25 and 28.)

Kazhamiakin, R., Benbernou, S. and Baresi, L. (2010). Adaptation of service-based systems, *Service research ...* . (Cited on page 40.)

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 40.)

Krafzig, D., Banke, K. and Slama, D. (2005). *Enterprise SOA*, Prentice Hall. (Cited on page 25.)

Kramer, J. and Magee, J. (2007). Self-Managed Systems: an Architectural Challenge., *FOSE* pp. 259–268. (Cited on pages xi, 36, and 37.)

Kruchten, P. and Royce, W. (1996). A rational development process, *CrossTalk* **9**(7): 11–16. (Cited on pages xiii, 160, and 161.)

Laddaga, R. and Robertson, P. (2008). Abstract Self Adaptive Software: A Position Paper. (Cited on page 35.)

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 40.)

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 39.)

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 76.)

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 76.)

MathWorks (2014). MATLAB, http://www.mathworks.com. [retrieved: November 2014]. (Cited on page 159.)

matplotlib (2012). http://matplotlib.org. [retrieved: November 2014]. (Cited on page 159.)

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 166 and 168.)

Menth, M., Henjes, R., Zepfel, C. and Gehrsitz, S. (2006). 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 78.)

Merz, N. and Warren, J. (2014). *Big Data - Principles and best practices of scalable realtime data systems*, Manning Publications. (Cited on pages 13 and 14.)

Microsoft Excel (2012). http://products.office.com/en-us/excel. [retrieved: November 2014]. (Cited on page 159.)

Nagle, J. (1984). Congestion control in ip/tcp internetworks, *SIGCOMM Comput. Commun. Rev.* **14**(4): 11–17. Available from: http://doi.acm.org/10.1145/1024908.1024910. (Cited on page 35.)

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 33.)

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 33.)

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 32.)

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 24 and 31.)

OMG (2008). Software Process Engineering Metamodel SPEM 2.0, *Technical Report ptc/08-04-01*, Object Management Group. (Cited on page 168.)

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 36.)

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 42, 43, and 44.)

PTP daemon (PTPd) (2013). http://ptpd.sourceforge.net. [retrieved: July 2014]. (Cited on page 66.)

Python (2013). https://www.python.org/. [retrieved: November 2014]]. (Cited on page 159.)

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 160.)

Richter, J.-P., Haller, H. and Schrey, P. (2005). Serviceorientierte Architektur, *Informatik Spektrum* **28**(5): 413–416. (Cited on page 19.)

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 35.)

Royce, W. W. (1987). Managing the Development of Large Software Systems: Concepts and Techniques., *ICSE* pp. 328–339. (Cited on pages 160 and 167.)

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 page 78.)

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 page 78.)

Salehie, M. and Tahvildari, L. (2009). Self-Adaptive Software: Landscape and Research Challenges, *ACM Trans. Auton. Adapt. Syst.* **4**(2): 1–42. (Cited on pages 35 and 36.)

Schulte, R. (2002). Predicts 2003: Enterprise Service Buses Emerge, Gartner. (Cited on page 19.)

Schwaber, K. and Sutherland, J. (2013). The Scrum Guide, http://www.scrumguides.org/scrum-guide.html. [retrieved: November 2014]. (Cited on pages 160 and 163.)

Scilab (2014). http://www.scilab.org. [retrieved: November 2014]. (Cited on page 159.)

SimPy (2014). https://pypi.python.org/pypi/simpy. [retrieved: November 2014]. (Cited on page 159.)

SOAP Specification [online]. 2007. Available from: `http://www.w3.org/TR/soap` [cited January 2008]. (Cited on page 24.)

Spring Batch (2013). http://static.springsource.org/spring-batch/. [retrieved: July 2014]. (Cited on page 57.)

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 33.)

SystemC (2014). http://www.accellera.org/downloads/standards/systemc. [retrieved: November 2014]. (Cited on page 159.)

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 33.)

Textor, A., Schmid, M., Schaefer, J. and Kroeger, R. (2009). SOA Monitoring Based on a Formal Workflow Model with Constraints, *QUASOSS '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 45.)

The Perl Programming Language (2014). http://www.perl.org. [retrieved: November 2014]]. (Cited on page 159.)

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 79.)

Weinstock, C. B. and Goodenough, J. B. (2006). On system scalability, *Technical report*, DTIC Document. (Cited on page 28.)

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 46.)

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 33 and 46.)

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 32.)

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 41, 47, and 85.)

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 34.)

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 33.)

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 41, 47, and 85.)

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.

Swientek, M., Humm, B., Bleimann, U. and Dowland, P. (2015). A Conceptual Framework for Guiding the Development of Feedback-Controlled Bulk Data Processing Systems, *Accepted for publication in ADAPTIVE 2015, The Seventh International Conference on Adaptive and Self-Adaptive Systems and Applications*.

Service-Oriented Architecture: Performance Issues and Approaches

Martin Swientek[1, 2, 3], Udo Bleimann[1], Paul Dowland[2]

[1]University of Applied Sciences Darmstadt, Germany
[2]Information Security and Network Research Group, University of Plymouth, United Kingdom
sd&m, software design & management AG, Berliner Str. 76, 63065 Offenbach, Germany
E-mail: martin@swientek.org

Abstract

The introduction of a Service-Oriented Architecture (SOA) can affect performance in a negative way. This exacerbates the application of SOA to systems for bulk data processing. This paper describes specific aspects of Service-Oriented Architectures that impact performance particularly. It discusses several approaches to these issues that are currently established and motivates the need for a framework to implement an SOA for bulk data processing systems.

Keywords

SOA, Service-Oriented Architecture, batch processing, performance

1. Introduction

Service-Oriented Architecture (SOA) is becoming a popular approach to integrate heterogeneous applications into an application landscape. Apart from functional requirements, an IT system has to meet the non-functional requirements of the functional and technical operations. Implementing an SOA has certain impacts on these non-functional requirements that ought to be considered beforehand.

Performance is an important non-functional requirement of an IT system and is vital to the acceptance and the operability of the system. Since performance tests are usually not performed until the system is already in place, performance issues are often revealed at a late stage in the development process. In order to improve the performance of the system extensive changes to the architecture are needed which ultimately leads to significant project risks. As the introduction of SOA can deteriorate the performance it is crucial to be aware of the performance drawbacks and how to address them properly at the stage of the system design.

Bulk data processing in particular demands a high-performance implementation. Current approaches to implement an SOA using web service technologies and infrastructures do not match very well with a batch-processing model since they are focused on a request-response communication scheme.

This paper describes the performance issues specific to SOA and discusses current approaches to address them. It motivates the need of a framework to integrate a batch processing system in a service-oriented application landscape. The paper is organized as follows: The next section introduces the concept of Service-Oriented Architecture, describes common properties of batch processing systems and the understanding of performance used in this paper. Section 3 describes the aspects of an SOA that have an impact on performance. The next Section discusses current approaches to the performance issues identified in the preceding section. Section 4 motivates the need of a framework for bulk data processing. This paper concludes with a summarization of the presented performance issues and approaches.

1.1. 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 plattform 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.

Building an SOA involves concrete technical decisions how to implement its concepts. This includes how to implement services, how to discover the appropriate service and how to interconnect them. This paper focusses on these technical decisions that need to be made in order to implement an SOA in a performant way.

1.2. Batch Processing Systems

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 case of a file-based interface, the application reads a record from the input file, processes it and writes the record to the output file.

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. Multiple systems are running in parallel controlled by a job scheduler to speed up 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.

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

1.3. Performance

Performance is a quality attribute of a software system and is crucial to the acceptance of a developed system both by users and IT operations.

The performance of a system can be described by multiple metrics. The following metrics are relevant to the understanding of this paper:

- **Response Time**
  Time it takes for the service consumer to receive a response from the service provider

- **Throughput**
  Number of requests a service provider is able to process in fixed timeframe

- **Latency**
  Time it takes a service request is received by the servicer provider and vice versa

2. SOA Performance Hotspots

This section describes the different aspects of a Service-Oriented Architecture where performance issues typically occur.

