Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master's Thesis No. MCS-0004

Decision Support for Middleware Performance Benchmarking

Tayyaba Azad

Course of Study:	Computer Science M.Sc
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	M.Sc., M.A. Marigianna Skouradaki
Commenced:	2nd November 2015
Completed:	2nd May 2016
CR-Classification:	H.4.1, K.6.2, D.2

Abstract

Along with the rapid development of computing systems, the heterogeneity amongst them also increases. With the usage of middleware technologies, the systems work together in a homogeneous environment and allow to integrate previously independent applications, together with new developments. Along with the growing popularity of middleware systems, the performance and efficiency of their underlying technology becomes crucial for the business. For that reason, applying benchmarks on such environments is highly required. Current technologies and literature focus on building diverse benchmarks in order to test the performance of the underlying middleware. The development of a standard benchmark for middleware is extremely challenging, as one needs to realistically stress the different software capabilities. However, information on how to do so, is generally missing, thus the users need to arbitrarily make crucial design decisions.

This Master's thesis aims at providing the means to ease the decision-making for selecting the appropriate middleware benchmark and enables the user to make crucial design decisions when the creation of a new middleware benchmark is intended. We propose the creation of the first Decision Support System for middleware performance benchmarking, capable of guiding the user through relevant components of a benchmark. The prototype is based on current web technologies using the REST architectural style and it provides a decision support approach for decision makers considering the creation of a new middleware benchmark or selecting the right middleware benchmark of choice. At the end, we validate through use cases to show that the functionalities of the system are accomplished successfully.

Key words: Benchmarking, Middleware, Decision Support, REST, Web Service

Acknowledgment

I would like to express my appreciation and gratitude to Prof. Dr. Dr. h. c. Frank Leymann and especially my supervisor Marigianna Skouradaki from the Institute of Architecture of Applications Systems (IAAS) at the University of Stuttgart who gave me this opportunity and believed in my vision towards achieving the results of this work. Your support, encouragement and advice have been affectionate while achieving the goals of my task.
I would also like to express my deep appreciation to my family. Words are not plentiful to convey how grateful I am for the prayers and assistance of my mother, my father, my brother, and my sisters.
I would also like to thank all the people who motivated and encouraged me to accomplish my target. Special thanks to my friends Asfa and Elekci, your moral support has always been deeply beneficial.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Benchmarking is known as the act of measuring quality, programs and performance of different peers [A^+15]. In order to attain performance characteristics from different systems it is necessary to apply benchmarking. Moreover, it allows the comparison with standard measurements of similar type. Previously, a number of middleware benchmarks have been developed and applied in different sectors for performance testing and comparisons, they did not fulfill the main requirements for a successful benchmark [SKBB09]. The main reasons for their failure was the consideration of artificial workload not demonstrating a real-world application scheme. Also, users of the benchmark could not adjust the workload flexibly in a customised manner, where specific components of the MOM performance could be aimed. For that reason being, standard benchmarks have to be introduced [SKBB09].

Event-based systems are used more often to construct loosely-coupled applications. Event-based systems have gained attention in many areas of the industry, such as, manufacturing, transportation, health-care and supply chain managements [KS09]. Along with the growing popularity of event-based systems the performance and efficiency of the underlying technology becomes crucial for the business. For that reason, applying benchmarks on such environments is highly required. [ASB10]. Therefore, these platforms have to be tested using benchmarks to determine and justify their performance and scalability, thereby the Quality of Service (QoS) requirements can be guaranteed. With benchmarking not only can other platforms be compared and validated, but benchmarks can also be applied to review the consequences of particular platform-based parameters on the overall system behaviour [SKCB07]. Apart from comparing different products with the help of a standard workload, developers can also recognise limitations and drawbacks that allow to identify further design guidelines. Moreover, with the detailed comprehension of system usage under specified loads, the scalability and the control of altering parameters can be assessed [ASB10].

In recent years, the usage of standadrised benchmarks have become generally acknowledged in the field of performance evaluation and comparison [A^+15]. These benchmarks are provided by corporations like SPEC and TPC and are published under high concealment, so outside parties have less chances to reuse internal processes and structures. With this increasing development of middleware benchmarking it may be necessary to aid the decision-making for benchmark users in terms of selecting the relevant benchmark.

1.1 Challenge Statement and Research Objectives

The previous section described how the research field of decision support for middleware benchmarking has been established. It also addressed the need of a decision support for selecting the right middleware benchmark. Currently, the State of the Art is missing an appropriate approach to provide decision support for middleware benchmarks.

A Decision Support System (DSS) that is considering different concerns regarding the relevant decision points in the field of middleware benchmarking is unavailable. A convenient concept that focuses on the involvement of application developers and stakeholders in regards to classifying substantial decision points is needed.

The main contribution of this Master's thesis is the first DSS for middleware benchmarking. We acknowledge the necessity of having an overview of all available middleware benchmarks, in order to filter the relevant information. To get an in depth knowledge of middleware benchmarks, we focused on standardised benchmarks that were published on industry-accepted consortia like Standard Performance Evaluation Corporation (SPEC) [SPE95] and Transaction Processing Performance Council (TPC) [TPC92c]. With the help of verified middleware benchmarks we were able to obtain the main structure and investigate the behaviour of these benchmarks.

To summarise, the primary research objectives of this Master's thesis is referring to the *elaboration, refinement* and *modeling* of the functionalities involved in the *DSS4MiddlewarePBenchmarking*.

Research Objective	**Description**
RO. 1	*Outline the current trends on middleware benchmarking and Decision Support Systems in regards to benchmarking*
RO. 2	*Identify decision points that are relevant for the construction of new middleware benchmarks*
RO. 3	*Analyse appropriate requirements for the implemetation of the Decision Support System*
RO. 4	*Deliver a prototypical implementation of the Decision Support System*

1.2 Thesis Outline

The remaining document follows the structure visualised in Figure 1.1. Chapter 1 provides a brief description of the covered topics to the reader and mentions the problem statement. Chapter 2 specifies the key concepts and terminologies required to get an elementary understanding of the covered topics. Then, in Chapter 3 the positioning of this thesis with already existing and relevant work is provided. It also mentions the different research approaches and associated trials faced. The main functional and non-functional requirements are covered in Chapter 4. This Chapter also identifies the set of functionalities supported by the system. A brief system overview is also listed. Moreover, it illustrates the taxonomy conducted through the literature review. Chapter 5 concentrates on the detailed description of the system architecture including UML diagrams and the REST architectural paradigm. Based on the previously mentioned taxonomy, the realisation phase of this Master's thesis is handled in Chapter 6. Used technologies and frameworks, as well as challenges faced while implementing the application and performing configurations are also listed in this chapter. In Chapter 7 the validation of the created web application is carried out with the help of queries and screen shots of the system. Finally, in Chapter 8 the overall summary is provided by stating the solutions for the research questions that arose initially. Furthermore, future extensions and enhancements of the system are suggested.

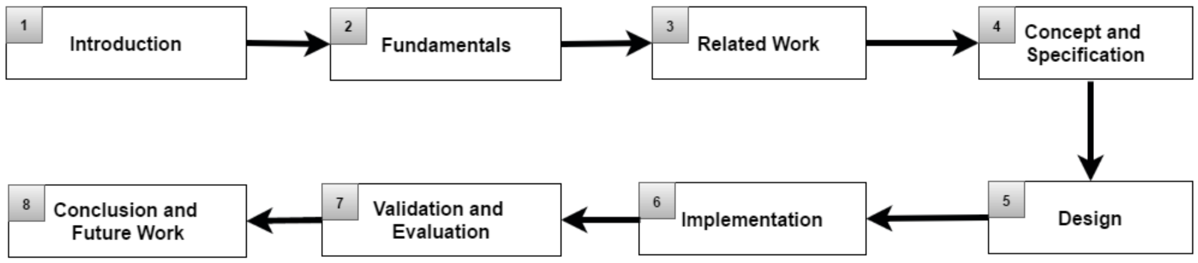

1
Introduction
2
Fundamentals
3
Related Work
4
Concept and Specification
5
Design
6
Implementation
7
Validation and Evaluation
8
Conclusion and Future Work

Figure 1.1: Thesis Outline

2 Fundamentals

2.1 What is Middleware?

Vinoski [Vin02] stated that with the expansion of computing and the number of devices and computing solutions, the heterogeneity of the systems has grown considerably and developers must address new and increasing integration problems. In order to make the systems work together in a homogeneous environment, it is necessary to integrate previously independent applications, together with new developments. The integration process deals with legacy systems. However, legacy systems can only be accessed via a distinguished interface and these systems do not permit any modifications. This development calls for a new integration solution for each component incorporated within the system. Therefore, the number of solutions and the number of complexity increases exponentially with the total amount of components in the system. Furthermore, with the modification of one component, all components that are associated to it must be adjusted accordingly in order to keep the system working.

A majority of systems form a compilation of separate devices associated by a network. Each specific device executes a function that invokes a local communication with the real world and a remote communication with diverse systems. The field of implementation cover decentralised manufacturing components, computer networks, telecommunication schemes and continuously running power supplies. Krakowiak [Kra03] suggests that the previously discussed problem can be solved with the usage of middleware systems (see Figure 2.1). An effortless application development can be granted to the application developers and integrators by contributing an universal programming concept, by covering the heterogeneity and distribution of the fundamental hardware and operating systems and lastly, by concealing low-level programming elements.

All applications employ intermediate software that is built above the operating system and communication protocols with the goal to achieve the following functions:

1. Distribution remains unseen.
2. Heterogeneity of the different hardware parts, operating systems and communication protocols remain unseen.
3. Homogeneous, effective and standard interfaces of the applications allow reusability, portability, easy composition and interoperability for developers and integrators.
4. Preventing the struggle of duplication and ease of cooperation between applications is achieved by having a number of common services that carry out different functions.

The dimensions of middleware can be distinguished as follows [Tig12]:

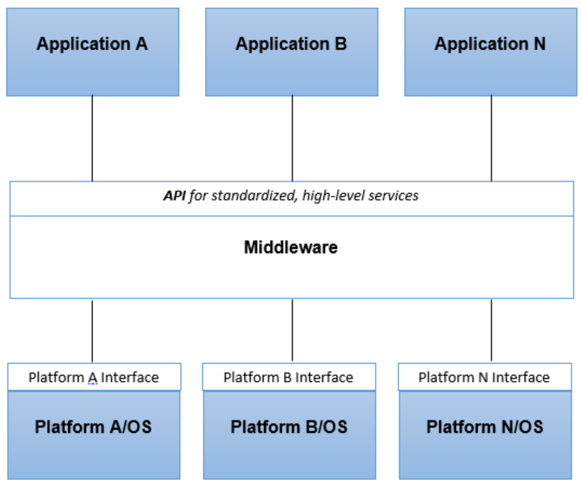


Figure 2.1: Middleware layer between the hardware layer and the application layer

1. Tight vs. Loose Coupling
2. Small vs. Large Ranged
3. Language-Based vs. Language Independent
4. Synchronous vs. Asynchronous Communication

There are various approaches to enforce middleware systems in order to deal with the integration question. The **different types** of middleware are described as follows [Tig12]:

2.1.1 Remote Procedure Call (RPC)

Marshall [Mar99] describes that the usage of RPC allows a distributed, synchronous communication based on servers and clients. It is an extension to the local procedure call where the called function can be located in a different address space than the calling function. Both processes can either reside on the same system or may be distributed among different systems with a network connection between them. Due to the fact that the RPC is transport independent, the physical and logical aspects of the data transmission structure is hidden from the application. Moreover, applications are given the opportunity to use a collection of transports. A RPC is comparable to a function call, where the calling parameters are sent to the remote procedure and the caller expects a reply back. The communication with RPC between two networked systems is carried out when the client program sends out a request, in the role of a procedure call, to the server and waits. Processing of the thread is haltered until either a response comes in, or a termination takes place. Upon arrival of the request the server executes it with aid of the requested systems, and forwards the reply to the client. The client program runs normally after the RPC call is wrapped up. Examples for an RPC include connecting to a network file systems with computers that are not in possession of a hard drive. Furthermore, when accessing a printer through a network, the computer indicates to

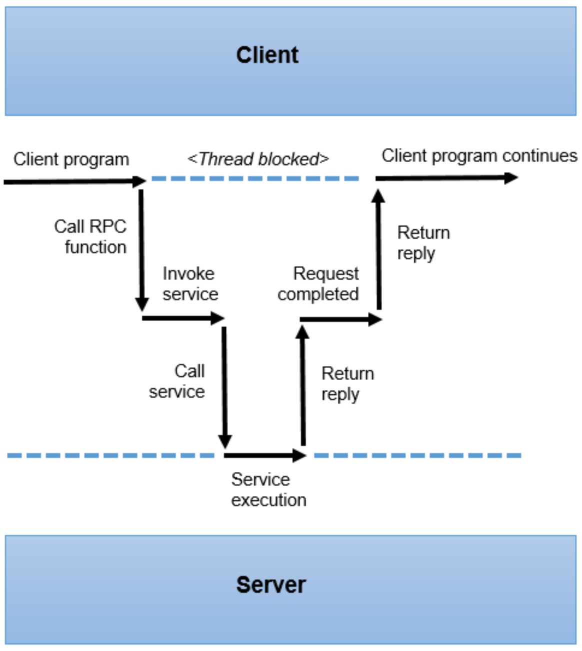


Figure 2.2: Activity flow during a RPC call between two networked machines

the printer what files and documents to print via an RPC. In Figure 2.2, the activity during a RPC call is illustrated.

The main characteristics of RPC [The97]:

Dynamic binding betweeen the calling function and the called function is important as binding mostly revolves around the communication method between the client and server, where the client institutes the binding via a specific protocol to a explicit endpoint and hosting system.

Speculation about shared memory cannot be made due to the fact that RPC's using input and output parameters acquire copy-in and copy-out syntax. With the usage of copy-in copy-out the specified reference is exclusively for the caller. When executing a function call its parameters should not be reachable by another execution thread, if this is not the case the parameter content has to be copied to a new reference not accessible by other threads. After receiving the reply from the function call the contents of the new reference are restored to the initial reference.

Individual failure incidents due to detachment of different systems. The concerns that arise with physically distributed machines are naming and binding issues, security concerns, protocol variance, remote system errors and issues with the network connection.

Contruction of a security framework, dependent on the used security protocols between server and clients, is necessary since interactions between physically distributed machines raise additional security problems. Moreover, further features for an access mechanism are required.

2.1.2 Message-Oriented Middleware (MOM)

Message-Oriented Middleware (MOM), which is a particular category of middleware, allows communication between distributed software segments in a loosely coupled manner. This is achieved, by applying the asynchronous communication pattern [SKBB09]. The methodology behind MOM is that a middleman takes care of all incoming messages and sends theses messages out to perhaps various message consumers. With the introduction of MOM message producers can simply forward the message to MOM without having to stop processing in order for the message to be processed and delivered, as MOM takes care of it. The main advantages of decoupling the communication participants include that, firstly, the consumers and producers can be unaware of each other, secondly, both parties do not have to stay active throughout the communication, and lastly, they are not occupied when they send or receive messages.

The standard interface to interact with enterprise MOM tools is called the Java Message Service (JMS). The MOM server that provides the JMS API is called the *JMS provider* and applications that interact with the JMS provider by exchanging messages are labeled as *JMS clients*. There are two different types of clients, one that produces messages and one that consumes messages. The JMS offers two messaging patterns: *publish/subscribe (pub/sub)* and *point-to-point(P2P)* [SKAB09]. In P2P messaging a virtual communication channel is introduced, the message queue. In this messaging model, messages are transmitted to a specified queue and later, consumed and processed by a single receiver. However, with pub/sub, messages are published to a *topic* and are retrieved from possibly multiple consumers. Each consumer subscribes to the *topic* he is interested in, only then published messages can be received. The party that provides messages is called *publisher* and the consumers that retrieve and process the messages are called *subscibers*. The terminology for queues and topics is referred to as *destinations*. There are different modes for the delivery of messages in JMS. Each of them is defined in the JMS specification and includes different QoS aspects [SKBB09]:

Persistent/Non-Persistent With the persistent mode of message delivery, messages are stored in a persistent storage, while they are waiting to be delivered. If a server crash would take place, all messages that were not delivered are retrieved from the stable storage and are ensured a *once-and-only-once* delivery. With the non-persistent mode, all pending messages are stored in main memory buffers which is not a persistent storage. The advantage with this delivery mode is, that there is only a low overhead of messages. The downfall is, that all pending messages are lost if the server crashes. When using the non-persistent message delivery mode, all messages are ensured a *at-most-once* delivery.

Durable/Non-Durable There are two different kind of message subscriptions in JMS. With the usage of durable subscription, the subscriber receives all published messages, regardless of being active or inactive. Non-durable subscription, on the other hand, only lets the subscriber retrieve published messages when being active. Messages that are published during the subscribers absence are failed to be delivered.

Transactional/Non-Transactional Each messaging session in JMS can be either transactional or non-transactional. Transactions are defined as a combination of messages that are implemented as a single atomic entity of work. A transaction can be split into two different groups: local and distributed. Transactions that are local can only operate with messages that are implemented on the JMS server. Transactions that are distributed are less limited and also permit additional operations.

2.1.3 Workflow Management System (WfMS)

Workflow Management System (WfMS) is referred to a piece of software that provides an environment where sequence of tasks, namely workflows, can be setup, executed and monitored [T$^+$04]. Orchestration is an important function of a WfMS, it coordinates the operation of the separate components that comprise a workflow. The theoretical ground workflow management is based on is the mathematical concept of petri nets. Workflows are made up of a task and dependencies between them. For the initialisation of a task the associated dependency condition has to be fulfilled. The main usage of WfMS is the transparent planning and control over every part of the enterprise. The main focus is on the interaction between different entities where information is shared.