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.

Apart from these general issues of a distributed system certain properties of an SOA deteriorate the performance even more.

2.1. 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'Brian 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).

2.2. 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 become interchangeable as long they implement the interface the client expects.

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

- **Knowledge**
  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. describe the following levels of coupling that are leveraged in an SOA (Krafzig et al., 2005).

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

Table 1: Levels of coupling (Krafzig et al., 2005)

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.

3. Current Approaches

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

3.1. 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 or more bandwidth will solve this problem. However, it is often not feasible to add faster hardware in a late stage of the project because it involves more costs than initially planned.

3.2. 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.

3.3. 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, 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).

3.4. 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. 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 inventory 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. argue that distant components should be coupled more loosely than close components (Hess et al., 2006).

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.

4. Applying SOA to Batch Processing Systems

How to apply the concepts of Service-Oriented Architecture to batch processing systems considering the arguments presented in section 2? A naive approach would be the utilisation of web service technologies for these kinds of systems as well. However, because of the performance issues mentioned in this paper, this option would not scale for bulk processing of data with a batch-processing model.

Wichaiwong et al. for example 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 (Wichaiwong et al., 2007). This approach solves the technical aspect of efficiently transferring the input and output data but does not pose any solutions how to implement lose coupling and how to integrate heterogeneous technologies, the fundamental means of an SOA to improve the flexibility of an application landscape.

In order to integrate a batch processing system into a service-oriented application landscape several design decisions need to get addressed:

- How to implement lose coupling?
- What is the appropriate degree of loose coupling?
- What is the right service granularity?
- Which middleware technologies can be utilised for the integration of heterogeneous technologies?
- Who is responsible for data transformation?
- What data formats should be used?

Given that there are no obvious answers to these questions, there is a certain need of a framework that supports the service-oriented integration of batch processing systems by offering proven solutions for these issues. The design of such a framework for the integration of batch processing systems in a service-oriented application landscape will be carried out in a PhD Thesis.

5. Conclusion

The introduction of an SOA generally has a negative impact on performance. Among general performance drawbacks an SOA shares with other distributed technologies, two main concepts of an SOA that deteriorate performance even more are described in this paper. The communication overhead introduced by using intermediate protocols and specific middleware like an ESB to integrate heterogeneous technologies and the utilisation of loose coupling in order to increase the flexibility and maintainability of the application landscape.

This paper discusses several approaches to improve performance in an SOA that are currently established.

The obvious approach is to utilize faster hardware and more network bandwidth. The compression of messages poses an option for reducing transmission times. Other approaches suggest reconsidering the service or architecture design. To decrease the communication overhead immanent in an SOA services should be coarse grained. Is the processing time of a coarse grained service causing problems for a specific service consumer, a finer grained service should be added.

To apply the proper approach to performance issues it is vital to know the bottleneck of the system. Unfortunately, the measuring of system performance and the investigation of bottlenecks can be done only at a late stage in the development phase. SOA Performance models are trying to anticipate the performance behaviour during the design phase but they are currently still under research.

Improving the performance of a system always impacts other quality attributes. Adjusting the degree of loose coupling affects the flexibility of the system. Merging services to more coarse grained services or splitting coarse grained services into finer grained services to solve performance issues of specific service consumers impacts the reusability of the services. Thus, it is vital to find the appropriate trade-off between performance and other quality attributes of the system like flexibility, maintainability and reusability.

Batch processing systems in particular demand a high-performance implementation. In order to integrate these kinds of systems in a service-oriented application landscape several design decisions need to get addressed. For example, what is the appropriate degree of loose coupling and the right service granularity to achieve the required performance? Given that there are no obvious answers to these questions, there is a certain need for a framework for service-oriented processing of bulk data.

The design of such a framework will be the subject of a PhD Thesis, which will be carried out at the University of Plymouth in conjunction with the University of Applied Sciences, Darmstadt.

6. References

Engels, G., Hess, A., Humm, B., Juwig, O., Lohmann, M., Richter, J.-P., Voß, M., Willkomm, J. (2008), Quasar Enterprise, dpunkt.verlag, ISBN: 978-3-89864-506-5.

EXI Working Group (2007), http://www.w3.org/XML/EXI. (Accessed January 2008)

Hess, A., Humm B., Voß, M. (2006), "Regeln für serviceorientierte Architekturen von hoher Qualität", *Informatik Spektrum*, Vol. 29, No. 6, Springer Verlag, pp. 395-411.

Josuttis, N. (2007), *SOA in Practice*, O'Reilly, ISBN: 0596529554.

Krafzig, D., Banke, K., Slama, D. (2005), *Enterprise SOA*, Prentice Hall, ISBN: 0131465759.

O'Brian, L., Merson, P., Bass L. (2007), "Quality Attributes for Service-Oriented Architectures", *Proceedings of the international Workshop on Systems Development in SOA Environments*, International Conference on Software Engineering, IEEE Computer Society, Washington, DC.

Richter, J.-P., Haller H., Schrey, P. (2005), "Aktuelles Schlagwort Serviceorientierte Architektur", *Informatik Spektrum*, Vol. 28, No. 5, Springer Verlag, pp. 413-416.

SOAP Specification (2007), http://www.w3.org/TR/soap (Accessed January 2008).

Wichaiwong, T., Jaruskulchai, C. (2007), "A Simple Approach to Optimize Web Services' Performance", *Proceedings of the Third international Conference on Next Generation Web Services Practices,* IEEE Computer Society, Washington, DC

A SOA Middleware for High-Performance Communication

M. Swientek[123], B. Humm[1], U. Bleimann[1], P. Dowland[2]

[1]University of Applied Sciences Darmstadt, Germany
[2]University of Plymouth, United Kingdom
[3]Capgemini sd&m AG, Offenbach, Germany
E-mail: martin@swientek.org

Abstract

Systems for bulk data processing are often implemented as batch processing systems. While this type of processing in general delivers high throughput, it cannot provide near-time processing of data. Message-based solutions such an ESB are able to provide near-time processing but cannot provide high throughput. This paper presents a new approach to the problem of delivering near-time processing while providing very high throughput by adjusting the data granularity at runtime. It describes how existing SOA middleware can be extended to implement this approach.

Keywords

Middleware, Bulk data processing, Near-time processing, Performance, SOA

1. Introduction

Business software 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. It consists of several sub components that process the different billing sub processes like mediation, rating, billing and presentment (see Figure 1).

Figure 1: Billing sub processes

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.

The performance requirements for such a billing system are high. It has to process more than 1 million records per hour and the whole batch run needs to be finished in a limited timeframe to comply with service level agreements with the print service provider. Since delayed invoicing causes direct loss of cash, it has to be ensured that the bill arrives at the customer on time.

The traditional operation paradigm of such a system for bulk data processing is batch processing (see Figure 2).

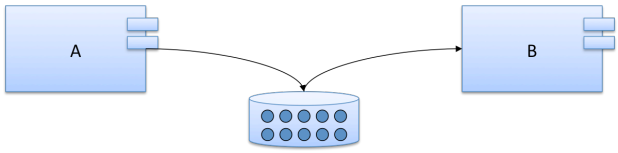


Figure 2: Batch processing

Batch processing exhibits the following properties (Swientek et al., 2008):

- **Bulk processing of data**
  A Batch processing system processes several gigabytes of data in a single run. Multiple systems are running in parallel controlled by a job scheduler to speed up 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.

While such a batch processing system is able to process bulk data and thus delivering a high throughput, it is not able to deliver near-time processing. That is, the latency of a batch processing system is high.

Near-time processing reduces the latency of the system, that is, the time that is spent between the occurrence and the processing of an event. In case of a billing system, it is the time between the user making a call and the complete processing of this call including mediation, rating, billing and presentment. From the customer point of view, an event should be viewable in the customer self-care website shortly after the call has been made. This requirement cannot be implemented using batch processing.

To decrease the latency of the system a message-based approach is needed (see Figure 3), for example by utilising an Enterprise Service Bus (ESB). While this approach provides near-time processing of data, it is not able to deliver the same throughput as batch processing.

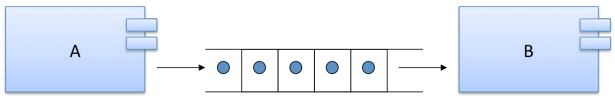


Figure 3: Message based processing

This paper describes a new approach to the problem of delivering near-time processing while providing very high throughput. It is organised as follows: The next section defines the performance attributes throughput and latency in more detail and explains why they are contrary to each other in this case. Section 3 defines the term data granularity and explains how throughput and latency depend on it. Section 4 describes how this approach can be implemented using Sopera ASF which provides an open-source SOA platform. The paper concludes with a summary of the described approach and an outlook to further research.

2. Throughput vs. latency

Throughput and latency are performance metrics of a system. We use the following definitions of throughput and latency in this paper:

- **Throughput**
  The number of events the systems is able to process in fixed timeframe.

- **Latency**
  The period of time between the occurrence of an event and its processing.

In the case of bulk data processing, throughput and latency are contrary to each other (as illustrated in Figure 4). A high throughput, as provided by batch processing, leads to a 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.

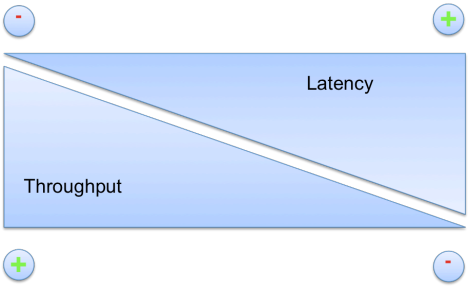


Figure 4: Throughput vs. latency

In order to achieve near-time processing with very high throughput, we propose a combination of both processing types (see Figure 5).

Figure 5: Combining batch processing with message-based processing

This solution should provide the best possible latency with the lowest throughput that is still acceptable to meet the performance requirements.

3. Data granularity

Throughput and latency of the system depend on the granularity of data that is being 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. Haesen et al. distinguishes between two types of data granularity (Haesen et al., 2008):

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

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

Additionally, data granularity can relate to different orientations:

- **Horizontal data granularity**
  Refers to the amount of data or fields that is contained in a single record

- **Vertical data granularity**
  Refers to the total number of records

The remainder of this paper focuses on vertical data granularity. No distinction is being made regarding input and output data granularity.

Batch processing uses a high granularity of data, which leads to high throughput and high latency. Message-based processing uses low granularity of data, which leads to low latency but also low throughput. The optimum data granularity would allow having the lowest possible latency with the lowest acceptable throughput and thus providing near-time processing of bulk data (see Figure 6).

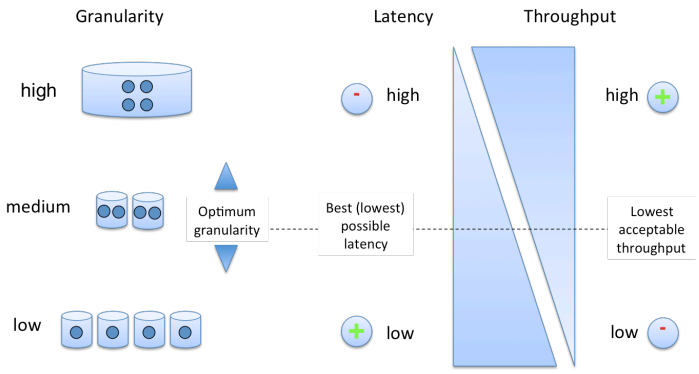


Figure 6: Throughput and latency depend on data granularity

3.1. Variable adjustment of granularity

The granularity of the data processed in one message will be adjusted at runtime. A middleware is needed that provides services to constantly measure the throughput and latency of the system and to control the granularity of the data (see Figure 7). If the throughput drops below the acceptable minimum, the granularity of the data needs to be higher. On the other hand, the granularity can be lowered, if the throughput of the system is above the minimum.

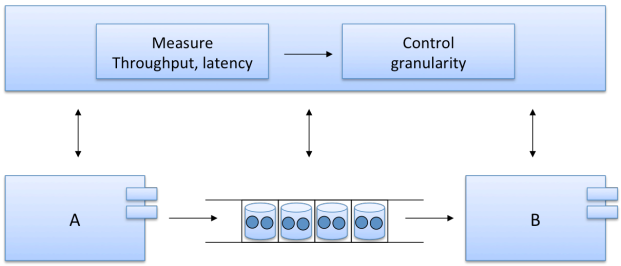


Figure 7: Variable adjustment of granularity

4. Implementation

This section describes how to implement the variable adjustment of data granularity by extending the Sopera ASF platform.

4.1. Sopera Advanced Service Factory

The Sopera Advanced Service Factory (Sopera ASF) provides an open source SOA (Service Oriented Architecture) platform, which has been developed and successfully deployed at Deutsche Post AG. The core of the platform is the Sopera ESB. The Sopera ESB is implemented as a distributed service bus. An Enterprise Service Bus (ESB) is an integration platform that combines messaging, web services, data transformation and intelligent routing (Schulte, 2002).

The main components of the Sopera ESB are the Sopera Library (SSB Library) and the Sopera Service Management (SSM). The Sopera Library represents the service container of the Sopera ESB and provides access for all participants, mediation of the SOA functionality and message exchange. Sopera Management provides functionality for monitoring the operations of the SOA platform including performance, error handling and reporting and provides methods to control the behaviour of the service participants.

Additional infrastructure services are provided as plug-ins. Sopera ASF includes the following plug-ins:

- Service registries/repositories
- Security services
- Messaging/Transport services
- Orchestration/Workflow server

Sopera ASF supports different Message Queuing Server such as Apache ActiveMQ, JORAM and IBM WebSphere MQ. In addition to the ESB, Sopera ASF also provides an extensive tool suite based on the Eclipse IDE including editors to define services, policies and process flows.

We will use the Sopera ASF platform to implement the adjustment of data granularity to reduce the latency of bulk data processing as introduced in section 3. The platform has been chosen because of its best of breed approach using open source components. All source code is freely available. Additionally, the reliability of the platform has been proven in a huge deployment at Deutsche Post.

The next section describes the design of the components that comprise the proposed solution.

4.2. Component architecture

Figure 8 shows the components, which are involved in the adjustment of data granularity at runtime.

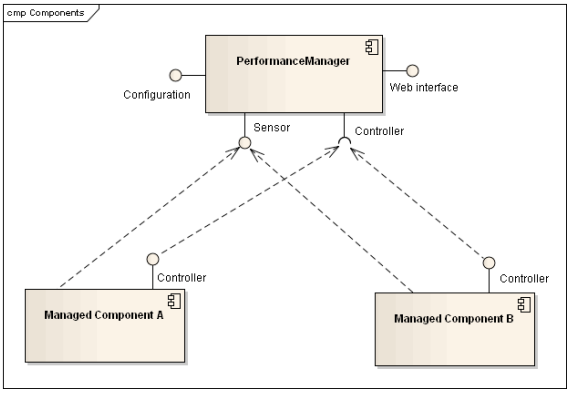


Figure 8: Components

The main component is the Performance Manager. It constantly measures the throughput and latency of managed components and controls their data granularity. Every managed component constantly sends a notification to the Performance Manager containing its current throughput and latency. The Performance Manager buffers the incoming notification messages and computes the current throughput and latency of the complete business process. If the computed latency exceeds the pre-defined limit, the Performance Manager adjusts the data granularity of the managed components.

The communication between the Performance Manager and the managed components will be implemented using Java Management Extensions (JMX).

4.2.1. Performance Manager

The Performance Manager is an infrastructure service and will be implemented as a Web application, which runs inside a standard Servlet container (see Figure 9).