2.1.4 Enterprise Service Bus (ESB)

The Enterprise Service Bus (ESB) is a message-based software architecture model that is used for the design and implementations of interactions between software applications in a Service-Oriented Architecture (SOA) [Men]. This integration is achieved by setting rules and principles for integrating a number of applications together over its bus infrastructure. Its main adoption is established in Enterprise Application Integration (EAI) of complex and heterogeneous platforms. The main aim of an ESB is to present a simple and consistent interface to end-users. Furthermore, it provides routing, invocation, and mediation services in order to reliably handle the interactions of applications and services [Men].

2.2 What is Benchmarking?

A benchmark describes the quality measurement of an organisation's products, programs and strategies. It also engages in comparing key metrics of their operations to other similar companies. With the use of benchmarks companies become more competitive. The comparison with other companies allows to see the areas where the system is under-performing. The main objectives of benchmarking can be described as follows [Inc16]: 1) Detection of required improvements and the area where deficiencies are present 2) Inspection of the key to high performance levels of other organisations 3) Enactment and application of the gathered information to improve performance The main characteristics for a good benchmark can be summarised as follows [Hup09]:

1. **Repeatable** - Despite the fact of frequent growth and occurring changes of the underlying technology, it is necessary to guarantee repeatability and consistency throughout the benchmark.

2. **Portable** - The ability to have a single application that can support a broad spectrum of other technologies. In practice, this cannot be achieved completely, however, benchmark technologies can make different adjustments that provide a sufficient solution.

3. **Verifiable** - The results provided by a benchmark should have a high level of confidence, meaning, they should be verifiable in terms of representing the actual performance of the System Under Test (SUT).

4. **Economical** - To make sure that the developed benchmark is worth its investment. With the growth of the complexity in computing systems, the supporting equipment become more costly. However, the high cost of supporting tools in benchmarks is counted as one of the main factors for a rejection of benchmark publishes.

In Figure 2.3 all steps included in a benchmarking process are defined [EK97]. The *planning phase* is the starting point, where all activities that need to be benchmarked are identified. The chosen activities should be measurable and easily comparable. Another activity this phase includes is choosing the appropriate benchmark against which the performance can be measured. The second step involves collecting data from the company that is providing the benchmark. Once a sufficient amount of data is collected, it is important to apply extensive analysis methods on this information. Some of the steps included are: data analysis, data presentation and identifying performance gaps in the processes. The *implementation phase* involves correcting the current state and aiming to reach to the expected state. A brief action plan should also be defined that is describing the changes required and assuring that a sufficient amount of resources is available in order to facilitate the changes. Finally, the *monitoring phase* manages the evaluation of the benchmarking process at a regular basis and takes care of necessary adjustments.

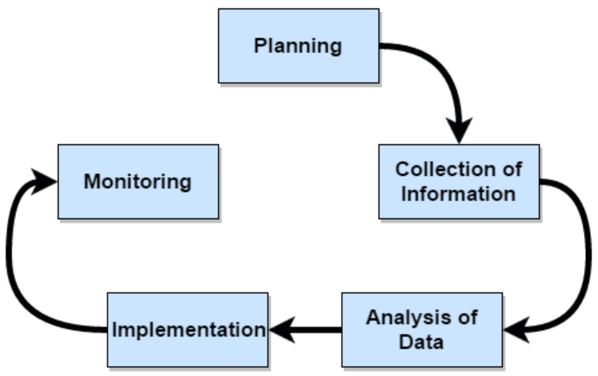


Figure 2.3: Steps Involved in the Process of Benchmarking

2.2.1 Standardised Middleware Benchmarks

There exists a number of organisations that provide standard benchmarks. Two most relevant ones that focus on performance benchmarking are known as the Standard Performance Evaluation Corporation (SPEC) and Transaction Processing Performance Council (TPC).

According to the specification provided by SPEC [SPE95], SPEC was created from few vendors who saw the lack of genuine and uniform performance tests in the marketplace. The aim of SPEC is to collect a valuable and straightforward set of metrics to separate potential systems. A reliable test that is feasible to employ will provide valuable results to the marketplace. SPEC provides the users with a predefined batch of source code to run on the system under test. Later, the benchmarker can adjust the system according to own preferences.

According to the specification provided by TPC [TPC92c], TPC focuses on defining transaction processing and database benchmarks and aims at providing verifiable performance data to the industry. TPC regards a transaction to be a commercial exchange of goods, services or money. Furthermore, a normal transaction, according to TPC, includes the updating of a database system. The following sections summarise the most relevant benchmarks published by these vendors.

2.2.2 SPEC JMS® 2007

The SPEC JMS® 2007 benchmark provides the assessment of performance for Message-Oriented Middleware (MOM) servers based on JMS (Java Message Service). With the standard workload and performance metrics it allows the benchmarker to achieve a competing product observation. Moreover, it provides a comprehensive performance analysis of enterprise messaging platforms. The performance analysis covers all components of the application environment made up of the hardware, JVM software, database software, system network and JMS server software. The benchmark offers two different dimensions for the metric; the horizontal and vertical topology will be discussed in depth in the following.

MOM, discussed in 2, is a relevant technology in many sectors, such as supply chain management, stock trading, online auctions and so forth. Furthermore, the publish-subscribe feature of MOM is considered as the main ingredient in the area of latest software architectures and technology, such as Service-Oriented Architecture (SOA) and Enterprise Application Integration (EAI). Nevertheless, innovative message-based applications undergo issues regarding scalability and performance.

Bacon et al. [SKBB09] identified that most newly established event-driven management for supply chain is immensely dependent on backend systems that support scalability and efficiency. This will ease the effort to process obtained real-time data and the integration of this data with enterprise applications and business processes. Sizable merchants usually have a high message throughput, therefore the underlying MOM is expected to perform and scale in a dependable manner.

Goals and requirements

The main aim of SPEC JMS® 2007 benchmark according to [SKBB09] and [SPE95] is to support a standard workload and metrics in order to measure and evaluate how scalable and how well JMS-based MOM platforms perform. Moreover, it provides an adaptable scheme to analyse the performance of JMS. These points can be achieved when the following requirements are fulfilled. First of all, users should be able to associate the examined behavior to their own environments and applications. This is achieved by reflecting the workload in a manner real-life systems are exercised. Secondly, the workload should have a level of comprehensiveness, meaning that all platform features are provided that are usually used in MOM, such as pub/sub and P2P. Thirdly, the workload should explicitly target the performance and scalability measurements of the MOM server's software and hardware units. The workload should lessen the influence of other services and units that are involved in the application scenario. Lastly, there should not be any scalability limitations within the workload of SPEC JMS® 2007. Users are able to increase the number of destinations (queues and topics), the number of messages per destination can be increased or users can scale the workload in a customised manner. The purpose of using this benchmark varies. It can be used for marketing reasons by publishing and producing standard results. Also, a large group of users has the intention to optimise and improve their platforms by analysing the performance of specific MOM components. Another purpose can be for academic research, where the scalability and performance is classified and could support the establishment of highly efficient MOM servers.

Scenario roles

The application scenario for **SPEC JMS® 2007** involves the model of a supply chain for a supermarket. The supermarket company, its stores, its distribution centers and its suppliers are the different participants that are involved in this scenario. The requirements discussed in the previous section are applied on this scenario. It allows a clear definition of interactions that stress defined features of the JMS Servers. For instance, pub/sub or P2P communication as well as diverse message types. Moreover, there are no limitations on scalability of the workload, the number of supermarkets can be increased and the number of products offered by a supermarket can also be increased.

2.2.3 SPEC-JBB

According to the specification provided for the SPECjbb®2015 benchmark [SPE15], it provides the performance measurement based on the latest Java application features. It is applicable to all organisations that are interested in measuring Java server performance. The benchmark includes a model that illustrates a supermarket company with an IT infrastructure that deals with point-of-sale requests, online purchases and data-mining operations. The metric included in the benchmark is a pure throughput metric. SPECjbb®2015 also supports visualisation and cloud environments.

2.2.4 TPC-C

According to [TPC92a] the TPC Benchmark C is an On-Line Transaction Processing (OLTP) benchmark. It is an improved version of the previously published benchmarks due to its multiple transaction types, more complexity in the database and the overall execution structure. TPC-C includes five concurrent transactions of different types and complexity. The involved database is made up of nine different types of tables with large sizes in regards to recording and population. The TPC-C benchmark is measured in transactions per minute (tpmC). It simulates a running computing environment where users execute transactions against a database. The transactions contain entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the corresponding warehouse. This benchmark is not restricted to a specific business area, but, rather speaks for any market sector that manages, sells, or distributes a product or service.

2.2.5 TPC-E

The TPC-E is an On-Line Transaction Processing (OLTP) benchmark that resembles the OLTP workload of a brokerage firm. The main target of the benchmark is the central database that executes transactions associated to the company's customer accounts. Despite the fact that the business model covered in TPC-E is a brokerage firm, the benchmark can be used on modern OLTP systems. The benchmark specifies the required combination of transactions it should be able to handle. The TPC-E benchmark is measured in transactions per second (tpsE). [TPC92b]

2.3 Performance Testing

Almeida et al. [DAM01] suggested that in order to determine the performance of Web services, performance testing has to be applied. Furthermore, the main reason for performing tests is to have a better understandability of the system when different workload conditions apply. With the usage of performance testing the system behaviour representing the activity of real-world users and scenarios can be constructed. There are three different types of performance tests [DAM01]:

1. *Load testing:* With this approach, the answer to whether a system is reaching its expected requirements is obtained. To achieve this, a simulated load is created for this purpose of mimicking the expected behaviour of operations.

2. *Stress testing:* This approach tests whether a system can cope with a load heavier than its performance requirements. Stress testing involves focusing on extreme conditions that are heavier than the expected load.

3. *Spike testing:* This type of testing is used for rare occasions, as the load is considerable higher than the average.

2.4 Decision Support Systems and Decision Support

Decision Support System (DSS) is referred to an information system that maintains decision-making tasks of a business or an organisation [PSB15]. DSSs contribute to the management, planning and operation of an organisation by providing decisions for problems of dynamic nature. The computation of DSS can be either fully computerised or human-based, there is the possibility of having a mixture of both approaches. The different types of DSS can be summarised as follows [PSB15]:

1. **Data-driven DSS** - provides access to large knowledge bases in order to extract information.
2. **Communication-driven DSS** - supports the shared access on a specific task where more than one person is involved in working on it.
3. **Document-driven DSS** - data is retrieved and manipulated in form of a document.
4. **Knowledge-driven DSS** - so called *expert systems* provide professional problem-solving in terms of defined rules and procedures.
5. **Model-driven DSS** - provide functionality by offering different models, data and parameters are provided by the user.

The three different components inside a DSS are, the knowledge base where all relevant information is stored, the model that defines different decision criteria and the user interface that presents the required output [PSB15].

2.5 Service-Oriented Architecture (SOA)

A Service-oriented Architecture (SOA) is a paradigm that organises a set of functionalities as specific services that are made available across the network in order to be accessed through assigned interfaces. In general, SOA offers the means to match different needs and problems with offered capabilities, by accessing and applying capabilities, and aggregating capabilities to meet requirements. Services interactions can be represented using a description language. SOA is also concerned with the concept of loose coupling and dynamic binding between services. Furthermore, a system that is built upon the SOA architectural style must provide *easy discoverability* of needs and capabilities and must provide a mechanism for the interaction between consumers and providers. [LL09]

Apart from being an architecture regarding services, SOA also includes three different participants that interact with each other:

1. Service provider
2. Service discovery
3. Service requester

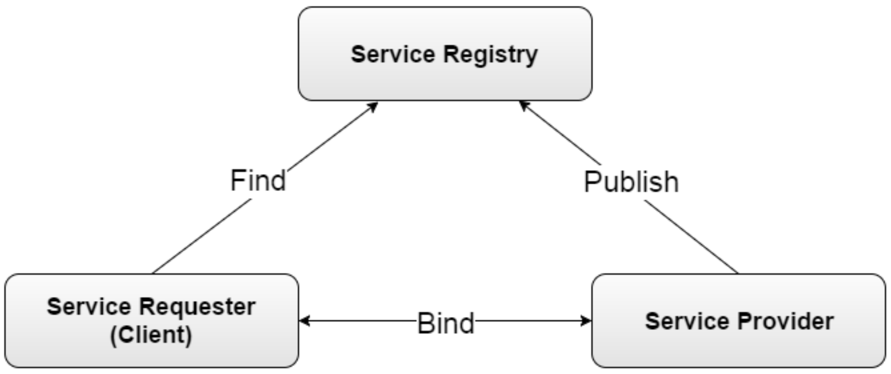


Figure 2.4: SOA Model

The operations that are used amongst them when interaction takes place are *publish, find*, and *bind* [BHM+04]. The tasks of a service provider involve hosting an application in form of a service that is accessible through a network and making it available for consumption. In order to identify these services, the service discovery agency offers a registry that contains a centralised library for the services. One example for a service registry is UDDI, further discussed in section 2.6.3. To sum it up, the service provider *publishes* the service in the registry, where the service requester can *find* it. With the help of a so called 'service description', the service requester can *bind* with the service provider in order to access the invoked service. The relationship of the three kinds of participants is illustrated in figure 2.4

The strong influence of SOA involves its beneficial attributes, such as, loosly joined services, high interoperability of services and increase of re-usability due to well designed services. SOA is a design principle that is not dependent on any technology or product. The possibility of offering "full" SOA "packages" is impossible because different organisations have different needs and requirements [Erl05]. SOA can be implemented using any service-based technology, however, the most common implementation of SOA is Web Services [LL09].

2.6 Web Services

A web service is known as any service that is at located at a specific endpoint in the web. Along with the evolution of web services, the maturity and potential of web services also increased. Therefore, the adoption of the SOA design philosophy was required. Even though most web services are based on the HTTP binding, the interoperability interaction of web services is achieved with standards that are based on the Extensive Markup Language (XML). The Simple Object Access Protocol (SOAP), the Web Service Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI) are known to be the three most popular web service standards [Ma05].

2.6.1 SOAP

The Simple Object Access Protocol (SOAP) is a protocol-independent, XML-based protocol to access available web services [Ma05]. SOAP provides the means to interact with applications that are located on different operating systems, that use different technologies and programming languages. For the transport mechanism SOAP supports HTTP or the Simple Mail Transfer Protocol (SMTP). The message encapsulation format of SOAP includes an *envelope*, unlimited number of *Headers* and only one *body*. The user's XML data is included in the *body*, whereas information about the service to be invoked is stored inside the *envelope*.

2.6.2 Web Service Description Language (WSDL)

WSDL is known to be the standard for service descriptions that are based on XML [Ma05]. The full WSDL description consists of an application-level service description and the concrete transport-dependent information. When using WSDL descriptions, requesters are able to understand and access the corresponding service at specific service endpoints. The two different aspects of the WSDL description exist due to the fact that similar application-level service capabilities are often deployed at different endpoints and have contrasting access protocol elements.

2.6.3 Universal Description, Discovery and Integration (UDDI)

UDDI defines a registry of web services and provides users the possibility to systematically find service providers and technical information about services. It is equivalent to numbers in a phone book [Ma05]. Service providers can dynamically insert services in the registry which can later be exploited by the service requester. When the service requester uses a service's general description, a list of matching services is retrieved from the registry. To be precise, the UDDI registry preserves a centralised database of services along with information about its location, the service type, and provider information. Furthermore, a standard API for users to query the database is provided. The application of UDDI internally in organisations has a better use, as users can query the registry using well-known identifiers for required services. When applying UDDI in a larger scope where human requests have to be matched to human descriptions of a service, human interruption is inevitable [Ma05].

2.6.4 RESTful Web Service

When client programs want to interact with devices , databases and web services, the Application Program Interface (API) is used [Mas11]. This API can be seen as a doorway between the client program and server program to access the underlying data of exposed functions. To form and design these APIs in order to facilitate the exchange of information, the Representational State Transfer Protocol (REST) is used. REST was first introduced in 2000 by Roy Fielding at the University of California, in his dissertation, "Architectural Styles and the

Design of Network-based Software Architectures" [Fie00]. The benefit of a well-designed REST API for web service developers and providers is the increasing interest of service clients. Representational State Transfer (REST) is an architectural style that follows a set of concepts and constraints. Therefore, web services or architectures that comply to these REST constraints are known to be *RESTful*. This architectural style is used to design networked applications and provides simple HTTP to make calls between different machines. Rather than using complex mechanisms like SOAP- and Web Service Description Language (WSDL), REST offers a lightweight and resource-oriented approach to expose services. The architectural constraints mentioned in [Fie00] include:

Client-server: In order to achieve successful communication between entities, established standards have to be followed. This allows the web communication system to bind these entities. In the case of Client-Server interaction, despite of both parties being independently implemented and deployed and using different languages or technologies, they have to follow the established standards so communication between them can take place.

Uniform Interface: The need for uniform interfaces in regards to the interactions between the different web components is extremely important. If the uniformity of the interfaces between server, client and network interactions is not achieved, the communication system is likely to fail. The following four constraints accomplish the uniformity of interfaces:

1. Identification of resources - With the definition of a global address space for the purpose of resource and service discovery.
2. Manipulation of resources through representations - A particular resource can appear in different representational styles to serve the needs of different clients.
3. Self-descriptive messages - Messages to be self-descriptive so that different representations can be extracted from them. Therefore, messages contain meta-data that defines the resource state as well as the representation format and size.
4. Hypermedia as the engine of application state (HATEOAS) - This is achieved by stateful interactions using hyperlinks. The state of a resource has to be included inside the message itself rather than keeping it on the server.

Stateless: With the enforcement of this constraint the client is required to keep all the information that is necessary when interacting with the web server. With this approach the server-side burden is reduced and scalability is facilitated

Cache: With the adoption of caching, some interactions are no longer required. This reduces the latency of interactions and leads to improved efficiency, scalability and performance. The caching of responses leads to high availability of responses. In the case of dynamic scaling of resources, the cached data can effortlessly be assigned to newly created resources.

Layered System:

The approach of layered systems increases loose coupling across different layers by exposing the internal layer exclusively to its immediate neighbouring layer and hereby enabling improved evolution and reuse. The application of layered systems allows a hierarchical layering where component details are disclosed to intermediate layers with which the interaction takes place. With the restriction of exposing information strictly to a single layer, the system complexity is lessened and the self-sufficiency is encouraged.