The Performance Manager provides the following interfaces.

- **Sensor interface**
  The Sensor interface receives JMX (Java Management Extension) notification messages from managed components containing their current throughput and latency.

- **Performance Manger Client interface**
  The Performance Manager client interface exposes the Performance Manager Client application and is used to set the pre-defined limits for the latency and throughput of the business process.

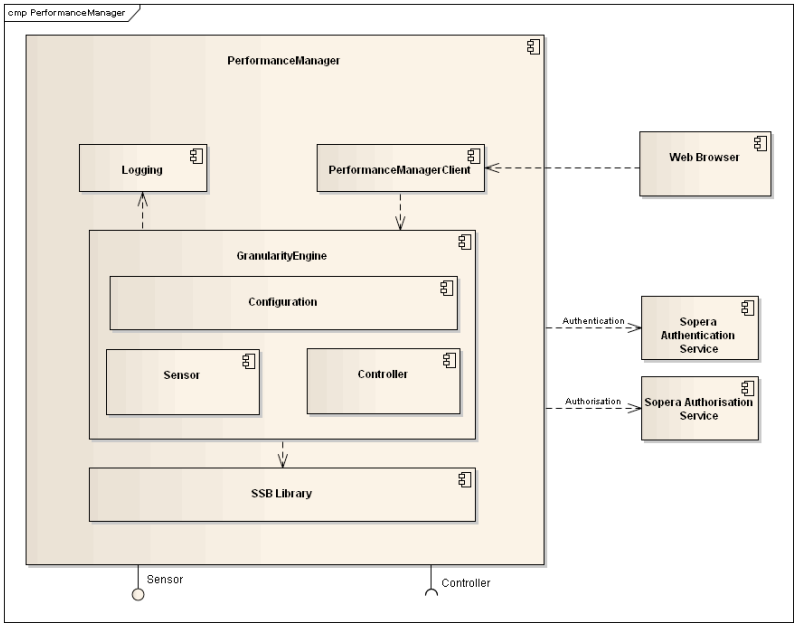


Figure 9: Component Performance Manager

The Performance Manager is comprised of the following sub components.

- **Granularity Engine**
  The Granularity Engine is the core of the Performance Manager service. It consists of the components Sensor, Controller and Configuration. The Sensor component receives JMX notification messages from managed components. The Granularity Engine computes the throughput and latency of the complete business process using the notifications and compares the computed values with the pre-defined limits stored in the Configuration component. If the computed values exceed the pre-defined limits, the Controller sends a message to the corresponding managed components to adjust the data granularity.

- **SSB Library**
  The SSB Library provides the integration of the Performance Manager in the Sopera ASF platform. It is used to receive the notification messages of the managed components.

- **Logging**
  The Logging component logs all measured and computed values of the managed components and all adjustments of the data granularity performed by the Granularity Engine. The logs can be viewed using the Performance Manager Client application.

- **Performance Manager Client**
  The Performance Manager Client application provides a user interface to set

the pre-defined limits for throughput and latency. It also offers functionality to manually control the data granularity and to view the logs written by the Logging component.

- **Authentication and Authorisation**
  The Performance Manager uses the Authentication and Authorisation services provided by the Sopera ASF platform.

4.2.2. Managed Component

The SSB Library already contains an SSM module which provides the management functionality for a service participant. The SSM module contains several MBeans (Management Beans), which monitor the message traffic that passes through an instance of the SSB Library, including the average number of requests per minute, the total number of request for a fixed time and the percentage of failed requests. The data is available at different levels of aggregation. The ParticipantMonitor provides data about the service participant. The ServiceMonitor and OperationMonitor provide data about a service and an operation respectively (Sopera Operations and Administration Guide).

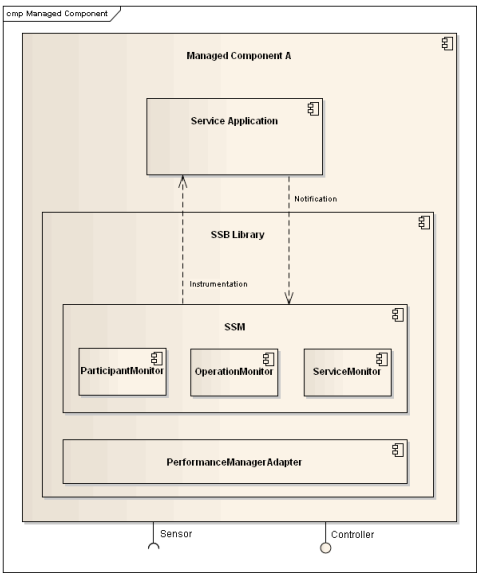


Figure 10: Managed Component

We will extend the existing SSM components to measure the throughput and latency of the service participant and add operations to control its data granularity.

The Performance Manager Adapter provides the Controller interface used by the Performance Manager to control the MBeans of the SSM components. Additionally, the adapter publishes notification messages containing the current throughput and latency of the managed component.

4.3. Implementation considerations

The following considerations need to be taken into account when implementing the proposed solution.

4.3.1. Measuring Throughput and latency of orchestrated services

In order to compute the throughput and latency of a complete business process, the Performance Manager needs to know which services are orchestrated to compose this business process. The Performance Manager will be able to retrieve business process definitions from the workflow engine attributed with limits for the maximum latency and minimum acceptable throughput. It might be necessary to extend the utilised business process language to support these attributes including the corresponding tools.

4.3.2. Transport of large messages

The message size cannot be arbitrarily increased because very large messages cannot be transported efficiently by the messaging system. If the data granularity exceeds a certain level, it might be required that the payload of the message is transported outside of the messaging system by using FTP (File Transfer Protocol) or similar transports.

5. Conclusion

Business software systems for bulk data processing commonly utilise batch processing. These systems are more and more faced to also provide near-time processing due to changed business requirements such as customer demand. While a batch processing system is able to provide the required high throughput, it cannot meet the requirements regarding low latency necessary for near-time processing. On the other hand, message-based processing is able to deliver low latency but cannot provide the required high throughput.

Latency and throughput depend on the granularity of data that is being processed. Batch processing uses coarse-grained data and therefore exhibits a high latency. Message-based processing uses fine-grained data, i.e. messages, and therefore exhibits a low latency. The optimum data granularity would allow having the lowest possible latency with the lowest acceptable throughput and thus providing near-time processing of bulk data. We suggest that the granularity of data will be adjusted at runtime by a middleware, which continuously measures the throughput and latency of the system.

Sopera ASF is an adequate integration platform to implement the described approach. The necessary infrastructure services for monitoring the throughput and latency of the system and for adjusting the granularity of data will be implemented as plug-ins of the Sopera ESB.

The next step is the implementation of the proposed solution along with comprehensive performance tests.

6. References

Chappel, D. (2004), *Enterprise Service Bus*, O'Reilly, ISBN 0-596-00675-6.

Haesen, R., Snoeck, M., Lemahieu, W. and Poelmans, S. (2008), "On the definition of service granularity and its architectural impact", *CAiSE '08: Proceedings of the 20th international conference on Advanced Information Systems Engineering*, Springer Verlag, Berlin, Heidelberg, Germany, pp. 375–389.

JMX, Java Management Extensions, http://java.sun.com/j2se/1.5.0/docs/guide/jmx/index.html, (Accessed 29. September 2009).

Schulte, R. W. (2002), "Predicts 2003: Enterprise Service Buses Emerge", Gartner.

Sopera ASF, http://www.sopera.de/en/home, (Accessed 10. August 2009).

Sopera Operations and Administration Guide, http://www.sopera.de/nc/en/support/bibliothek/sopera-32/opadmin32/, (Accessed 30. September 2009).

Swientek, M., Bleimann U. and Dowland P. (2008), "Service-Oriented Architecture: Performance Issues and Approaches", *Proceedings of the Seventh International Network Conference (INC 2008)*, Plymouth, UK, pp. 261-269.

An Adaptive Middleware for Near-Time Processing of Bulk Data

Martin Swientek
Paul Dowland

School of Computing and Mathematics
Plymouth University
Plymouth, UK
e-mail: {martin.swientek, p.dowland}@plymouth.ac.uk

Bernhard Humm
Udo Bleimann

Department of Computer Science
University of Applied Sciences Darmstadt
Darmstadt, Germany
e-mail: {bernhard.humm, udo.bleimann}@h-da.de

***Abstract*—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 over time. In this paper, we introduce the concept of a middleware 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 minimise the end-to-end latency for different load scenarios.**

Keywords–adaptive middleware; message aggregation; latency; throughput

I. 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 [1]. 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 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.

In this paper, we propose a solution to this problem:

- We introduce the concept of a middleware 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. (Section III)

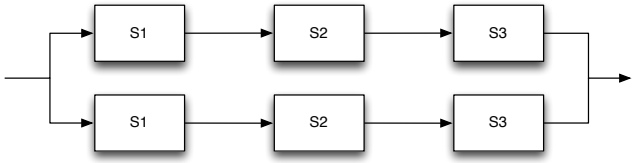
The remainder of this paper is organized as follows. Section II defines the considered type of system and the terms throughput and latency. The proposed middleware and the results of preliminary performance tests are presented in Section III. Section IV gives an overview of other work related to this reasearch. Finally, Section V concludes the paper and gives and outlook to the next steps of this research.

II. BACKGROUND

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).

 ISBN: 978-1-61208-341-4

(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 beeing 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.

A. 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 organized in records using a file- or database-based interface. In the case of a file-based interface, the application reads a record from the input file, processes it and writes the record to the output file.

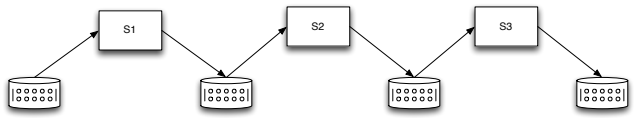


Figure 2. Batch processing

B. 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 [2].

Figure 3. Message-based processing

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 maximum throughput because of the additional overhead for each processed message. Every message needs, amongst others, to be serialized and deserialized, mapped between different protocols and routed to the appropriate receiving system.

C. End-to-end Latency vs. Maximum Throughput

Throughput and latency are performance metrics of a system. We are using the following definitions of maximum throughput and latency in this paper:

- **Maximum Throughput**
  The number of events the system is able to process in a fixed timeframe.
- **End-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.

Latency and maximum throughput are opposed to each other given a fixed amount of processing resources. High maximum 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 maximum throughput needed for bulk data processing because of the additional overhead for each processed event.

III. An Adaptive Middleware for Near-Time Processing of Bulk Data

This section 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.

A. Middleware Components

Figure 4 shows the components of the middleware, that are based on the Enterprise Integration Patterns described by Hohpe et al. [3].

1) Aggregator: 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 messaging route.

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.

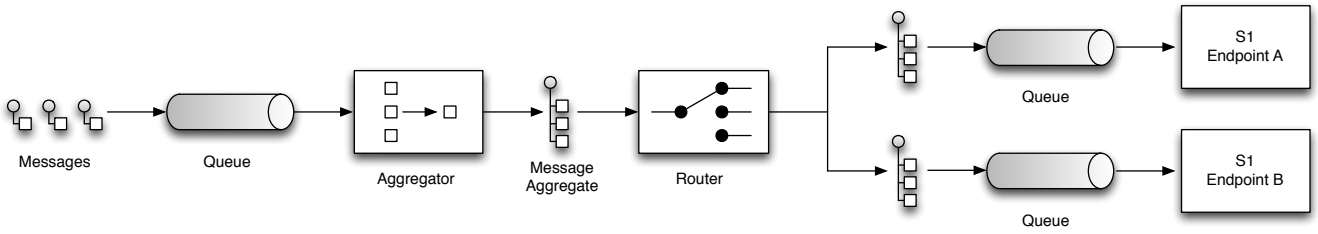


Figure 4. Components of the Adaptive Middleware. We are using the notation defined by [3]

- **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.

2) Feedback Loop: To control the level of message aggregation at runtime, the middleware uses a closed feedback loop with the following properties (see Figure 5):

- **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 the average end-to-end latency.

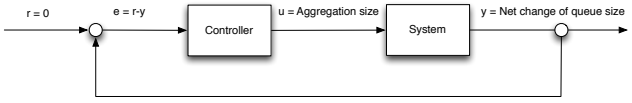


Figure 5. Feedback loop to control the aggregation size

The concrete architecture and tuning of the feedback loop and the controller is subject to our ongoing research.

3) Router: Depending on the size of the aggregated message, the Router routes the message to the appropriate service endpoint, which is either optimized for batch or single event processing.

When processing data in batches, especially when a batch contains correlated data, there are multiple ways to speed up the processing:

- To reduce I/O, data can be pre-loaded at the beginning of the batch job and held in memory.
- Storing calculated results for re-use in memory
- Use bulk database operations for reading and writing data

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.

B. Prototype Implementation

To evaluate the proposed concepts of the adaptive middleware, we have implemented a prototype of a billing system using Apache Camel [4] as the messaging middleware.

Figure 6 shows the architecture of the prototype system.

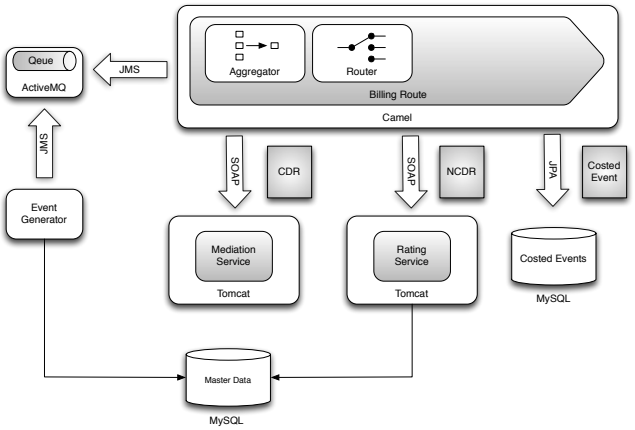


Figure 6. Architecture of the prototype system

Using this prototype, we have done some preliminary performance tests to examine the impact of message aggregation on latency and throughput. For each test, the input message queue has been pre-filled with 100.000 events. We have measured the total processing time and the processing time of each message with different static message aggregation sizes.

Figure 7 shows the impact of different aggregation sizes on the throughput of the messaging prototype. The throughput increases constantly for $1 < aggregation\_size <= 50$ with a maximum of 673 events per second with $aggregation\_size = 50$. Higher aggregation sizes than 50 do not further increase the throughput, it stays around 390 events per second.

The increased throughput achieved by increasing the aggregation size comes with the cost of a higher latency. Figure 8 shows the impact of different aggregation sizes on the 95th percentile latency of the messaging prototype.