Code-On-Demand:

Clients expect to understand and use the required code that is located on the server, therefore a technology coupling between client and server is mandatory. The only optional constraint of the REST paradigm is Code-On-Demand.

The architectural features that are supported by REST consist of [RESb][Mas11]:

1. Performance - Caching closer to the resource is empowered since Resources are uniquely identified using a Uniform Resource Identifier (URI). This also leads to a rapid identification of resources. A resource can be described as a block of information that is identified using a Uniform Resource Locator (URL). A resource can have different types usually it is interpreted as a file, a query result generated dynamically or a document.
2. Scalability - With the help of this feature the growth in demand can be addressed. The application of caching would permit to sidestep the origin server. The request itself includes all the needed state information. Furthermore, the request is not dependent on any particular server, meaning that the request can be fulfilled by any server that can provide the required cache content. The request is spread across numerous servers which leads to high scalability of the system.
3. Simplicity - For the implementation of a simple and understandable application it is required to separate the concerns. The four verbs Create, Read, Update and Delete (CRUD) discussed in section 2.6.5 enable an easy implementation.
4. Modifiability - This feature describes the ease at which changes can be done to an architecture. REST also includes the feature of evolvability, extensibility, customisability and re-usability.
5. Visibility - The enforcement of monitoring and regulation of interactions between parts of an architecture, by another part of the architecture. This can be achieved by introducing agents that keep track of information passed between services and clients.
6. Portability - All services must be platform and technology independent. This means, that the application must be compatible to run on heterogeneous environments and platforms. This approach increases the aspect of easy usability.
7. Reliability - This feature is described as the level of vulnerability to failure of services and solutions within a system. In order to increase reliability inside an architecture, it is necessary to avoid single points of failure. Furthermore, failover methods and the monitoring of features that can dynamically address failure situations should be enforced.
8. Data Independence - REST enables content negotiation where one resource can be formatted in the capabilities the clients can offer. This allows to use the required format for a resource.

RESTful API Design

The means to access web services is referred to the Application Programming Interface (API). Client programs are able to access the APIs that offer a set of functionalities that make the interaction between computer programs possible [Mas11]. To be precise, client programs communicate with web services through an API, where these web services are made available over the web such as Amazon Simple Storage Service (S3) [1] that supports storage capabilities. Therefore, a web API that is according to the REST architectural style is referred to a REST API. When designing a REST API many principles have to be taken in account. Some practices follow the HTTP standard, however, due to flexibility in the design of APIs, the design can be oriented according to the functionalities that the service offers. The API points to a resource, that means in order to access the resource a URI has to be designed. Furthermore, the possibility to perform different actions on the resources, with the use of HTTP methods, should be granted.

API Design Standard

In order to achieve consistency and understandability, a number of rules have been specified. These rules contribute to a standardised and clean API, offered to clients. Moreover, designers are provided with a guideline to follow when designing the API and do not fall into confusion and doubts. The majority of these rules are considered to be a standard while some of the rules have a freedom for slight modifications. The design rules for the construction of URI scheme [Mas11][Agr15]:

1. Hierarchical relationships must be indicated using forward slash separators (/).
2. When naming the resources a forward slash should not be followed (/my-resource/).
3. When defining URI paths lowercase letters should be used throughout.
4. Selection mechanisms provided by HTTP should be used by REST clients rather than including file extensions in URIs. Users should be encouraged to clearly mention the expected response format.
5. The segment separated by forward slash must point to a unique resource within the resource model of the REST API.
6. Document names [2] have to be composed using a singular noun while collection names [3] and store names [4] should be written as a plural noun.
7. The CRUD operations should not be used in URIs.

[1]https://aws.amazon.com/s3/

[2]A document name is pointing to a database record or an object instance e.g. `http://api.fruit.restapi.com/types/apple`

[3]A collection resource is referred to a directory of resources e.g. `http://api.fruit.restapi.com/types` - where the collection is referred to as *types*

[4]A store is referred to a repository created by clients e.g. `PUT/fruits/3456/favourites/apple`

8. The inclusion of query components inside URIs allow for further filtering of collections or stores. This is done by including a key-value pair in the query that specifies the exact criteria that the response message should contain.

9. It is not allowed to use GET and POST methods to tunnel other request methods meaning that they should not be used incorrectly as it would limit the client with less HTTP terminology.

10. The method GET is used to retrieve a resource representation, PUT allows both, the insertion and update of resources. It also provides a request message that defines the expected changes. POST enables the creation of a new resource inside a collection. Finally, DELETE simply removes a resource from its source.

11. It is not allowed to change the behaviour of HTTP methods using custom HTTP headers. It is recommended that a specific resource is used to process a request using the required HTTP method. The following should be avoided: POST /fruit/apple/colour HTTP/1.1 X-HTTP-Method-Override: PUT.

The rules mentioned above are crucial for the design of REST APIs in general. For that reason, we considered the application of these rules on our REST APIs in Chapter 5 of this thesis.

2.6.5 Hypertext Transfer Protocol (HTTP)

HTTP is known to as a stateless protocol that is responsible for the transfer of representations of resources over a network [F^+99a]. HTTP is an underlying network protocol used by the World Wide Web (WWW). It defines concerns about message formatting and transfer in a precise manner. Furthermore, it provides a clear explanation for web servers and browsers on how to respond to different commands. When a URl is entered in the browser, a HTTP command is sent to the web server. This HTTP command instructs the web server to retrieve and transmit the requested web page. REST and HTTP are closely associated to each other where the CRUD operations (discussed in 2.6.5) are used together with REST. REST is not fully restricted to the web but when it is used in the implementation of web-based systems, it uses HTTP for interactions. HTTP executes each command independently without the need to know the previous command or request, therefore, HTTP is known to be a stateless protocol.

HTTP Status Code

HTTP status codes can be described as "standard response codes" which are displayed by the server/client when an error takes place over the network during interaction. The main purpose of HTTP status codes is to identify the cause of a problem when the web page is not loading as expected. The HTTP status code is a combination of HTTP status code and HTTP reason sentence, this combination is also known as the HTTP status line. The purpose of the reason phrase is easy readability, where the user is provided with a short description of the status code in text format. In general, the status codes demonstrate if a particular HTTP request has been processed successfully. There are five different categories for responses: informational response, success response, redirection, client error and server error. In the following Table 2.1, a list of some basic status codes is illustrated:

Status Code	Status Text	Description
100	Continue	The server has received the request headers, and the client should proceed to send the request body.
200	OK	The request is OK (this is the standard response for successful HTTP requests).
201	Created	The request has been fulfilled, and a new resource is created.
204	No Content	The request has been successfully processed, but is not returning any content.
303	See Other	The requested page can be found under a different URL.
401	Unauthorized	The request was a legal request, but the server is refusing to respond to it. For use when authentication is possible but has failed or not yet been provided.
404	Not Found	The requested page could not be found but may be available again in the future
500	Internal Server Error	A generic error message, given when no more specific message is suitable.
501	Not Implemented	The server either does not recognize the request method, or it lacks the ability to fulfill the request.

Table 2.1: HTTP Status Codes [W3S]

HTTP Methods

Data and functionality of resources in RESTful web services is exposed using URIs. These URIs use the CRUD actions which are the four HTTP methods. In order to manage and modify resources, GET, POST, PUT and DELETE is used. These methods allow to create, retrieve, update and delete resources. The "uniform interface" constraint is primarily achieved by these HTTP verbs. A detailed description of these HTTP methods is provided below [F+99a]:

HTTP method	**CRUD action**
GET	Retrieve a resource
POST	Create a resource
PUT	Update a resource
DELETE	Delete a resource

Table 2.2: Association of HTTP verbs with the corresponding CRUD action

PUT

The HTTP PUT method is mainly used to update existing resources. This means, to put new capabilities to a known resource URI with the information about these newly added capabilities in the request body. However, when the resource ID is chosen by the client instead of the server, the PUT method can also be used to create new resources. In other words, the PUT is pointing to a URI that incorporates the information about a non-existing resource ID. When the modification is successfully undertaken, the status code 200 or, when no respond is expected, 204 is outputted. In the case of using PUT for the creation of resources, the status code 201 is displayed upon success (see tale 2.1 for more details). As the client already set the resource ID it is not required to include the link inside the location header when the resource is created successfully. PUT can modify or create state on the server, therefore, it is regarded to be unsafe. Additionally, PUT is considered to be idempotent, which means when a resource is updated or created any number of following identical calls won't have an impact on the state of the resource. There are some cases where the PUT method is not regarded as idempotent due to underlying logic. However, it is suggested to have idempotent PUT methods. For non-idempotent request the POST method should be used.

POST

The main pupose of the HTTP POST method is the creation of new resources. Most of the times the creation involves resources that are dependent on their parent resource e.g. middleware contains different *types* of middleware. Therefore, when a new resource is created the service makes sure that the new resource is associated to its parent by using and ID and creating a new resource URI. When created successfully the status code 201 for CREATED is displayed indicating that the request has been fulfilled and a new resource is created. Along with the status code, a location header is also returned containing the link to the newly-created resource.

GET

The HTTP GET method is used to retrieve or simply read a representation of a resource. When applying the GET method and no error occurs in the result, a representation in XML or JavaScript Object Notation (JSON) is returned. Additionally, the HTTP response code 200 for OK is returned. In the case of client-side errors, either a status code of 400 for BAD REQUEST or 404 for NOT FOUND is returned. In the case of a server-side error, 500 for INTERNAL SERVER ERROR is outputted. According to the HTTP specification, using GET together with HEAD, the modification of data is not permissible. This means that using both as a combination, no risk of data corruption or modification occurs. Furthermore, they are considered to be idempotent, multiple identical requests are considered to have the same result as a single request.

DELETE

DELETE is used in order to remove a resource accessed by its URI. When a resource is deleted successfully, the HTTP status 200 along with a response body is returned. The response body contains the representation of the deleted resource. In the case of no associative content in the requested URI, the status code 204 is returned. According to the HTTP specification, DELETE is considered to be idempotent. A single deletion request removes the resource completely, therefore, multiple calls of the same kind won't make an influence.

3 Related Work

This chapter introduces the main research fields related to this thesis, in order to assure a common comprehension. In Section 3.2, we introduce recent techniques on middleware performance benchmarking and components of middleware benchmarks. In Section 3.3, we summarise web-based Decision Support Systems (DSS) in general, different existing DSS and their offered functionalities. Subsequently, a short summary of the State of the Art in decision support for middleware benchmarking is provided.

3.1 Benchmark Architecture

The vast and highly changing nature of the computing industry constantly requires new benchmarks. According to Bacon et al. [SKBB09] to guarantee that applications are reaching the Quality of Service (QoS) obligations, it is necessary that the underlying platforms are tested with benchmarks, so that the performance and scalability can be measured. There are numerous characteristics that have to be met by the benchmark in order for it to be beneficial and dependable [Hup09]. Firstly, the stressed platform has to represent a real-world application. Secondly, it must operate all crucial services that a platform offers and it should provide some space to carry out performance comparisons. Lastly, the results of the benchmark must allow reconstruction, so that the benchmark is reliable. It should also prevent any implicit scalability constraints [A$^+$15]. Additionally, in order to provide the means for performance comparison, a benchmark needs to span over a considerable time period [Hup09].

One important aspect in terms of benchmarking is the correct use of the benchmark results. There are many projects that get devalued results due to generally known and avoidable mistakes [A$^+$15]. Arnold et al. [A$^+$15] discovered that there are various benchmarks that use an artificial workload which can easily lead to faulty or useless results, as it is challenging to determine information about the performance of real applications from the outcome of suchlike benchmarks. Therefore, Leymann et al. [SRL$^+$15] suggested that an appropriate workload should be found, that is comparable to realistic workload, in order to regulate this issue.

Additionally, results have to be understood in the correct meaning. It is important that the obtained results express the measured characteristics and do not lead to intervention of other features. This can be achieved by letting the system under test reach to a steady state. Another important guideline is that comprehensive configuration data is issued along with the benchmark outcome. With this information different configuration adjustments can be made. Moreover, middleware designers should be motivated to provide approved settings for particular applications and used workloads. This would lead to publication of correct and relevant benchmark results.

Brebner et al. [B$^+$05] identified that some of the concerns in regards to benchmarking is the challenge of resources required to operate a benchmark, also required devices that will perform the work, the temporal length for running the benchmark, and know-how needed to comprehend the entire benchmark configuration. Most often the resource requirements are excessive for benchmark conduction. In addition, the lifetime of the proficiency, as well as the results is limited. For that reason, the outlay of the benchmark is expanded.

Lastly, another concern is how to manage the publication of benchmark results for auditing and review. Sensitive information could arise new issues for the specific party or it could lead to misinterpretation of information.

A good approach would be to maintain a knowledge base that would store all issues regarding benchmarking, so engineers can avoid mistakes beforehand. Even with this idea difficulties emerge, such as updating technique of this knowledge base. It would allow engineers to view their benchmarking results in different perspectives. This kind of database would involve concepts to solve concerns on reliability, credibility and anonymity of the benchmarking outcomes. The associated complexity of universal data format needs to be determined in order to ease the contribution of the results and engines to deal with them. [B$^+$05]

Furthermore benchmarks can be a great support for monitoring and auditing reasons. Some of the two main areas where benchmarks are applied are listed in the following:

1) **Design of Middleware** Benchmark can be used to construct the ideal middleware architecture with regards to the available constraints such as memory capability, processing power or network latency and throughput. The major aspect in this scenario is the effort to approve models developed during the design phase [B$^+$05].

2) **Assessment of Middleware** Another important use that can be achieved with benchmarking is to evaluate middleware. In order to achieve this, the performance of different middleware architecures and middleware configurations, is compared. Furthermore, it is important to classify the scalability of the middleware. With this classification the benchmarking performance of the middleware and its underlying systems can be indicated when the limit of scalability is reached. [B$^+$05].

3.2 Current Trends with Middleware Benchmarking

Despite the fact that middleware technology is widely spread there is still a lack of putting middleware benchmarking in practice [B$^+$05]. Some middleware developers use specific testing suites that make it nearly impossible to compare the output with diverse middleware implementations. Furthermore, middleware users employ simple testing suites where the output can lead to misconceptions. To face these shortcomings it is important to have successful standardised middleware benchmarks. The architecture of a benchmark is completely dependent on the intended handling of the results [B$^+$05].

These standardised benchmarks are made available by consortia that are accepted by the market. The major goal of a benchmark consortium is, to provide the marketplace with legitimate and beneficial performance metrics in order to examine the latest development of IT components [SPE95][TPC92c]. Moreover, these organisations supply benchmarks for system performance assessment in numerous application fields. The standard workload and metrics provided aid the user to measure and evaluate the performance and scalability of middleware platforms [SPE95].

The following fundamental prerequisites have to be met in exchange for a constructive and dependable benchmark [Kou06]:

1. The workload used in the benchmark should represent real-world applications
2. It must handle all crucial services offered by platforms
3. It must provide a customisable spectrum for performance observation
4. The result produced by the benchmark should allow reproduction
5. It must not have any restrictions on the scalability

Some copyrighted benchmarks are available and have also been adopted for testing and product comparison reasons, however, these benchmarks do not fulfill the requirements mentioned above [Act06][JBo06]. The reason for that is due to their usage of artificial workload that does not represent real-world application schemes. Additionally, their focus lies on stressing isolated features of the MOM rather than providing the overall Message-Oriented Middleware (MOM) server performance [SKCB07].

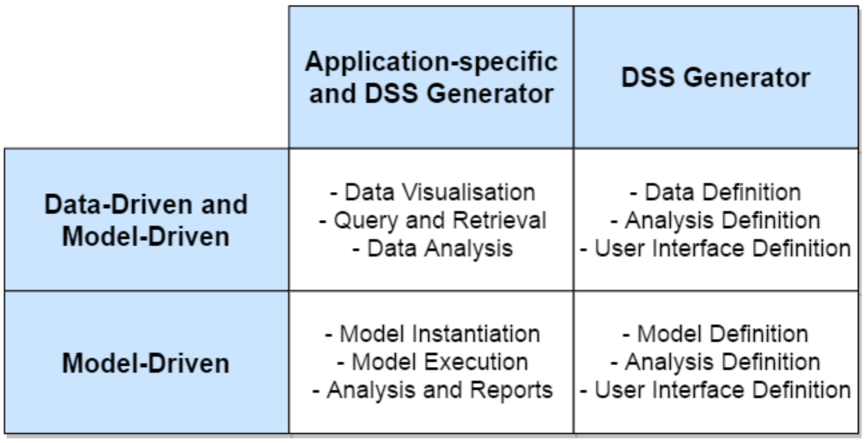
MOM is widely adopted on modern information-driven applications such as supply chain management that is event-driven, stock exchange and online auctions. In financial services and enterprise applications, the usage of MOM is broadly applied. In order to support these kind of applications in the commerce, the performance and scalability of the underlying MOM platforms, play a significant role [SKCB07]. Therefore, a standardised benchmark from the research and commercial aspect is necessary to assess the scalability and performance of MOM [SKCB07]. A middleware implementation constructs a settled component for many object-oriented systems. The scope of middleware implementations covers a wide range, as discussed in Section 2.1. A crucial aspect of these implementations is their performance under different circumstances and conditions. The results of this characteristic is beneficial for middleware developers in terms of enhancing and promoting their product and for middleware users in order to decide for the appropriate product [B$^+$05].