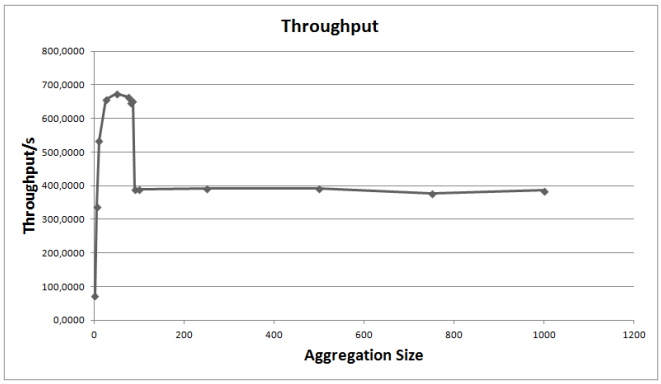


Figure 7. Impact of different aggregation sizes on throughput

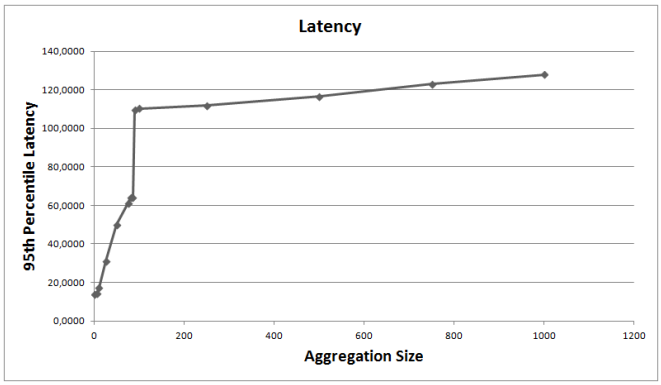


Figure 8. Impact of different aggregation sizes on latency

An aggregation size of 50, resulting in the maximum throughput of 673 events per seconds, shows a 95th percentile latency of about 68 seconds.

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. This threshold needs to be considered by the control strategy. We are currently investigating the detailed causes of this finding.

IV. Related Work

Research on messaging middleware currently focusses on Enterprise Services Bus (ESB) infrastructure. An ESB is an integration plattform that combines messaging, web services, data transformation and intelligent routing to connect multiple heterogeneous services [5]. 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 [6], routing [7] [8] [9] and load balancing [10].

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 [11]. Gonzaléz et al. [12] propose an adaptive ESB infrastructure to adress 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.

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 [13] [14] [15] [16].

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 [17] or Google App Engine [18]. 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.

V. Conclusion and Future Work

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 messsage, 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 controller 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

References

[1] J. Fleck, “A distributed near real-time billing environment,” in Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA ’99, 1999, pp. 142–148.

[2] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch, Enterprise Application Integration: Grundlagen, Konzepte, Entwurfsmuster, Praxisbeispiele. Elsevier, Spektrum, Akad. Verl., 2006.

 ISBN: 978-1-61208-341-4

[3] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[4] Apache Camel. http://camel.apache.org. [retrieved: March 2014].

[5] D. Chappell, Enterprise Service Bus. Sebastopol, CA, USA: O'Reilly Media, Inc., 2004.

[6] S.-H. Chang, H. J. La, J. S. Bae, W. Y. Jeon, and S. D. Kim, "Design of a dynamic composition handler for esb-based services," in e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on, Oct 2007, pp. 287–294.

[7] X. Bai, J. Xie, B. Chen, and S. Xiao, "Dresr: Dynamic routing in enterprise service bus," in e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on, Oct 2007, pp. 528–531.

[8] B. Wu, S. Liu, and L. Wu, "Dynamic reliable service routing in enterprise service bus," in Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE, Dec 2008, pp. 349–354.

[9] G. Ziyaeva, E. Choi, and D. Min, "Content-based intelligent routing and message processing in enterprise service bus," in Convergence and Hybrid Information Technology, 2008. ICHIT '08. International Conference on, Aug 2008, pp. 245–249.

[10] A. Jongtaveesataporn and S. Takada, "Enhancing enterprise service bus capability for load balancing," W. Trans. on Comp., vol. 9, no. 3, Mar. 2010, pp. 299–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1852392.1852401

[11] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," Software Engineering, IEEE Transactions on, vol. 37, no. 3, May 2011, pp. 387–409.

[12] L. González and R. Ruggia, "Addressing qos issues in service based systems through an adaptive esb infrastructure," in Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, ser. MW4SOC '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:7. [Online]. Available: http://doi.acm.org/10.1145/2093185.2093189

[13] R. Friedman and R. V. Renesse, "Packing messages as a tool for boosting the performance of total ordering protocls," in Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, ser. HPDC '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 233–.

[14] R. Friedman and E. Hadad, "Adaptive batching for replicated servers," in Reliable Distributed Systems, 2006. SRDS '06. 25th IEEE Symposium on, 2006, pp. 311–320.

[15] P. Romano and M. Leonetti, "Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning," in Computing, Networking and Communications (ICNC), 2012 International Conference on, Jan 2012, pp. 786–792.

[16] D. Didona, D. Carnevale, S. Galeani, and P. Romano, "An extremum seeking algorithm for message batching in total order protocols," in Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on, Sept 2012, pp. 89–98.

[17] "Amazon ec2 auto scaling," http://aws.amazon.com/autoscaling, [retrieved: March 2014].

[18] Auto scaling on the google cloud platform. https://cloud.google.com/developers/articles/auto-scaling-on-the-google-cloud-platform. [retrieved: March 2014].

 ISBN: 978-1-61208-341-4

A Conceptual Framework for Guiding the Development of Feedback-Controlled Bulk Data Processing Systems

Martin Swientek
Paul Dowland

School of Computing and Mathematics
Plymouth University
Plymouth, UK
e-mail: {martin.swientek, p.dowland}@plymouth.ac.uk

Bernhard Humm
Udo Bleimann

Department of Computer Science
University of Applied Sciences Darmstadt
Darmstadt, Germany
e-mail: {bernhard.humm, udo.bleimann}@h-da.de

***Abstract*—The design, implementation and operation of an adaptive enterprise software system for bulk data processing differs from common approaches to implement enterprise systems. Different tasks and activities, different roles with different skills and different tools are needed to build and operate such a system.**

This paper introduces a conceptual framework that describes the development process of how to build an adaptive software for bulk data processing. It defines the needed roles and their skills, the necessary tasks and their relationship, artifacts that are created and required by different tasks, the tools that are needed to process the tasks and the processes, which describe the order of tasks.

***Index Terms*—adaptive middleware; software development process**

I. Introduction

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 over time.

Traditionally, enterprise systems for bulk data processing are implemented as batch processing systems [1]. 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 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 [2].

Additionally, enterprise systems often need to handle load peaks that occur infrequently. When the system faces moderate load, a low end-to-end latency of the system is preferable. 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.

In [2] we have introduced the concept of a middleware 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.

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 as 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 [3].

This paper introduces a conceptual framework to guide the design, implementation and operation of an adaptive system for bulk data processing. It defines views, roles, tasks and their dependencies, and processes to describe the necessary steps for design, implementation and operation of an adaptive system for bulk data processing.

Figure 1 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**
  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.

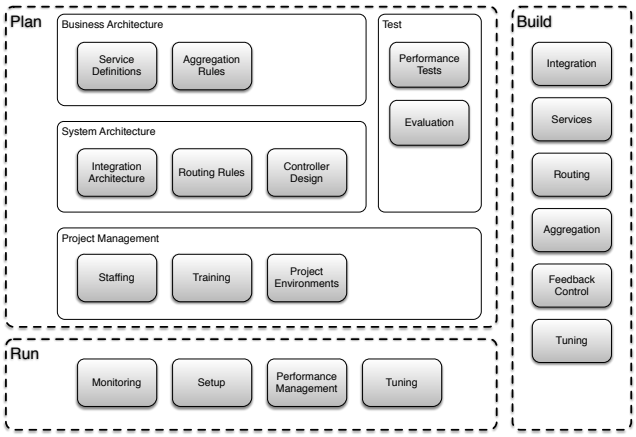


Fig. 1. Overview of Conceptual Framework

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.

The remainder of this paper is organized as follows. Section II briefly introduces the concept of an adaptive middleware for bulk data processing. The conceptual framework is presented in Section III. Section IV gives an overview of other work related to this research. Finally, Section V concludes the paper and gives an outlook to the next steps of this research.

II. Background

This section briefly 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 [2].

Figure 2 shows an overview of the adaptive middleware and its components.

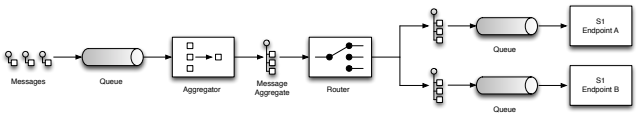


Fig. 2. Overview of the adaptive middleware for bulk data processing [2]

The components of the middleware are based on the Enterprise Integration Patterns described by [4], as shown in Table I.