3.3 Web-Based Decision Support Systems

Decision Support System (DSS) are currently highly developed where a variety of functionalities is offered such as information collection, analysis, model construction, collaboration, alternative assessment and decision implementation [BPS07]. Recent technologies in the field of DSS consider the World Wide Web as their main platform for either providing DSS products or deploying DSS applications using the web browser interface as their client. The high demand in the field of web is calling for strong attempts to further broaden the development and implementation of web-based DSS in different sectors such as education, health care, private companies, and government [BPS07]. According to Power [Pow04], the most widely used approaches for DSSs are referred to data-driven and model-driven DSS. The main purpose of data-driven DSS is to aid the organisation, analysis and retrieval of large amounts of data. Model-driven DSS work with precise representations of decision models while providing analytical help. According to Bhargava et al. [BPS07] the three other categories of DSS have also gained popularity with the growth of web technologies. Communication-driven DSS depend on electronic communications between different parties in order to achieve decision-making. With the Web this type of DSS is granted with linking various decision makers that are separated in space or time. Knowledge-driven DSS provide people like managers suggestions and recommendations. Even this type is covered by the Web, where the audience is a much bigger number. At last, the document-driven DSS provides managers document retrieval and analysis by integrating storage and processing mechanisms.

The computation of DSS can be either Web-based or Web-enabled while different architectural styles are possible [BPS07]. One example could be the scenario where the entire data collection, decision models, algorithms and relevant information is stored on a Web server and is accessed via a Web browser. The other possibility is to embed all required components on the Web browser in order to support application specific decision support. For instance, the Network-Enabled Optimization System (NEOS) Server provides optimisation solutions over the Internet [CMM96]. NEOS receives optimisation problems from users and finds the relevant optimisation solver. After computing all supplementary information that is required by the solver, the initial optimisation problem is linked with the solver and the result is presented.

There are many technologies available that support the implementation and deployment of DSS. However, recent technologies used in the field of developing DSS involve Web services and messaging protocols like SOAP and XML-related languages [BPS07]. There are different tasks involved in the creation and usage of data- and model-driven DSS. With the usage of Web services these tasks can be performed on a Web client. According to Sprague [SJ80] there is a differentiation to consider between application-specific DSS and DSS generators. In the case of application-specific DSS, a concrete decision problem is solved with the help of a data, models and a user interface. DSS generators provide algorithms and different tools in order to build new DSS. Figure 3.1 defines the 10 major tasks that are involved in building and using data- and model-driven DSS [BPS07]. These DSS-related tasks that users can do through Web browsers involve model instantiation, model execution, creation of analyses and reports , data visualisation, query and retrieval, data analysis, model definition, data definition , analysis definition, and user interface definition [HSC99]. On the one hand, the Web can be used to facilitate DSS Information Platforms where information about vendors, methodologies and products related to decision support technologies are offered [Pow02]. Examples for this kind of decision support portals are DSSResources.com and DataWarehousing Online, where information regarding optimisation of DSS and decision analysis is offered. On the other hand, the Web can be utilised to provide application-specific decision support for



	Application-specific and DSS Generator	DSS Generator
Data-Driven and Model-Driven	- Data Visualisation - Query and Retrieval - Data Analysis	- Data Definition - Analysis Definition - User Interface Definition
Model-Driven	- Model Instantiation - Model Execution - Analysis and Reports	- Model Definition - Analysis Definition - User Interface Definition

Figure 3.1: Major Tasks involved in DSS

arising decision problems [BPS07]. This is achieved by vendors like Lumina[1] and TreeAge[2] that provide capabilities to develop products that generate Web-based application-specific DSS. Lumina, is specialised in selling desktop DSS generator that are based on influence diagrams, furthermore, it provides the Analytica Decision Engine that offers developers to create Web-based DSS applications [BPS07]. The TreeAge Software provides the means to develop Internet-based decision-tree applications. Another Web-based DSS for the purpose of structuring of analytical decision problems is the Web-HIPRE [3].

Currently, there are a number of web-based DSS available that serve different purposes. The two areas that the DSS research can be divided in is the field of architectures and technologies and, secondly, the field of applications and implementations. The DSS metadata model for distributing DSS on the Web was developed by Gregg et al. [GGP02]. The steps towards the development included to consider the capabilities of end users in order to find suitable resources on the Web. After conducting an experiment, the authors concluded that a metadata aids end users to find and comprehend a specific DSS functionality on the Web.
Bharati and Chaudhury [BC04] conducted an experimental research that pointed out important factors that are required to achieve decision-making satisfaction in the field of Web-based DS. The outcome of this empirical study reported that system quality and information quality had an high impact on customers' satisfaction, whereas information presentation did not have a significant impact on decision-making satisfaction. Bharati and Chaudhury suggested that developers of Web-based DSS should rather have their focus on system quality and information quality, rather than on information presentation. In terms of system quality the refinement should take place in regards to ease of use, convenience of access, and system reliability. The information quality can be enhanced in regards to relevance, accuracy, completeness, and timeliness. Guentzer et al. [GMMS07] applied Structured Service Models in order to help users to find information resources that are exposed as online services in the scope of an Intranet. Due to the fact that the computing of similarity between two models

[1] `http://www.lumina.com/`
[2] `https://www.treeage.com/`
[3] `http://hipre.aalto.fi/`

requires heavy computational processing, Guentzer et al. conducted a heuristic concept and attained the performance characteristics from computational tests.

There is a high level of ongoing research in the field of Web-based DSS case studies and developments of prototypical implementations [BPS07]. One example for a Web-based DSS implementation is implemented by Ngai and Wat [NW05], where a risk analysis for the e-commerce sector is provided. The prototype is a support for project managers that want to identify, analyse, and prioritise risk involved in e-commerce developments. The Web browser was used to present the DSS and the server maintained the models and the database. Even in the area of health care decision-support is provided. Remko et al. [OvVSH16] designed a Web-based DSS that guides patients to make suitable decisions in case of low back pain. The system was validated by conducting online surveys.

To the extend of our knowledge this work implements the first decision support system in the area of middleware benchmarking. Our work follows guidelines and recommendations in regards to building a DSS pointed out by Bhargava et al. [BPS07]. The proposed DSS can be classified as a knowledge-driven approach, where the decision-making is based on the defined information stored in the database. We have conducted an extensive search on the available State of the Art, however, we did not come across other decision support systems for middleware benchmarking. The search included contents of the WWW as well as books provided in the academic library of University of Stuttgart. The keywords and phrases used in this extensive search are listed below:

- Decision Support for Benchmarking
- Decision Solving for Benchmarks
- Decision Support for Middleware Benchmarking
- Decision-Making in the field of Benchmarking
- DSS and benchmarks
- How to choose a Benchmark
- What is the Right Benchmark
- Selecting the Right Benchmark

4 Concept and Specification

This chapter provides a compressed list of requirements for the *DSS4MiddlewarePBenchmarking*, classified as functional and non-functional requirements. It also details the use cases that the system must cover. Moreover, we provide a brief description of the involved system components and the overall system architecture. It also covers the taxonomy that our system is built on and the methodology on how we conducted the information inside the taxonomy.

4.1 Requirement Analysis

This section deals with the functional and non-functional requirements that were considered to be relevant for our Decision Support Framework. The following table (see Table 4.1) describes the defined Functional Requirements (FR) and the second table (see Table 4.2) lists the identified Non-Functional Requirements (NFR).

No.	*Requirement*	*Description*
FR-1	Visualisation of the Decision Support System	Extract data from the *DSS4MiddlewarePBenchmarking* in a human recognisable manner in order to provide solutions for decision-making in the matter of Middleware Performance Benchmarking.
FR-2	Easy expansion of the decision tree through the UI	CRUD operations are provided for each table inside the database and can be accessed in the user interface level.
FR-4	Highlight relationship of elements	Upon selection of a specific criteria the system should dynamically display additional options if there is any nested hierarchy.
FR-5	Clear indication of selected field	Upon selection of different check boxes, the output must display a text that mentions the criteria that were selected. This enables a good understandability.

Table 4.1: Functional Requirements for the implementation

No.	*Requirement*	*Description*
NFR-1	Usability	The developer should specify all main and relevant benchmark characteristics according to the decision tree with an easy interactive interface. The interface should be graphical, web-based and user-friendly that allows querying the knowledgebase. The software platform should be self-explained to the user, where the user knows exactly what the required steps are.
NFR-2	Consistency	The decision support system should allow the user to view the results of the selected criteria in a consistent manner. The user interface and all included operations should always behave the same way.
NFR-3	Performance	The knowledge base conducted from the decision tree should be exposed as a RESTful application.
NFR-4	Web-based development	Portability, cross-platform support and a user-friendly interface have to be provided using web programming language and related state-of-the-art technologies
NFR-4	Easy installation	The configuration and installation of the software must be rapid and easy.
NFR-5	Well supported technologies	For future extension and modification on the source code latest technologies, clean coding and detailed comments should be provided in the implementation.

Table 4.2: Non-Functional Requirements for the implementation

4.2 Use Cases

After defining the functional and non-functional requirements in Section 4.1 the use cases that the *DSS4MiddlewarePBenchmarking* must support are extracted. The system supports two different types of users, system administrators and system users. The system administrator is considered to be the more privileged one as it can perform all actions that users can perform and additional actions exclusively for its role. Figure 4.2 illustrates a use case diagram followed by a detailed definition of each use case.

NOTE: The term 'resource' used in the use case diagram refers to the entities mentioned in the resource model in 5.2.

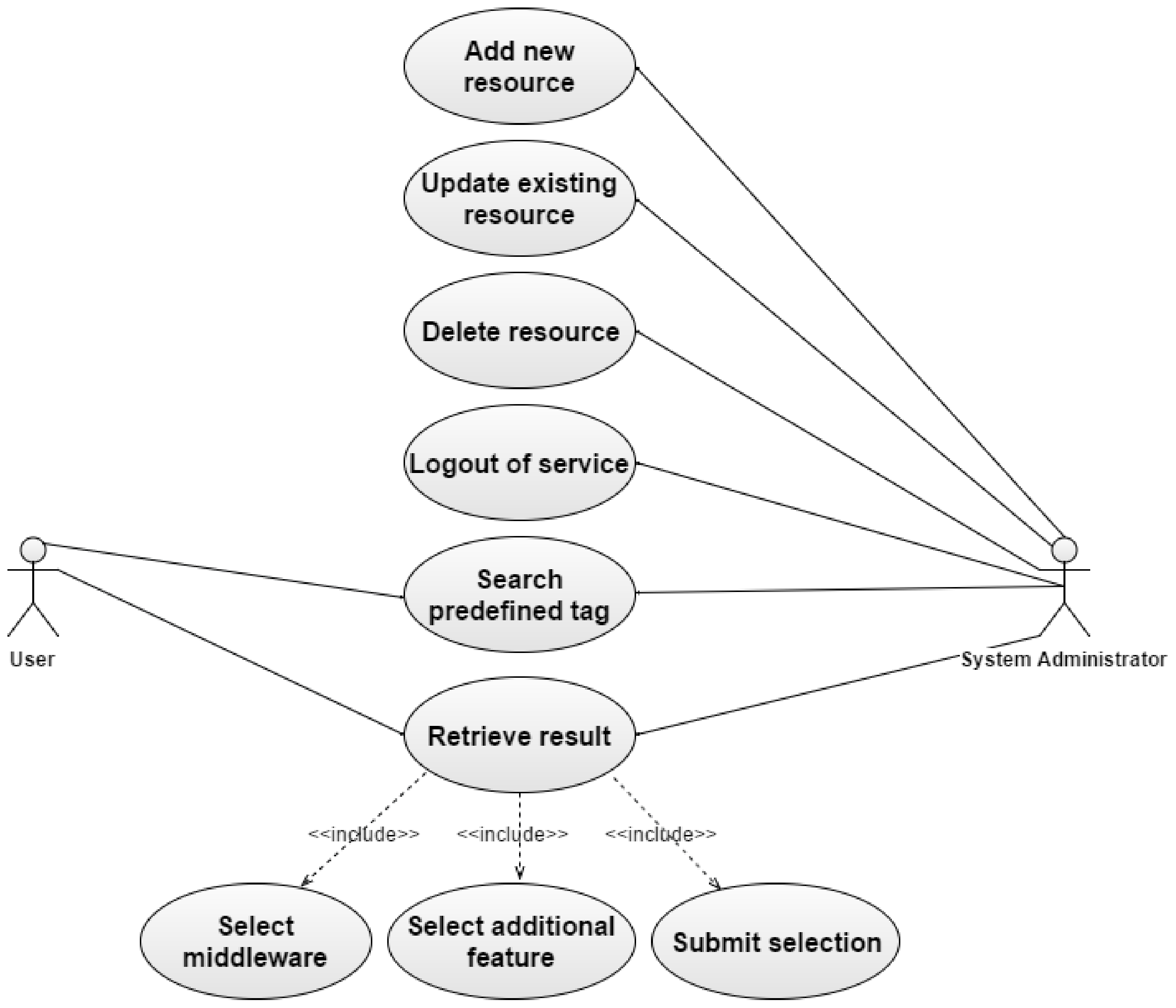


Figure 4.1: Use Case diagram for DSS4MiddlewarePBenchmarking

4.3 Taxonomy

When comparing the different middleware benchmarks it was important to only include relevant features into the taxonomy. The taxonomy includes features related to the workload and other important features of each benchmark. Information that does not remain the same was excluded from the taxonomy (e.g. specific software that the benchmark environment should run on).

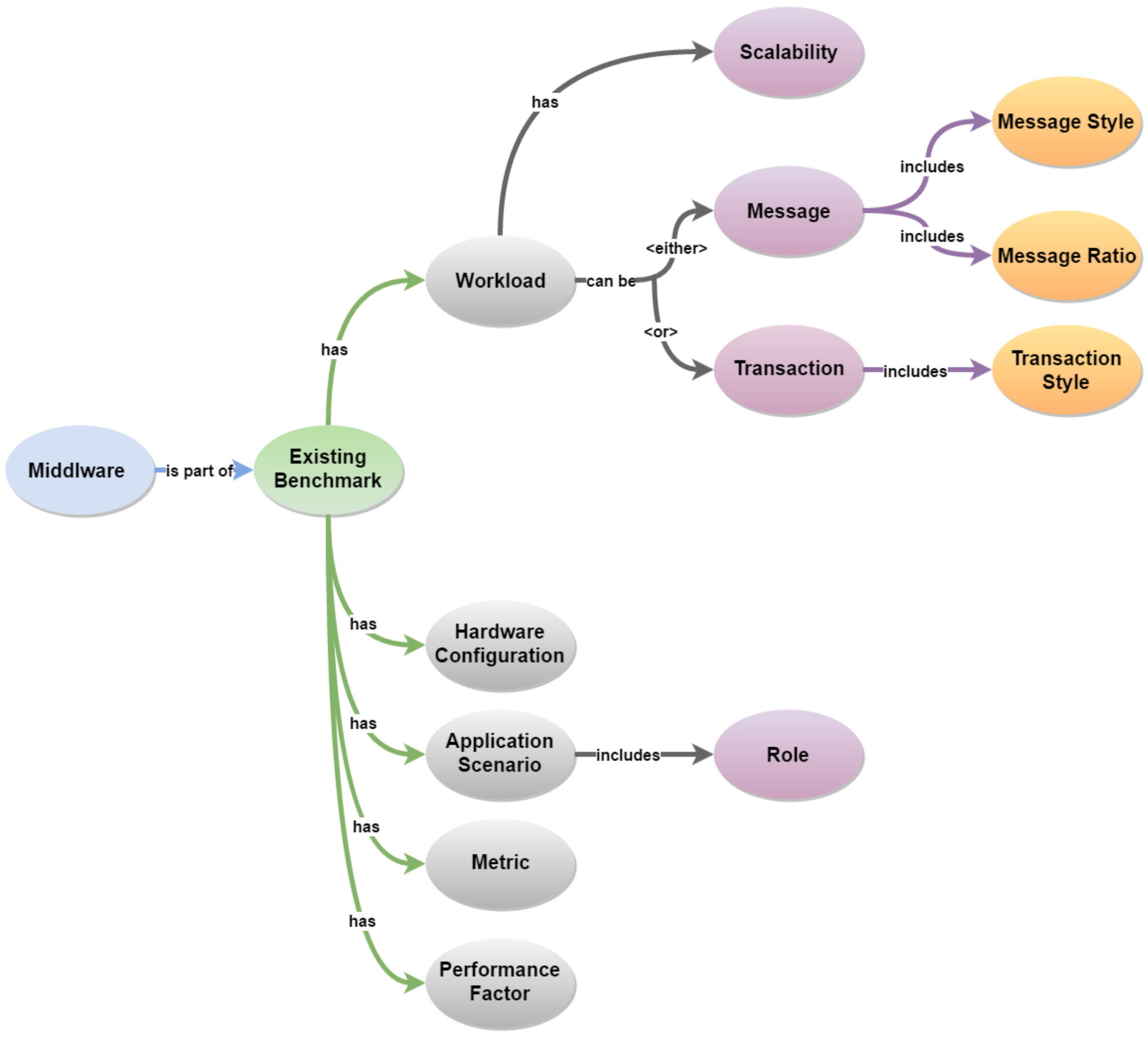


Figure 4.2: Taxonomy on which the DSS4MiddlewarePBenchmarking is based on

The taxonomy includes relevant information that is required for the creation of new middleware benchmarks. Moreover, the information provided can also be used as an informative source for middleware benchmarks. All information inside the taxonomy was taken from four different standardised middleware benchmarks that are currently active (see Table 4.3). Two of them were published by *Standard Performance Evaluation Corporation (SPEC)* while the other two are part of the *Transaction Processing Council (TPC)*.

Benchmark name	Consortium	Middleware
SPEC JMS® 2007	SPEC	Message-Oriented Middleware (MOM)
SPEC-JBB	SPEC	Java Server
TPC-C	TPC	Database Management System (DBMS)
TPC-E	TPC	Database Management System (DBMS)

Table 4.3: Overview of standardised benchmarks used in the taxonomy

Detailed contents of the main entities inside the taxonomy are explained in the following. At first, most relevant decisions and their possible outcomes are introduced. Due to the fact that all benchmarks have the same purpose of measuring performance characteristics for middleware technologies, the structure of these benchmarks is overlapping in many aspects. When comparing the different benchmarks it can be noticed that all of them contain overlapping or similar information. Therefore, details that are relevant and identical within the different benchmarks are combined into one entity. The main aspect that leads to combining the information into entities is the amount of information that can be generalised. The description also includes the procedure on how the information conducted is grouped into these entities.

4.4 Decision Point 1 - Middleware

The following chapter describes the elaborated *Decision Point 1* regarding *Middleware* as depicted in Figure 4.3.

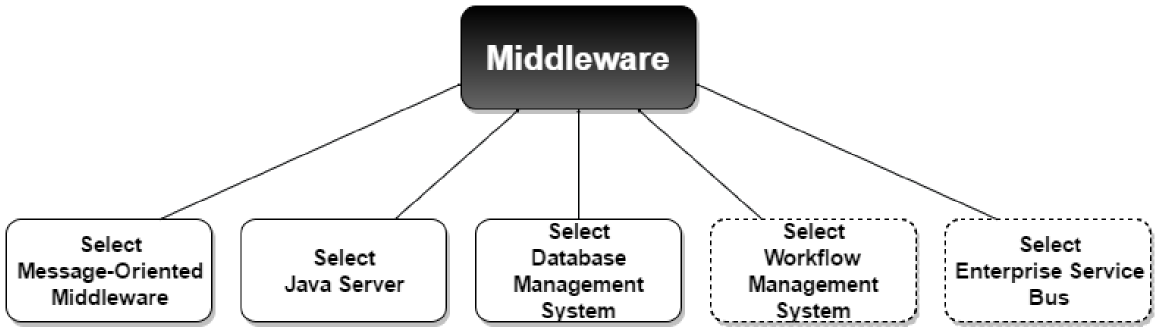

Middleware
Select Message-Oriented Middleware
Select Java Server
Select Database Management System
Select Workflow Management System
Select Enterprise Service Bus

Figure 4.3: Elaboration of Decision Point 1 - Middlware

Description

Currently, there are numerous middleware systems available. A few of them are covered by standardised benchmarks and for some only custom benchmarks are available. When comparing these middleware technologies, the different architectural features have to be considered, i.e. Workflow Management Systems deal with work flows whereas Message-Oriented Middleware (MOM) uses messages. The five different middleware technologies mentioned in Figure 4.3 were considered to be the most relevant and demanded in terms of benchmarking, as Workflow Management Systems [RHW14] and Enterprise Service Bus [KS09] are involved in current research topics. Furthermore, there are standardised benchmarks available for the performance of Java Server [SPE15], Message-Oriented Middleware [SPE07] and Database Management Systems [TPC92a] [TPC92b].

The accessing point of the decision support framework is selecting the required middleware in order to retrieve additional features according to the selected middleware. There exists a possibility to either select a single middleware and further render the additional features or search the whole knowledgebase that includes information about all elaborated middleware technologies.

4.5 Decision Point 2 - Workload

The following describes the elaborated *Decision Point 2* regarding the different *Workloads* of the conducted *Benchmarks* as depicted in Figure 4.4.

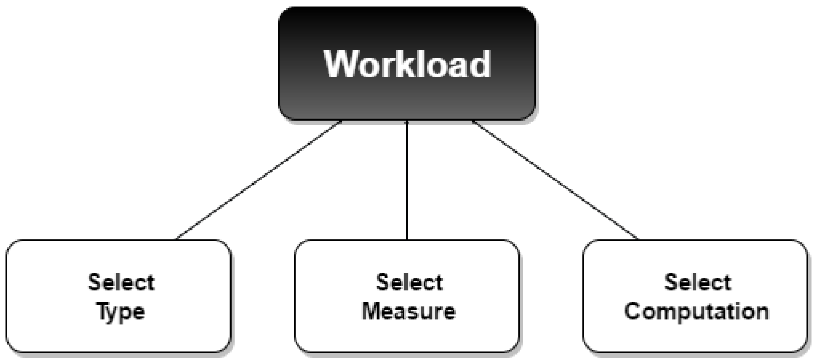


Figure 4.4: Elaboration of Decision Point 2 - Workload

Description

The main questions that arise in regard to the usage of benchmarks are [DAM01]: 1. How many transactions can our system process per minute?; or 2. How many requests can the system serve per second? In order to understand the benchmark results correctly, it is important to understand the workload in a correct manner. Once the benchmark results are interpreted as expected, the steps towards improving the system can be made. Therefore, the inclusion of the *workload* as a selection criteria was necessary. Every benchmark consists of a workload, where workload is referred to the average amount of work handled by an entity during a specified time. The best way to evaluate the performance for a specific system is by applying actual workload on the platform and get a concrete measurement of the results [DAM01]. The need of a correct workload is very important as pointed out by Avritzer et al. [AW96]. As it is a crucial characteristic of a benchmark and it is covered by all conducted benchmarks it was straightforward to aggregate it into one decision point. For the conducted benchmarks different workload types could be identified. The information regarding the different workloads was taken from the corresponding design documents provided by the benchmark provider.

When *selecting* the *type* of the workload, the possible outcome of the decision can be the different workload types for each benchmark. For instance, the workload type for the SPECjms2007 benchmark is *SPECjms2007@horizontal* and *SPECjms2007@vertical*. The two different types allow the users to configure and customise the workload based on their own preferences [SKBB07]. In the case of the horizontal topology, the system is able to manage an increasing amount of destinations (i.e. increased number of physical locations) while maintaining the message traffic at each location. In terms of the vertical topology, the system is able to handle an increasing amount of message traffic while maintaining the number of physical locations.
The *workload type* in case of the TPC-C benchmark is *transactions*. The five transactions that make up the workload are: *New-order, Payment, Delivery, Order-Status* and *Stock-level* [TPC92a]. The distribution of the execution time for each transaction varies. When selecting the *measure* for the TPC-C benchmark the option *transactions per minute* is displayed.

4.6 Decision Point 3 - Metric

The following describes the elaborated *Decision Point 3* regarding the different *Metrics* that are part of the conducted *Benchmarks* as depicted in Figure 4.5.

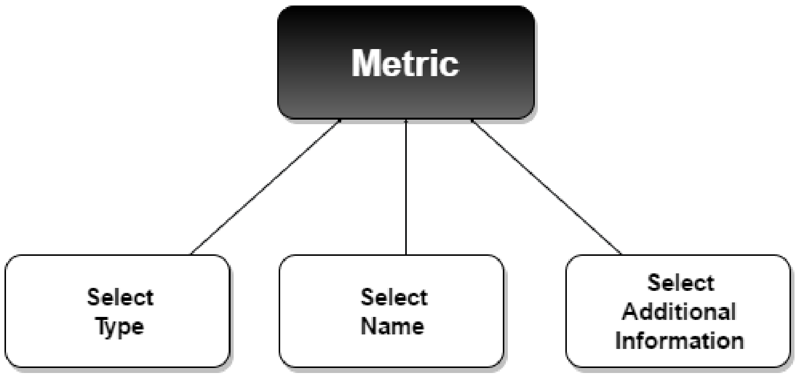


Figure 4.5: Elaboration of Decision Point 3 - Metric

Description

One of the characteristics of a good benchmark is the usage of meaningful and understandable metric [Hup09]. According to Alves et al. [AYV10], the usage of metrics provide a control instrument in the field of software development and the maintenance progress. It can also be intended to have a clear comparison and rating of software products. For instance, the metric used for the SPECjbb2015 benchmark is referred to *SPECjbb bops*. This term can be easily interpreted by the benchmark user that it represents *Business Operations per Second* [Hup09]. Another example is the metric of the TPC-C benchmark, namely *tpmC*. As it is a transactional benchmark that is measuring throughput, it is measuring the *Transactions per Minute*.

4.7 Decision Point 4 - Application Scenario

The following chapter describes the elaborated *Decision Point 4* regarding the different *Application Scenarios* that are included inside each *Benchmark* as depicted in Figure 4.6.

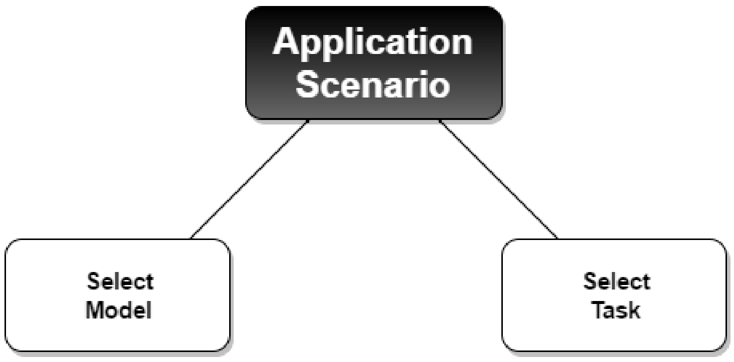


Figure 4.6: Elaboration of Decision Point 4 - Application Scenario

Description

The application scenario in a benchmark should cover similar target audience. It is important to assure that the benchmark can be suitable for a large number of users [Hup09]. Every benchmark covers an application scenario that provides conceptual frameworks for a specific area containing underlying components that are well-known for applications of this kind [DAM01]. Furthermore, the application scenario should be chosen in a way that different subsets of the functionality offered by the underlying technology are stressed [SKCB07]. All conducted benchmarks cover a specific business model along with defined tasks that are being fulfilled.

When selecting the *model* and *task* the available output is:

1. Wholesale supplier - TPC-C models an application that takes care of the orders involved in a wholesale supplier [TPC92a].

2. Supply chain - SPECjms2007 models an application that involves the supply chain of a supermarket company. It stresses different subsets of the capabilities provided by JMS servers [SPE07].

3. Supermarket company IT infrastructure - SPECjbb2015 models different aspects of a world-wide supermarket company [SPE15].

4. Brokerage firm - TPC-E models a brokerage firm that involves different customers that work with different types of transactions. These transactions are related to functionalities that are covered by a brokerage firm, such as, trades, market research, and account inquiries [TPC92b].

4.8 Decision Point 4.1 - Role

The following chapter describes the elaborated *Decision Point 4.1* regarding the different *Roles* that are defined in the previously mentioned *Application Scenarios* as depicted in Figure 4.7.

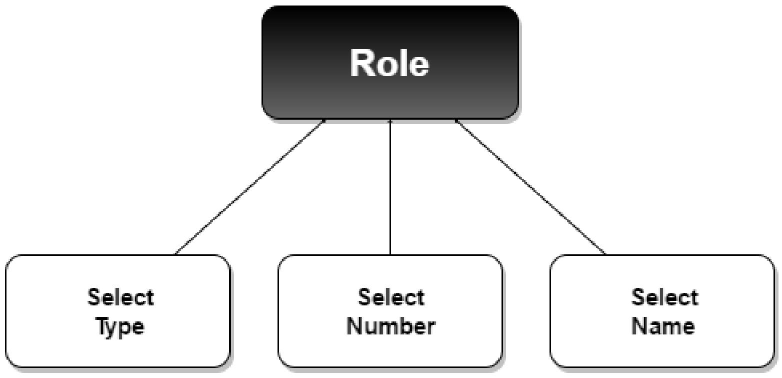


Figure 4.7: Elaboration of Decision Point 4.1 - Role

Description

Each application scenario involves different roles that have a focus on different tasks. The roles defined in the application scenario could be grouped according to the nature of their tasks. We came up with the conclusion to group these tasks in two groups, namely admin and (regular) user. Most of the conducted application scenarios had a clear distinguishing between administrative tasks and normal operations, therefore, this solution was most suitable. For instance, in the SPECjms2007 benchmark four different participants were involved in the scenario [SPE07]. Firstly, the *Company Headquaters*, responsible for the accounting of the company, can be classified as the *admin* of the scenario. The *Distribution Centers, Supermarkets* and *Suppliers* are involved in tasks related to their capable functionalities, therefore these were grouped as (regular) *users*.

4.9 Decision Point 5 - Hardware

The following chapter describes the elaborated *Decision Point 5* regarding the *Hardware configuration* for setting up the requirement of the System Under Test (SUT) in order to run a *Benchmark* as depicted in Figure 4.8.

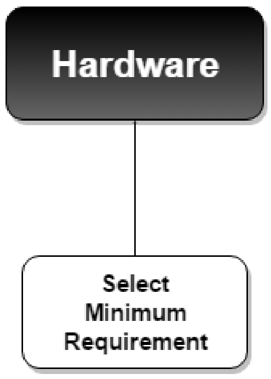


Figure 4.8: Elaboration of Decision Point 5 - Hardware

Description

Mostly benchmark users are clueless about the technical configurations of a benchmark. Therefore, it was necessary to include the Hardware requirements needed to assure a smooth benchmark experience. Furthermore, the use of hardware in a benchmark should be in a way that can be closely related to the consumer environment [Hup09].

4.10 Decision Point 6 - Scalability

The following chapter describes the elaborated *Decision Point 6* regarding *Scalability* of the previously mentioned *Workload* of a benchmark as depicted in Figure 4.9.

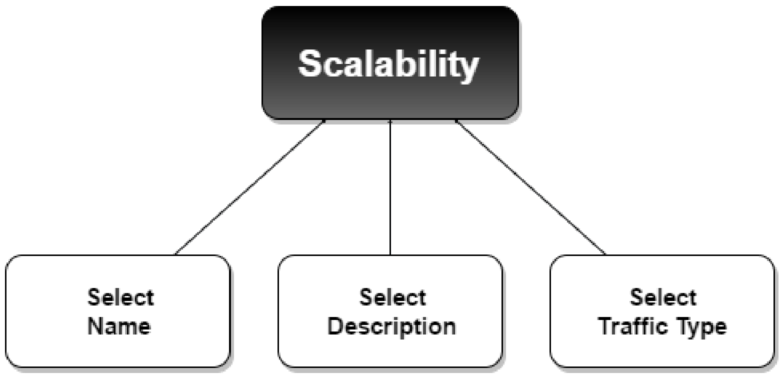


Figure 4.9: Elaboration of Decision Point 6 - Scalability

Description

It is important the the benchmark tests can be applied to a broad spectrum of systems, in regards to cost, performance, and configuration [DAM01].
For some of the benchmarks, the ability to scale the workload was provided. For instance in the SPECjms2007 [SPE07] benchmark a natural way to scale the workload was provided. To be precise, two different types of scaling was provided, the *horizontal* approach supported scaling in terms of increasing the number of *supermarkets*. The other way to scale the workload was the *vertical* approach, where the number of products sold in each supermarket could be scaled.

5 Design

5.1 System Architecture

The prototypical implementation is developed as a single-page web application. In Figure 5.1 the system architecture is depicted showing the logical architecture in regards to the different application layers with technologies used for each layer. The main part inside

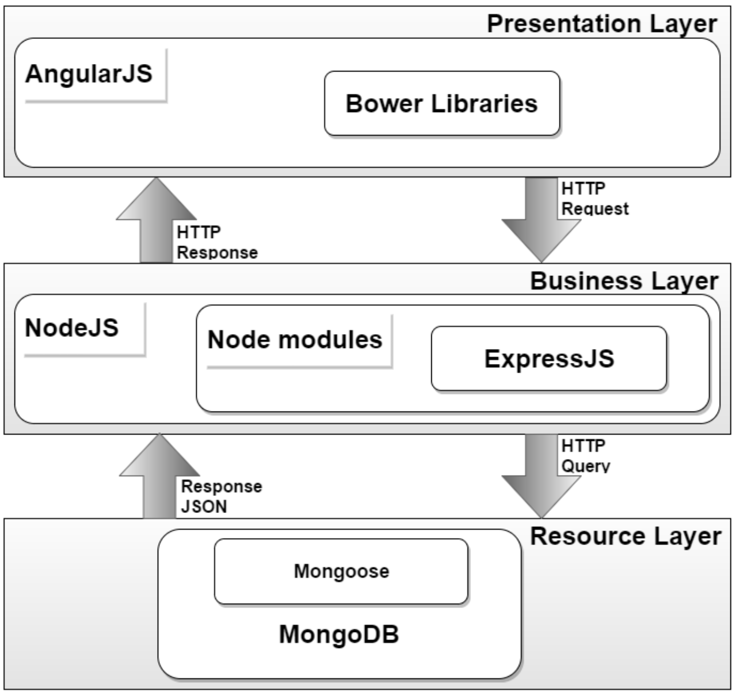


Figure 5.1: System Architecture of the Prototypical Implementation

the prototype's Presentation Layer are the bower libraries used. Bower is responsible for the whole web platform and offers packages that contain scripts, CSS stylesheets, HTML templates, images and fonts. Therefore, bower is mostly used for client-side work. Another important component inside the business layer is called the Node Package Manager (NPM). NPM is used in the context of JavaScript modules that are compatible for node. Furthermore, the object modeling tool *Mongoose* that resides in the resource layer is also a module provided by node.
The interaction between the business layer and the resource layer is done by sending HTTP queries to the database and receiving responses in JSON-format. Inside the business layer the interaction takes place with the presentation layer where the interaction takes place in form of outgoing HTTP responses and incoming HTTP requests.

5.2 Class Diagram

This section models the resources used in our system. A REST API involves the attachment of linked resources. According to Fielding, "REST uses a resource identifier to identify the particular resource involved in an interaction between components" [Fie00], therefore it is necessary to define resources for a REST-based service.

The resource model specifies resources and the relation between them. The use of an ER diagram displays the relation between them. Each entity also includes an object ID which is automatically generated by MongoDB. The data type of IDs is provided by mongoose: *mongoose.Schema.ObjectId*.