To control the level of message aggregation at runtime, the middleware uses a closed feedback loop with the following properties (see Figure 3):

TABLE I
Components of the Adaptive Middleware. We are using the notation defined by [4]

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 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.
	Service Endpoint	Represents a business service.

- **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.

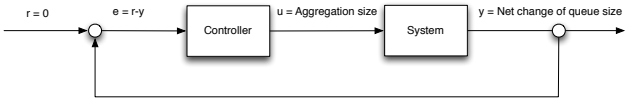


Fig. 3. Feedback loop to control the aggregation size

III. Conceptual Framework

The design, implementation and operation of a system based on the adaptive middleware introduced in Section II differs from common approaches to implement enterprise systems. We have therefore developed a conceptual framework to describe a development process how to build such a system.

A. Metamodel

The conceptual framework consists of the following entities, as shown in Figure 4:

- **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.

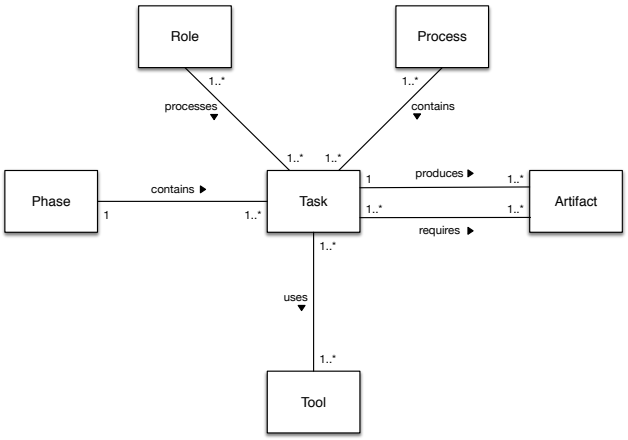


Fig. 4. Metamodel

B. 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.

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.
- **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.

A role is described by the following attributes:

- **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.

C. 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 [2]. It does not describe common tasks or activities that are needed for every software system.

Figure 5 shows an overview of the tasks grouped by the different phases of the Conceptual Framework.

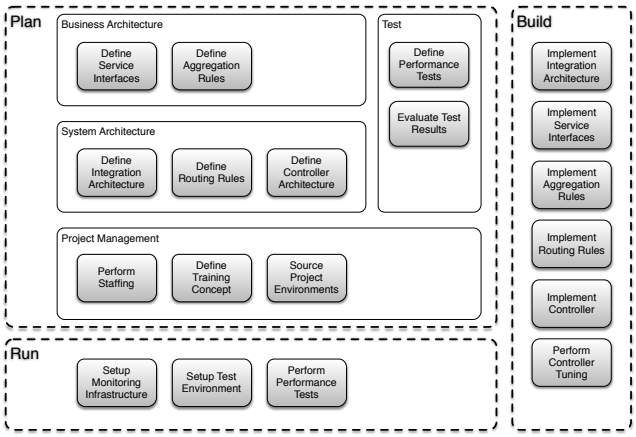


Fig. 5. Overview of tasks

A Task is described by the following attributes:

- **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 artificats 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.

D. 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 Unified Modeling Language (UML) activity diagrams. The conceptual framework describes the following processes:

- Implement Integration
- Implement Aggregation
- Implement Feedback-Control

1) Implement Integration: This process describes the necessary tasks to implement the integration layer and the integrated service interfaces, as shown in the UML activity diagram in Figure 6.

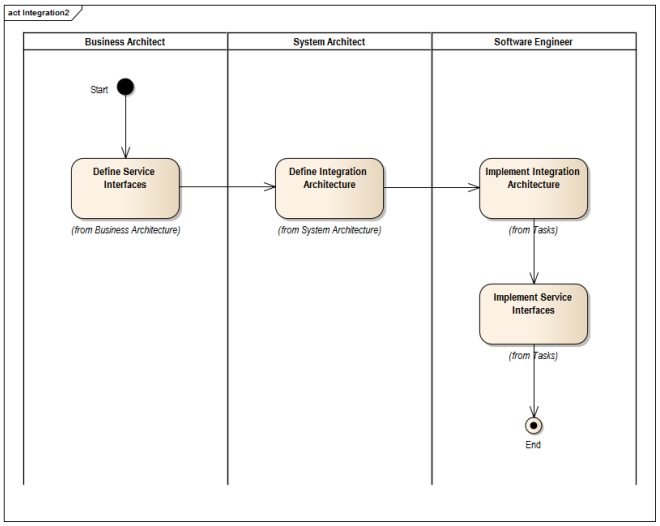


Fig. 6. UML Activity Diagram: Implement Integration

2) Implement Aggregation: This process is concerned with the implementation of the message aggregation, as shown in the UML activity diagram in Figure 7.

3) Implement Feedback-Control: This process contains tasks that are concerned with the design, implementation and tuning of the feedback-control loop, as shown in Figure 8.

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 9a.
- 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 9b.