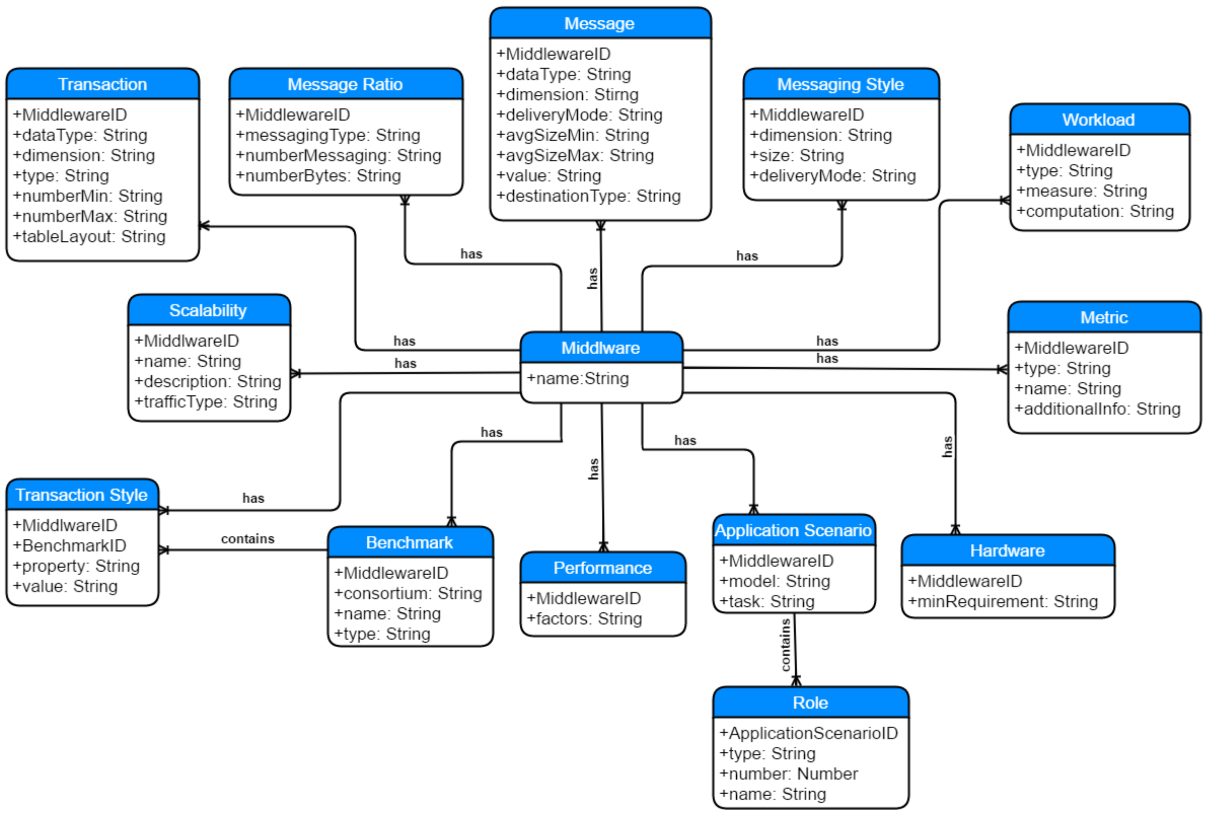


Figure 5.2: Resource Model Representation in the Form of ER Diagram

5.3 RESTful API

Previously, all design rules related to REST APIs are discussed intensively in section 2.6.4. According to the resource model in Figure 5.2, the design of this API is constructed in terms of identifying all resources available to the user. The communication between a client and as server in a RESTful web service is based on an interaction of exposed resources on the server side.

5.3.1 API Design

All resources covered by the implementation can be divided into two groups, one of them is responsible for computational operations. The other type supports querying and viewing of results. In order to uniquely identify each resource, an easy understandable URI is referred to each resource. The structure of the URI is described as follows:

- The first part of the URI describes the resource that is addressed by the client. Therefore, it can be */middlewares*, */benchmarks* or */workloads* etc. These URIs illustrate set of middlewares, benchmarks or workloads.
- The second part of the URI describes the ID for a specific resource where actions need to be performed. In our example: */middlewares/1* would address the middleware where ID equals to 1.
- In order to query the system in a deeper level using the 'filter down' selection criteria in the front-end, it is necessary to specify the function name and the input parameters. The URI would be presented as: */api/benchmarks?populate=1&fields[]=consortium*. This is an example that displays the different consortia inside *benchmarks*.

Each resource implements the HTTP methods GET, PUT, POST and DELETE. The impacts of a HTTP request on a resource is expressed below:

Note: For understandability the APIs related to the back-end are differntiated from the ones used in the front-end with '/api/' prefix before mentioning the first part of the URI that is described above.

/api/resource *(e.g. /api/middlewares/)*

GET: displays the list of specified resources in the system. If a query is included in the URI, the output will match the selection specified in the URI. All users can access this method in our prototypical implementation.

PUT: updates already existing resources in the system. This is accessible to system administrators only.

POST: inserts a new specified resource to the system and the information is provided in the body of the request. Only administrators can add a new specified resource in our system.

DELETE: deletes the specified resource. Only administrators are authorised to delete a resource.

/api/resource/x *(e.g. /api/middlewares/56eea9749c8471ce22e2c100)*

Note: 'x' is referred to a resource identifier

GET: lists the data of the resource that has identity X according to the scope of the request placed by the user.

PUT: updates the information of the resource that is identified by X. The planned insertion is taken from the body of the request. Only administrators are authorised to update a resource.

POST: a new resource with identity X is added by an authorised system administrator.

DELETE: authorised system administrator deletes the resource with identity X

/api/resource/x/resource *(e.g. /api/middlewares/56eea9749c8471ce22e2c100/workload*

GET: lists the information of a resource with identity x associated to another resource. In our case, when the user selects a specific middleware and also queries one hierarchy deeper by selecting an additional feature.

PUT: not applicable.

POST: not applicable.

DELETE: not applicable.

/api/resource/x/resource?key=value *(e.g. /api/benchmarks?populate=1&fields[]=consortium)*

GET: displays the information of the resource with identity x and fulfilling the query string. For our system, this is a URI designed for further in-depth querying of the system.

PUT: not applicable.

POST: not applicable.

DELETE: not applicable.

The rules concerning API design were discussed in detail in section 2.6.4. We considered these rules when creating the APIs and used only nouns for the expressions. Furthermore, we followed the design rules and provided a customised output for the user to ensure better readability and usability.

6 Implementation

This section involves the realisation view of this Master's thesis with the focus on the implementation of the specified Decision Support System. In order to produce a successful Decision Support Framework it is important to provide data to stakeholders that is available and accessible for decision-making purposes. First of all, Next, technologies, frameworks and illustrated system architecture are described. Later, the implementation is demonstrated using screenshots showing the used data visualisation approach.

6.1 Technologies and Frameworks

For many years developers focused on the LAMP tool stack [Law05] (named after its constituent parts: Linux Operating System, the Apache Web Server, relational MySQL database and the scripting language PHP). In the past few years the new MEAN technology stack has gained popularity in the development of web-applications. MongoDB, Express.js, Angular.js and Node.js are based around the programming language JavaScript, therefore applications based on MEAN do not need separate languages for front-end and back-end execution environments [Mea14]. Despite the fact that originally JavaScript was mainly developed for client side web programming, it has invaded the server-side programming by virtue of environments like Node.js. In the following we examine the components of the MEAN web development tool stack and show its appropriateness and advantages in the contents of implementing RESTful web services.

6.1.1 The MongoDB, Expressjs, Angularjs and Nodejs Stack (MEAN Technology)

The implementation of our *DSS4MiddlewarePBenchamrking* was carried out using the MEAN development. The MEAN technology stack is an open-source JavaScript Software stack that allows the development of dynamic web applications. This acronym stands for the four components it includes in the stack, namely: MongoDB, Express.js, Angular.js and Node.js. With the help of this technology stack we were able to develop back-end services along with a web-based user interface in the front-end. In the following, its constituent components are described briefly [PJC15]:

1) **MongoDB**

Most web-services are based on a data storage which usually takes the form of a database management system. Although relational database management solutions were commonly provided, the tendency to use NoSQL type database has increased. The NoSQL database MongoDB enables the storage of data that vary in structure in JSON-like documents. Similar information is stored together to allow fast access through the MongoDB query language.

MongoDB has a different approach to manage the documents and does not have a specific language. The manipulation and querying of documents is accomplished using a very rich set of operators that are composed with each other applying JSON structure. The comparison of traditional relational database and NoSQL database is examined in table 6.1

2) **Expressjs**

Express.js provides a server framework for web applications, while it is built on the elemental functionalities of Node. It provides the developer a mechanism to deal with web routing and HTTP methods (discussed in section 2.6.5) with the use of a wrapper around an underlying Node environment. With the usage of Express.js an easier and more refined solution is granted than resolving these capabilities directly applying Node.

3) **Angularjs**

Angular.js (or just *Angular*) is a web application framework which offers a client-side framework for MVC (Model-View-Controller)[LR01] single page web applications. Another package that we included in our system is the *bootstrap*. With the aid of this package a maximum benefit from Angular can be achieved as it well-designed CSS elements that makes the design of modern web content easy and smooth. The combination of this tool and the low-level interface of Angular provides opportunities to create elegantly designed and powerful web applications which are able to utilise the web-services made available by the other components in the technology stack.

4) **Nodejs**

The most important tool of the technology stack is called Node.js (mostly referred to just Node). Node is a JavaScript execution environment on the server-side that is built on Google Chrome's V8 JavaScript runtime. It aids the development of concurrent and highly scalable applications in a rapid manner. Web server environments that are either lightweight or even high performance can be built using Node [C+15]. In the creation of web-service APIs it is an optimal candidate. A number of leading companies in the market are using Node, PayPal [Dic15] being one of them.

6.1.2 REST (Representational State Transfer) API

The Representational State Transfer (REST) is an architectural style that is applied on a distributed hypermedia system. The adoption of this paradigm is increasing as it is considered to be a clearer substitute to web services that are SOAP- and Web Services Description Language (WSDL)-based [Rod]. The RESTful paradigm for web-services makes use of the HTTP methods: *POST, GET, PUT and DELETE*[F+99b]. These HTTP operations allow the mapping on to the fundamental CRUD database operations – Create, Read, Update and Delete. With the construction of a simple URI the four HTTP methods can be linked to functions that allow to Create, Read, Delete or Update entries inside a web-service. Any type of authenticated user is able to consume this service afterwards. There are different types of clients available, in our case the client is a web application.

HTTP Status codes

For the implementation of the framework we also used HTTP status codes (see 2.6.5). These status codes are standardised response codes that are outputted by the web server. With the use of these codes we can diagnose the source of the problem when a web page or an abject does not load as expected. There are two different types of groups to display errors; 4xx Client Error and 5xx Server Error. The additional groups are informational, confirming success or redirecting. We applied the following list of status codes on our system [resa]:

1) **2xx Success**

200 OK - This code means that the request has succeeded. The returned information depends on what type of method was used in the request.

2) **4xx Client Error**

400 Bad Request - When the server is not able to understand the request due to the syntax being malformed. *404 Not Found* - The request-URI cannot be matched to any resources by the server.

3) **5xx Server Error**

500 Internal Server Error - Generic error message when the server encounters an unpredictable situation which prevented the normal processing of the request.

6.2 Prototypical Implementaion

The prototypical implementation of the *DSS4MiddlewarePBenchamrking* guides the user to the conclusion of deciding which benchmark is more relevant.

6.2.1 Methodology Description

The following sequence diagram (see Figure 6.1) portrays the steps that lead to the rendering of results on the user interface. First of all, the user selects different options and submits the request on the user interface. The call is sent to a function of the *Front-End Controller* which calls the method of the *Service (Factory)* that is responsible to send the request in form of a HTTP request to the server. Once on the server, the call is matched against the existing APIs defined as routes. If a matching URL is found, the call gets redirected to one of the methods in the relevant *Back-End Controller*. In case of an error, the status code 500 for INTERNAL SERVER ERROR is sent back to the *Service (Factory)*. Upon success either the status code 200 for OK is sent or 404 for NOT FOUND. At the end, the response is forwarded to the *Front-End Controller* which finally displays the response to the user.

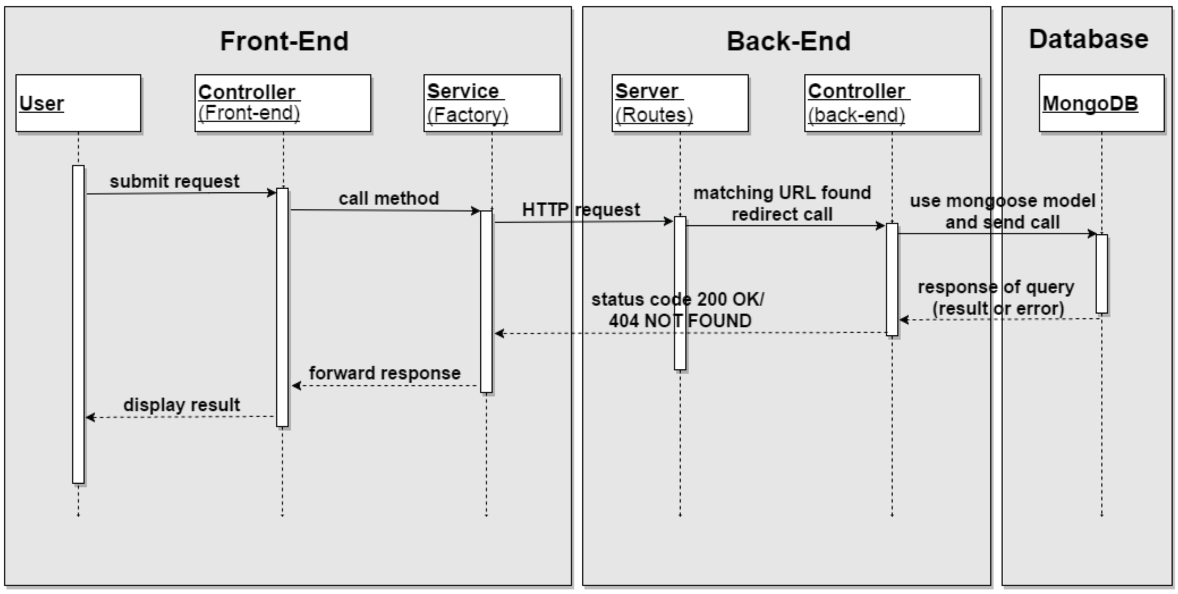


Figure 6.1: Sequence Diagram for Use Case *Retrieve Results*

6.2.2 Knowledgebase

The knowledgebase is made up of the data that is included inside the taxonomy mentioned in section 4.3. For the implementation of our knowledge base the booming non-relational database technology MongoDB is used. In the following table (see Table 6.1) a comparison with the popular relational database management system (RDBMS) is made. The table concludes on to why a non-relational solution is best suitable for our *DSS4MiddlewarePBenchmarking* [Mon16].

All points mentioned above have led to the conclusion that MongoDB is in favour of the *DSS4MiddlewarePBenchmarking*. Our system involves tables which have an irregular data model, therefore a schemaless development was needed. With this approach new properties could be introduced on the go without the need of performing schema evolutions and data migration. Furthermore, as our system portrays a decision support system where it is predictable that a great amount of querying is required in order to support the decision-making, MongoDB is best suited for us as it is developer-friendly in regards to querying and manipulating documents [Mon15].

Another reason why we chose MongoDB is that the adoption of MongoDB in the industry is growing everyday and more and more companies are open to this fairly new technology. In 2013, eBay deployed their first Node.js web application based on MongoDB [Dic15].

Feature	*MongoDB*	*MySQL*
Dynamic Schema	Enables a dynamic schema that allows adding tables on demand during implementation.	Requires a strict schema for its data model which means that all tables should be created with columns defined.
Data locality	Related data is stored together to provide fast query access which omits the need of joining tables.	Multiple number of tables where some queries require joining tables together.
Complexity	Simplified development due to automatic mapping of MongoDB documents to object-oriented programming languages.	Complex object-relational mapping (ORM) layer translating objects in code to relational tables.
Changeability	Database schema is easily expandable with new requirements.	Small changes require more time and effort.
Availability and Scalability	Without any changes MongoDB scales easily with no downtime.	Requires critical and custom engineering effort.
RESTful paradigm	Easily accessible via a RESTful JSON API.	No simple access via REST services, a much more complex SQL API is required.
Data Management	Querying and manipulation of data requires a set of operators. The unified representation is expressive and easy to understand. Furthermore, no additional programming language knowledge is required, due to its self-explanatory nature.	The special purpose SQL programming language is used to manage data inside relational database management systems.

Table 6.1: Feature comparison of MySQL and MongoDB

6.2.3 Resource Model

Our resource model described in section 5.2, highlights the resources and their interaction. In regards to our implementation, the resources have been defined as single resources. The interaction amongst them is highlighted by using IDs in order to relate resources to each other. The listing 6.1 is shown to be consisting of the model definition for *role* resource with the reference to the application scenario resource. They are mapped to our NoSQL database in order to store relevant data like creating a role and then assigning the application scenario to it. Despite the fact that we are using a schemaless database, these model definitions are provided by mongoose for validation reasons, meaning that it prevents the storage of undefined data type inside resources. Furthermore, it is possible to define which fields are considered to be required.

```
var mongoose = require('mongoose');

module.exports = mongoose.model('Role', {
type : {type : String, required: true},
number:  {type : Number, required: true},
name : [{type : String}],
// reference of application scenario document
applicationScenario: {type: mongoose.Schema.ObjectId, required:
   true, ref: 'ApplicationScenario'}
});
```

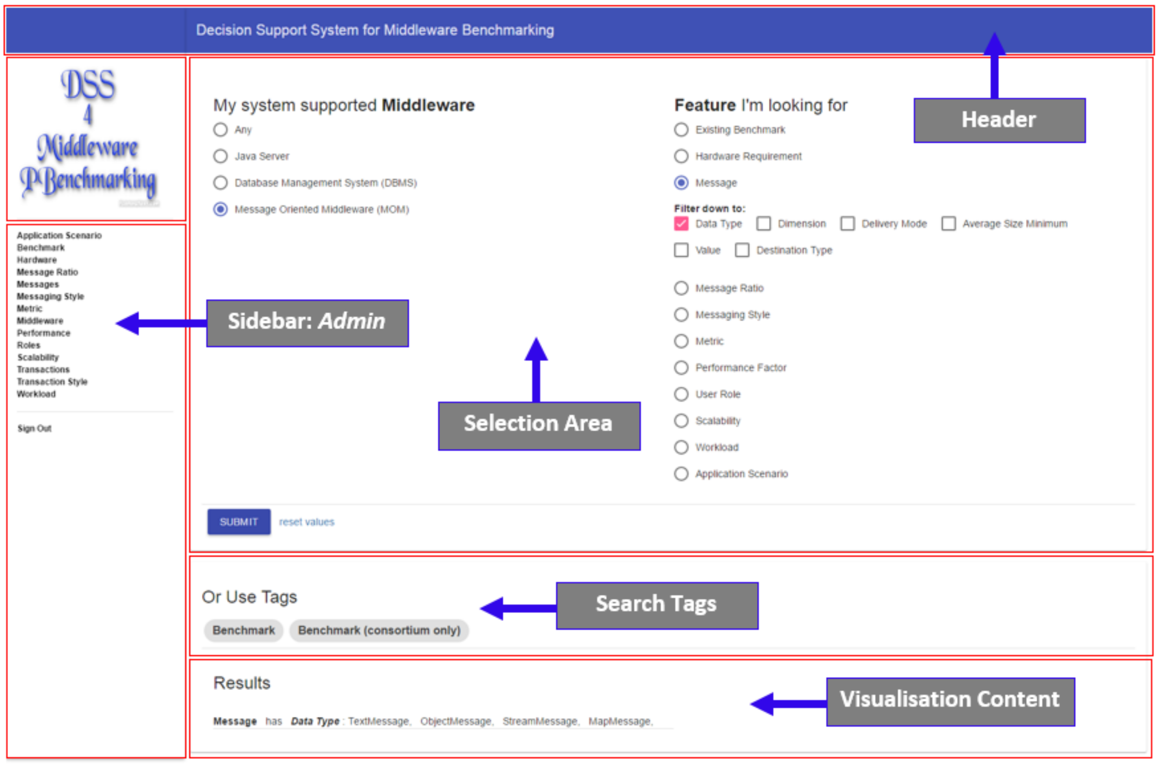
Listing 6.1: Model Definition of a single Resource

6.2.4 User Interface

The prototype is implemented as a RESTful web service fulfilling all non-functional requirements mentioned in section 4.1. According to the NFR-1 in table 4.2, the user interface should have a qualitative usability with an easy interactive interface. The interface should be graphical, web-based and user-friendly that allows querying the knowledgebase. The software platform should be self-explained to the user, where the user knows exactly what the required steps are.

In figure 6.2 the user interface of our *DSS4MiddlewarePBenchamrking* prototype is illustrated were all main areas are highlighted accordingly. The Header area does not contain any information regarding the functionality offered by the *DSS4MiddlewarePBenchamrking* but rather contain general information such as heading. The sidebar provided at the left hand side (i.e. sidebar admin) includes log in and CRUD operations that are provided for system administrators. The majority of the screen is reserved for the selection area, where the user can choose a combination of different check boxes and get the corresponding results displayed in the divided result screen underneath.

The selection fields are divided into two different groups and an additional dynamic group that only appears when a specific selection is made (see figure 6.6). The first group is completely static, whereas the second group is partially static and some check boxes appear dynamically. In the first group, the user is asked to select a specific middleware that he is interested in. If there are not any preferences for a specific middleware the user can simply select *any* which will show results for all middleware systems that are stored inside the database 6.3). However, when selecting a specific middleware, the following groups dynamically adjust, based on what information is stored inside the database for the selected middleware.



Decision Support System for Middleware Benchmarking

Application Scenario
Benchmark
Hardware
Message Ratio
Messages
Messaging Style
Metric
Middleware
Performance
Roles
Scalability
Transactions
Transaction Style
Workload

Sign Out

My system supported **Middleware**

- Any
- Java Server
- Database Management System (DBMS)
- [✓ Message Oriented Middleware (MOM)]

Feature I'm looking for

- Existing Benchmark
- Hardware Requirement
- [✓ Message]

Filter down to:
[✓ Data Type] Dimension Delivery Mode Average Size Minimum Value Destination Type

- Message Ratio
- Messaging Style
- Metric
- Performance Factor
- User Role
- Scalability
- Workload
- Application Scenario

SUBMIT reset values

Or Use Tags

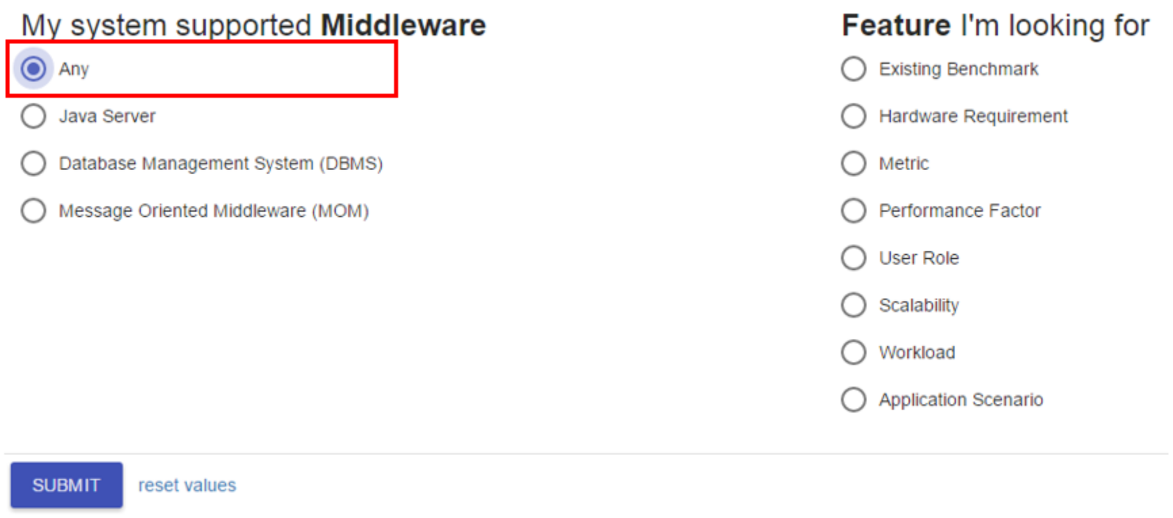
Benchmark Benchmark (consortium only)

Results

Message has ***Data Type*** : TextMessage, ObjectMessage, StreamMessage, MapMessage,

Figure 6.2: User Interface of the Prototypical Implementation

Upon selection of *any* in the first group, features that appear in the second group are the ones covered by all middleware systems in general (see figure 6.3).



My system supported **Middleware**

- [✓ Any]
- Java Server
- Database Management System (DBMS)
- Message Oriented Middleware (MOM)

Feature I'm looking for

- Existing Benchmark
- Hardware Requirement
- Metric
- Performance Factor
- User Role
- Scalability
- Workload
- Application Scenario

SUBMIT reset values

Figure 6.3: Body Area of the User Interface When no Specific Middleware is Selected

When *Message-Oriented Middleware (MOM)* is selected in the first group, the following group that covers the features automatically adjusts according to information related to *MOM* (see figure 6.4). Accordingly, when *Database Managements Systems (DBMS)* is selected, only features regarding to *DBMS* appear on the user interface for selection purposes (see figure 6.5). The dynamic group appears when the user is interested to know the *user role* mentioned

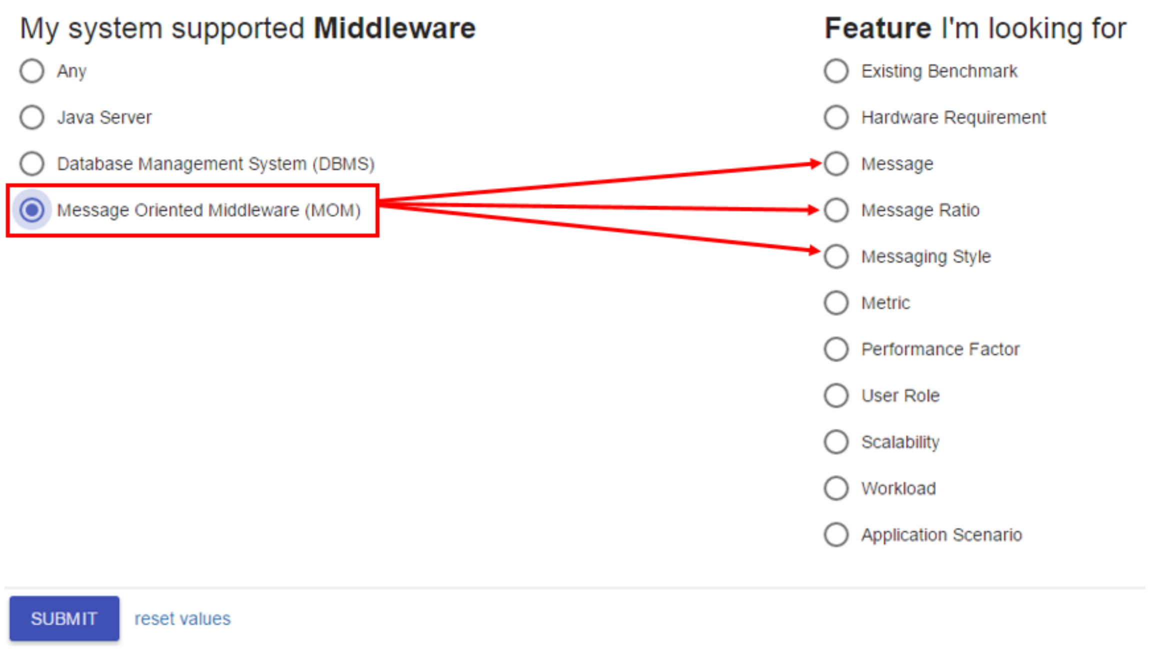
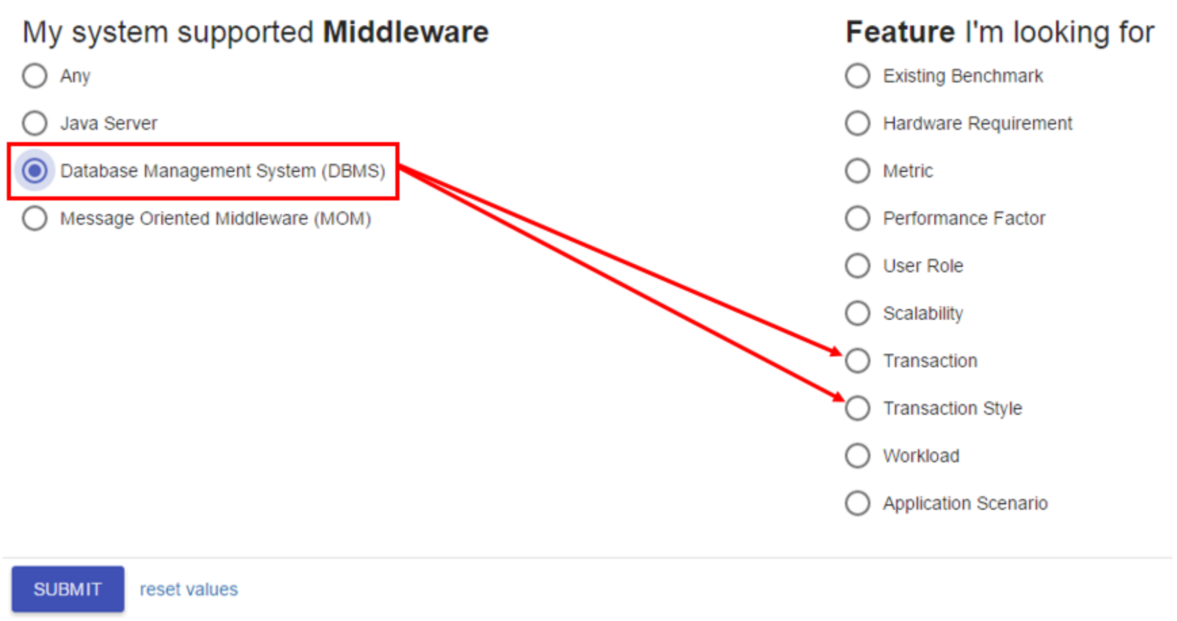


Figure 6.4: Body Area of the User Interface, MOM selected

inside the different application scenarios. Hereby, the selection is extended to an additional group that provides the option to choose the *application scenario* of interest. However, the expansion is not dependent on the selection of a specific middleware (see listing 6.2).



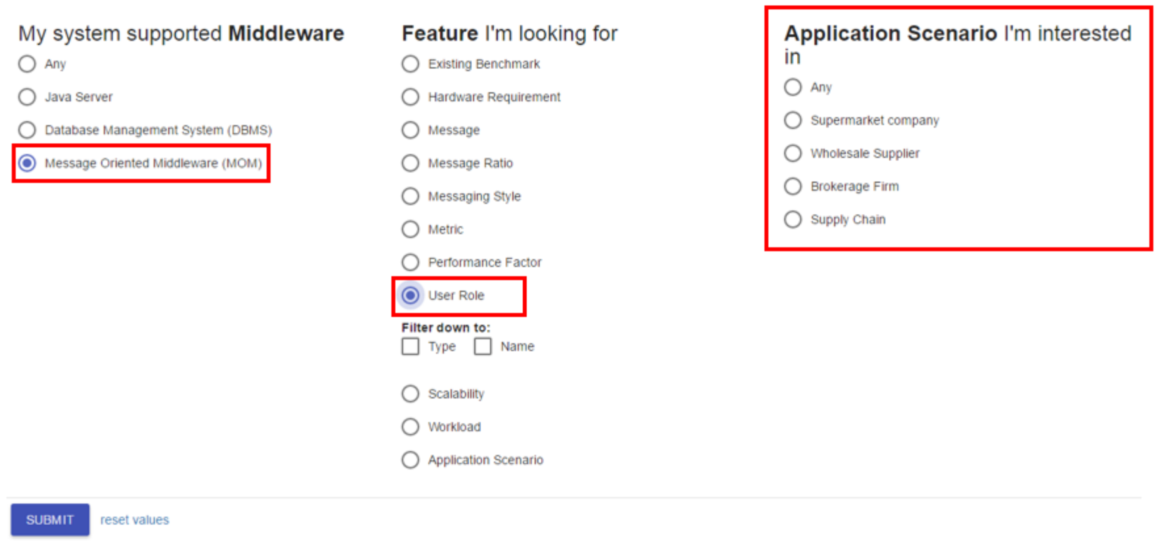
My system supported **Middleware**

- Any
- Java Server
- [✓ Database Management System (DBMS)]
- Message Oriented Middleware (MOM)

Feature I'm looking for

- Existing Benchmark
- Hardware Requirement
- Metric
- Performance Factor
- User Role
- Scalability
- Transaction
- Transaction Style
- Workload
- Application Scenario

Figure 6.5: Body Area of the User Interface, DBMS selected



My system supported **Middleware**

- Any
- Java Server
- Database Management System (DBMS)
- [✓ Message Oriented Middleware (MOM)]

Feature I'm looking for

- Existing Benchmark
- Hardware Requirement
- Message
- Message Ratio
- Messaging Style
- Metric
- Performance Factor
- [✓ User Role]

  Filter down to:
  - Type
  - Name
- Scalability
- Workload
- Application Scenario

Application Scenario I'm interested in

- Any
- Supermarket company
- Wholesale Supplier
- Brokerage Firm
- Supply Chain

Figure 6.6: Body Area of the User Interface, Additional Dynamic Group

```
// on submit button click
// start with url ='/api
var url = "/api";
//check if a specific middleware is selected and it's not "any".
   Append it to url.
if($scope.middlewareSelected && $scope.middlewareSelected !== '
   any' ){
url = url + "/middlewares/"+$scope.middlewareSelected
}
// the selection of the "feature group" selected is "user roles".
    Append to url accordingly.
if($scope.featureSelected == "roles"){
if($scope.appScenarioSelected && $scope.appScenarioSelected !== "
   any"){
url = "/api/application-scenarios/"+$scope.appScenarioSelected+
   "/roles"
}
else{
url = "/api/"+ "roles"
```

Listing 6.2: Additional Dynamic Group

6.2.5 Use Cases and Query Results

Our prototype creates the HTTP query upon selection of different fields on the front-end. The predefined APIs include all combinations of each possible selection. Furthermore, the different API designs that our system is capable to handle are provided in section 5.3. The URLs can only contain queries that are constructed according to that design. In the following, we provide examples on what shape the URLs take and how the corresponding user interface is visualised.