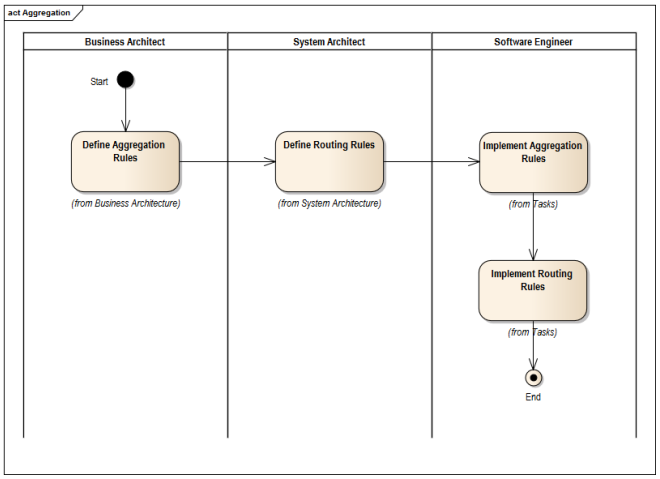


Fig. 7. UML Activity Diagram: Implement Aggregation

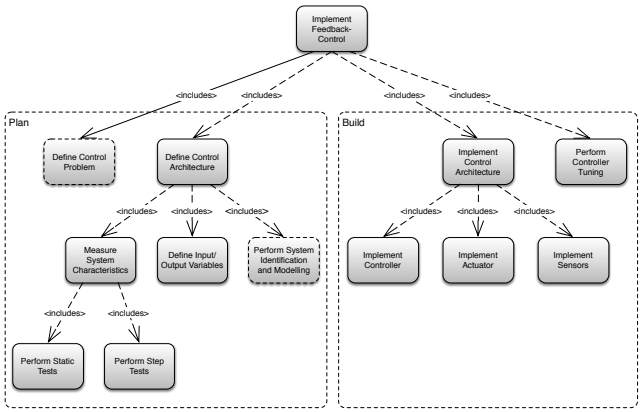


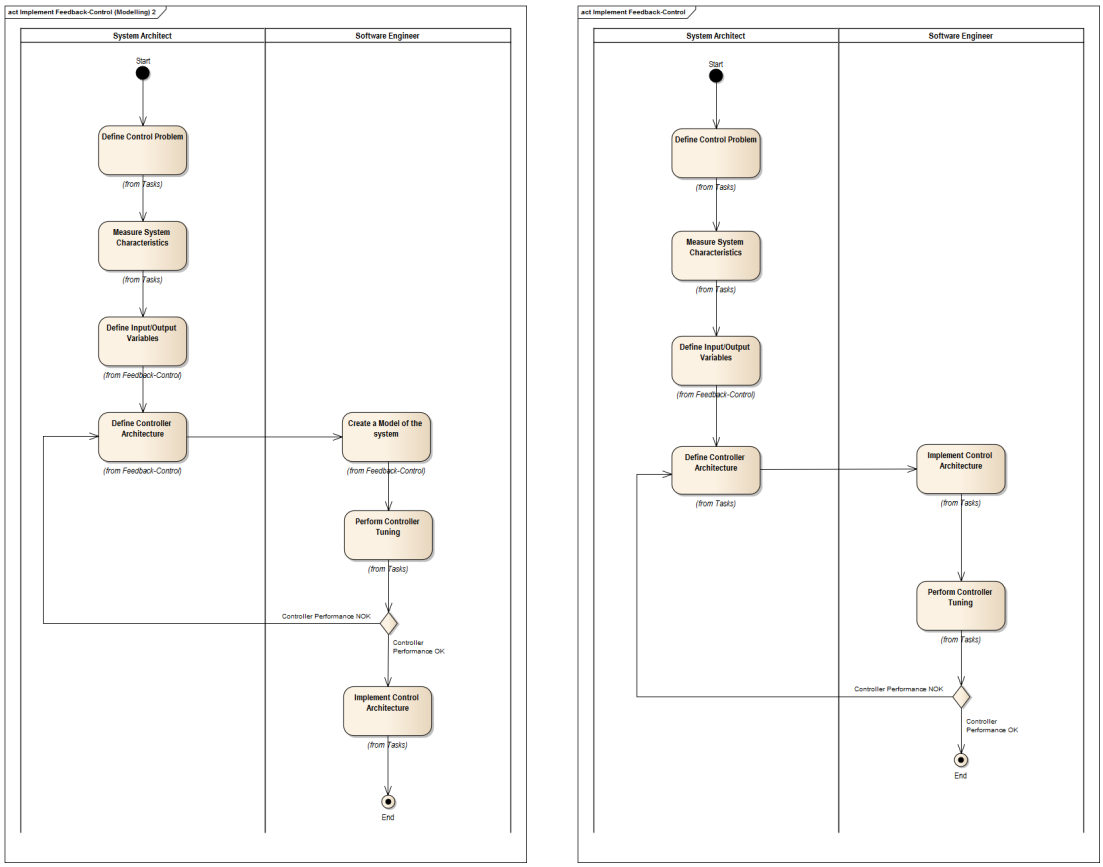
Fig. 8. Tasks for implementing the feedback-control loop

E. 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 defines the following artifacts:

- **Performance Requirements**
  Defines the requirements regarding the performance of the system, such as required maximum throughput, required maximum latency or desired minimum latency. Defines the workload scenarios of the system.
- **Service Interface Definition**
  Defines the structure of input and output data. Does not include informations about the technical format, such as Extended Markup Language (XML) or JavaScript Object Notation (JSON), and the integration style, such SOAP or Representional State Transfer (REST).
- **Aggregation Rules**
  Defines how events should be correlated with each other by the Aggregator.



(a) Using a model

(b) Without using a model

Fig. 9. UML Activity Diagram: Implement Feedback-Control Loop

- **Integration Architecture**
  Defines the technical integration of the business services, including Middleware technology or product, transports, such as Java Messaging Service (JMS), SOAP or File Transfer Protocol (FTP), Technical format of the input and output data, such as XML or JSON, Comma Separated Values (CSV) or binary formats.
- **Routing Rules**
  Defines which service endpoint should be called by the Router for a given aggregation size.
- **System Model**
  The system model is used to build a simulation of the system which can be used for implementing the controller.
- **Controller Configuration**
  The controller configuration specifies the parameter of the Controller.
- **Training Concept**
  Defines the training concept, including the audience, the content and the type of training. Additionally it contains a timeplan, learning modules and needed facilities to conduct the training.
- **Staffing Plan**
  Defines the required team members and their utilisation over the project time (staffing curve), the required roles and their assignment to team members and a skill matrix that shows the required skills and the knowledge of each team member.

An artifact is described by the following attributes:

- **Name**
  The name of the artifact.
- **Description**
  A description of the artifact.
- **Task**
  The task that produces the artifact.
- **Role**
  The role that is responsible for producing the artifact.

IV. Related Work

This section discusses work related to the conceptual framework presented in this paper. It introduces the terms *Software Process* and *Software Process Modelling* and discusses approaches to model the software process using UML.

A. Software Process

"The software process is a partially ordered set of activities undertaken to manage, develop and maintain software

systems." [5]

McChesney [6] 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 Fugetta [3]. 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. [7] Examples of software lifecycle models are the waterfall model [8] or the spiral model [9].

B. Software Process Modelling

Software process modelling describes the creation of software development models [5]. A software process model is "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" [10].

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 [13].

Typical elements of PMLs are (see for example [11], [5], [3] and [12]):

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

Process Models commonly use different perspectives to describe the software process [12]:

- 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

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

C. 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 [13], [14]. 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 [15]. 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.

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.

[16] 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, that are described by UML activity diagrams, class diagrams and use-case diagrams: process-oriented, activity-oriented, product-oriented, and role-oriented. The following UML diagram types are used by their approach:

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.

V. Conclusion

In this paper, we have presented a conceptual framework to guide the design, implementation and operation of an enterprise system that implements the adaptive middleware for bulk data processing as described in [2].

The conceptual framework consists of the entities phases, roles, tasks, artifacts 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 should be noted that software processes are not fixed during their lifetime, they need to be continuously improved. [3] The conceptual model can therefore be tailored to specific projects requirements, it does not have to be followed strictly.

The next step of this research is the evaluation of the conceptual framework by using quantitative research methods, such as expert interviews and its application in real-life projects.

REFERENCES

[1] J. Fleck, "A distributed near real-time billing environment," in *Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA '99*, 1999, pp. 142–148.

[2] M. Swientek, B. Humm, U. Bleimann, and P. Dowland, "An Adaptive Middleware for Near-Time Processing of Bulk Data," in *ADAPTIVE 2014, The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications*, Venice, Italy, May 2014, p. 37 to 41.

[3] A. Fuggetta, "Software process: a roadmap." *ICSE - Future of SE Track*, pp. 25–34, 2000.

[4] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[5] S. T. Acuña and X. Ferré, "Software process modelling." in *ISAS-SCI (1)*, 2001, pp. 237–242.

[6] I. McChesney, "Toward a classification scheme for software process modelling approaches," *Information and Software Technology*, vol. 37, no. 7, pp. 363 – 374, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/095058499591492I

[7] S. T. Acuña and X. Ferre, "The software process: Modelling, evaluation and improvement," *Handbook of Software Engineering and Knowledge Engineering*, vol. 1, pp. 193–237, 2001.

[8] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques." *ICSE*, pp. 328–339, 1987.

[9] B. W. Boehm, "A Spiral Model of Software Development and Enhancement." *IEEE Computer ()*, vol. 21, no. 5, pp. 61–72, 1988.

[10] P. Feiler and W. Humphrey, "Software process development and enactment: concepts and definitions," in *Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*, Feb 1993, pp. 28–40.

[11] K. Benali and J. C. Derniame, "Software processes modeling: What, who, and when," in *Software Process Technology*. Berlin/Heidelberg: Springer Berlin Heidelberg, Jan. 1992, pp. 21–25.

[12] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 75–90, Sep. 1992.

[13] R. Bendraou, M.-P. Gervais, and X. Blanc, "UML4SPM: A UML2.0-Based Metamodel for Software Process Modelling," in *Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 17–38.

[14] ——, "UML4SPM: An Executable Software Process Modeling Language Providing High-Level Abstractions," in *Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International*, Oct 2006, pp. 297–306.

[15] OMG, "Software Process Engineering Metamodel SPEM 2.0," Object Management Group, Technical Report ptc/08-04-01, 2008.

[16] S. Dietrich, P. Killisperger, T. Stückl, N. Weber, T. Hartmann, and E.-M. Kern, "Using uml 2.0 for modelling software processes at siemens ag," in *Information Systems Development*, R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry, and M. Lang, Eds. Springer New York, 2013, pp. 561–572. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-4951-5_45