Use Case 1

1. Query question: What are the *data types* that the *messages* inside my *MOM-based* system can have?

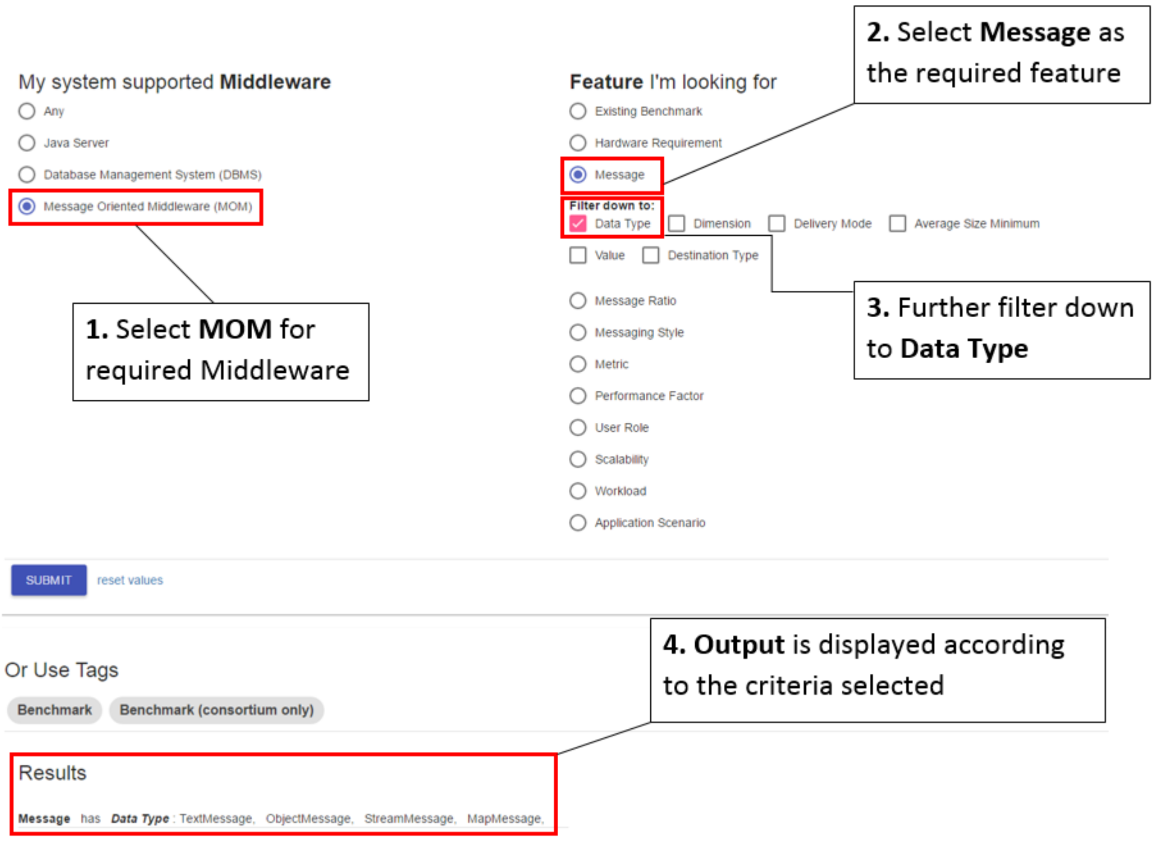
2. HTTP query: `localhost:8080/api/middlewares/56ef0de65f2f91fc163e8d1c/messages?populate=1&fields[]=dataType`

3. Result displayed: **Message** has **Data Type**: TextMessage, ObjectMessage, StreamMessage, MapMessage

4. Result in JSON:

```
"_id": "56ef0de65f2f91fc163e8d3a",
"dataType": [
"TextMessage",
"ObjectMessage",
"StreamMessage",
"MapMessage"
]
```

5. Query visualisation on the user interface:


My system supported **Middleware**
Any
Java Server
Database Management System (DBMS)
Message Oriented Middleware (MOM)
1. Select **MOM** for required Middleware
Feature I'm looking for
Existing Benchmark
Hardware Requirement
Message
Filter down to:
Data Type
Dimension
Delivery Mode
Average Size Minimum
Value
Destination Type
Message Ratio
Messaging Style
Metric
Performance Factor
User Role
Scalability
Workload
Application Scenario
2. Select **Message** as the required feature
3. Further filter down to **Data Type**
SUBMIT
reset values
Or Use Tags
Benchmark
Benchmark (consortium only)
4. Output is displayed according to the criteria selected
Results

Message has *Data Type* : TextMessage, ObjectMessage, StreamMessage, MapMessage,

Figure 6.7: Datatypes for Messages in a MOM-based System

Use Case 2

1. Query question: What *types* and *measures* of the *workload* have to be considered when benchmarking a *DBMS-based* system?

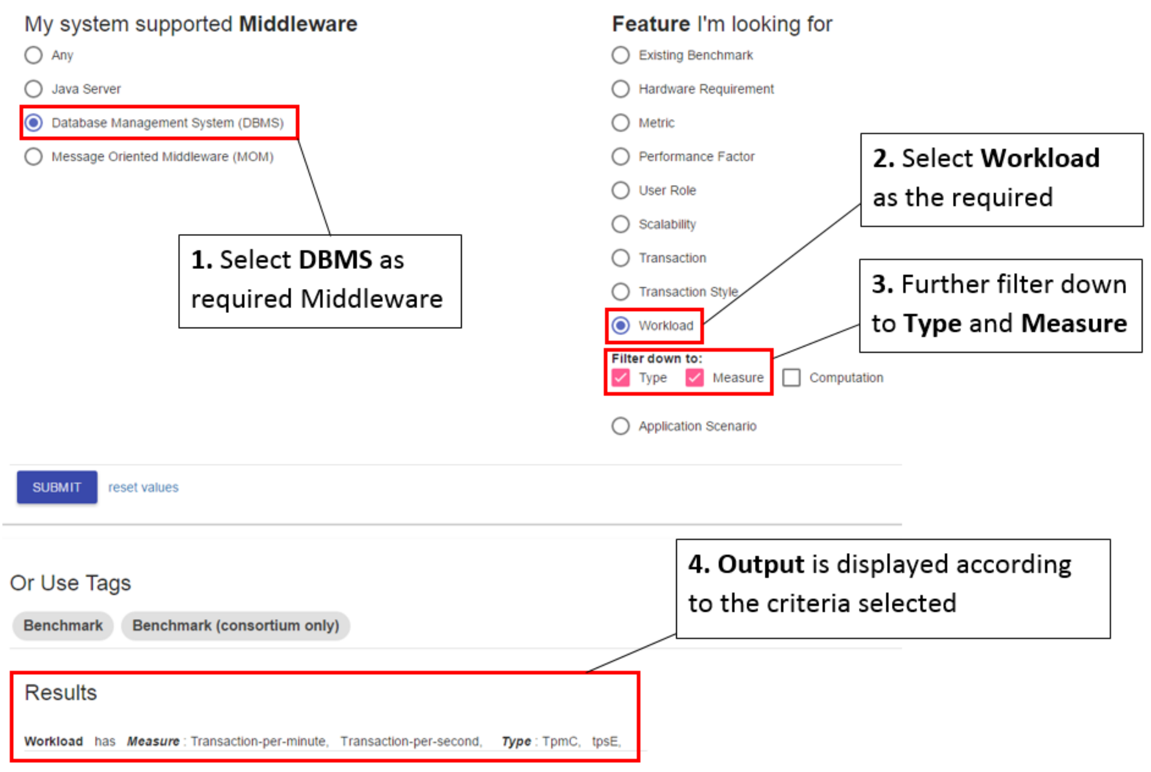
2. HTTP query: `localhost:8080/api/middlewares/56ef0de65f2f91fc163e8d1b/workloads?populate=1&fields[]=type&fields[]=measure`

3. Result displayed: **Workload** has **Measure**: Transaction-per-minute, Transaction-per-second, **Type**: TpmC, tpsE.

4. Result in JSON:

```
"_id": "56ef0de65f2f91fc163e8d23",
"measure": [
"Transaction-per-minute",
"Transaction-per-second"
],
"type": [
"TpmC",
"tpsE"
]
```

5. Query visualisation on the user interface:


1. Select DBMS as required Middleware
2. Select Workload as the required
3. Further filter down to Type and Measure
4. Output is displayed according to the criteria selected
SUBMIT
reset values
Or Use Tags
Benchmark
Benchmark (consortium only)

My system supported **Middleware**
Any
Java Server
[✓ Database Management System (DBMS)]
Message Oriented Middleware (MOM)

Feature I'm looking for
Existing Benchmark
Hardware Requirement
Metric
Performance Factor
User Role
Scalability
Transaction
Transaction Style
[✓ Workload]
Filter down to: [✓ Type] [✓ Measure] Computation
Application Scenario

Results

Workload has ***Measure*** : Transaction-per-minute, Transaction-per-second, ***Type*** : TpmC, tpsE,

Figure 6.8: Type and Measure of the Workload in a DBMS-based System

Use Case 3

1. Query question: What *types* and *measures* of the *workload* have to be considered when benchmarking a *DBMS-based* system?

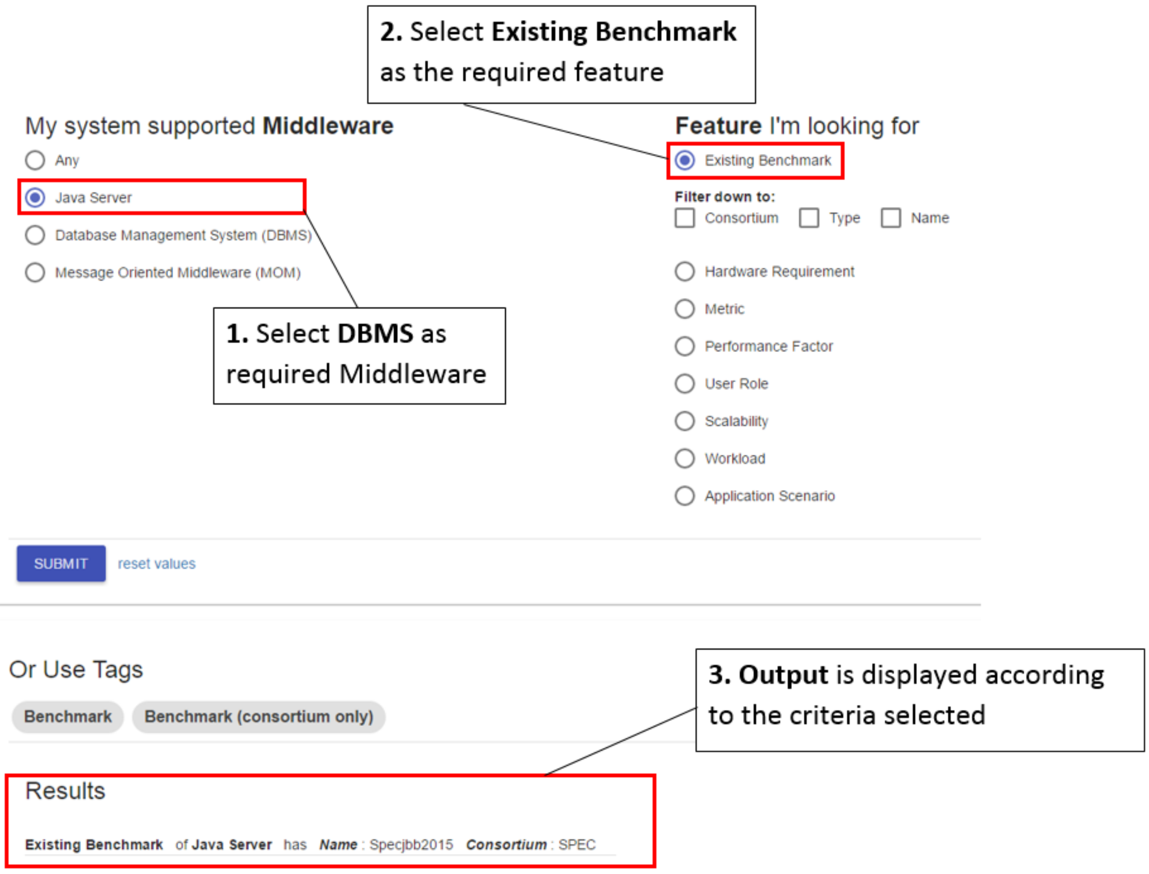
2. HTTP query: `localhost:8080/api/middlewares/56ef0de65f2f91fc163e8d1a/benchmarks`

3. Result displayed: **Existing Benchmark** of **Java Server** has **Name** : Specjbb2015 **Consortium** : SPEC

4. Result in JSON:

```
"_id": "56ef0de65f2f91fc163e8d20",
"middleware": {
"name": "Java Server"
},
"name": "Specjbb2015",
"consortium": "SPEC",
```

5. Query visualisation on the user interface:


2. Select **Existing Benchmark** as the required feature
My system supported **Middleware**
Any
Java Server
Database Management System (DBMS)
Message Oriented Middleware (MOM)
1. Select **DBMS** as required Middleware
Feature I'm looking for
Existing Benchmark
Filter down to:
Consortium
Type
Name
Hardware Requirement
Metric
Performance Factor
User Role
Scalability
Workload
Application Scenario
SUBMIT
reset values
Or Use Tags
Benchmark
Benchmark (consortium only)
3. Output is displayed according to the criteria selected
Results

Middleware: [✓ Java Server]

Feature: [✓ Existing Benchmark]

Existing Benchmark of **Java Server** has ***Name*** : Specjbb2015 ***Consortium*** : SPEC

Figure 6.9: Existing Benchmark for Java Servers

6.3 Installation and Configuration

One important aspect that we seek in our design and implementation is a straightforward installation of the framework in order to achieve a quick deployment. Therefore, we used different *package managers* and the open source distributed version control system *git* [git]. The prerequisite for this framework is to have the full MEAN software technology stack available.

The *Node Package Manager (NPM)* was used for installing Node.js modules and bower.js was used to install front-end components like html, css and js. One of the file names included in the *.gitignore* file is *Node_modules(library home)*, when cloning the project repository from *Git* for the first initialisation, the command *npm install* installs all dependencies from the *package.json* file inside the, automatically generated, */Node_modules* directory. Another file name mentioned in *.gitignore* is the *public/libs* directory, when pulling the whole project from *Git* for the first time, the command *bower install* installs all dependencies from the *bower.json* file inside the *public/libs* directory that will also be created automatically. As these two directories remain unchanged, it is best practice to avoid unnecessary load in terms of data transfers, therefore the exclusion feature in *git* is convenient. This requires to run only two commands when deploying the system for the first time and enables a speedy data transmission. The details about these two dependency management tools are discussed in 6.1.1.

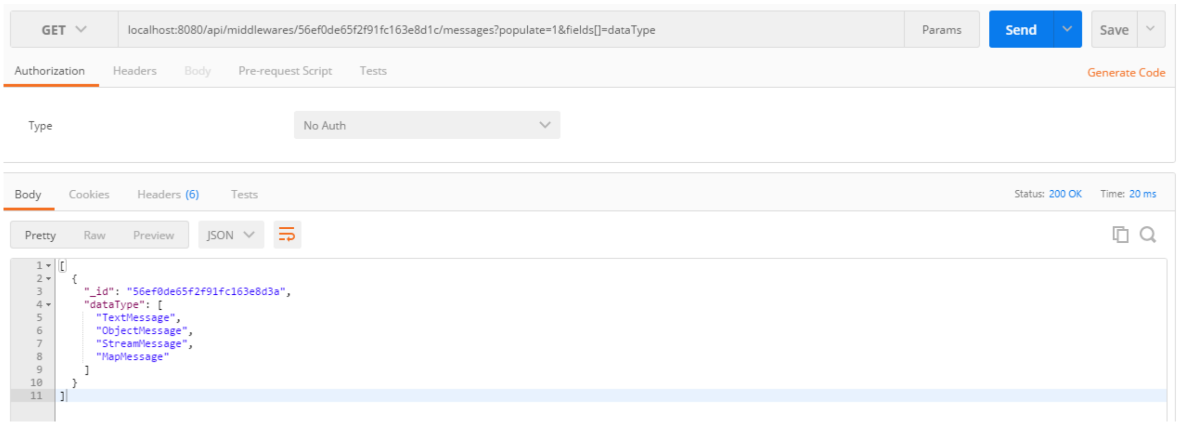
However, two aspects have to be considered every time the *DSS4MiddlewarePBenchmarking* is started. Firstly, the MongoDB has to be started through the command line interface running *mongod.exe* using its full path. Secondly, the *server.js* file has to run using the command *node server*, this will display the port where the web application is made available on the localhost. We worked on Ubuntu, Mac and Windows, therefore, the framework can be deployed on all three platforms without complications.

7 Validation and Evaluation

In this chapter, the prototype is validated with the help of POSTMAN API Client [pos] according to the functionalities mentioned in the use cases in section 4.2. Furthermore, the requirements discussed in section 4.1 must be fulfilled, thus the requirements need to be validated. In order to achieve the validation, the responses received for specific requests are matched with expected responses. A number of queries are defined and presented along with their outputs.

7.1 Validation of the Queries

The following figures (figures 7.1, 7.2 and 7.3) validate the three use cases from section 6.2.5 using the postman API client. The descriptions about the queries and the expected results are described in detail in section 6.2.5. All three use cases display the expected results and status code 200 is displayed for the indication of success.



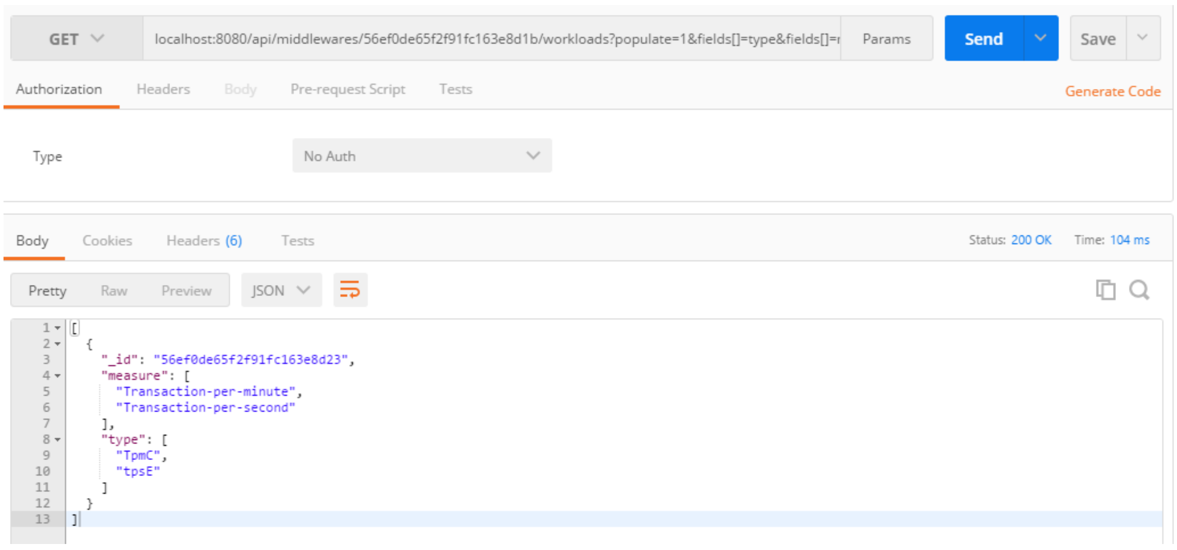
```
localhost:8080/api/middlewares/56ef0de65f2f91fc163e8d1c/messages?populate=1&fields[]=dataType
```

Type: No Auth

Status: 200 OK Time: 20 ms

```
[
  {
    "_id": "56ef0de65f2f91fc163e8d3a",
    "dataType": [
      "TextMessage",
      "ObjectMessage",
      "StreamMessage",
      "MapMessage"
    ]
  }
]
```

Figure 7.1: Validation of Use Case 1



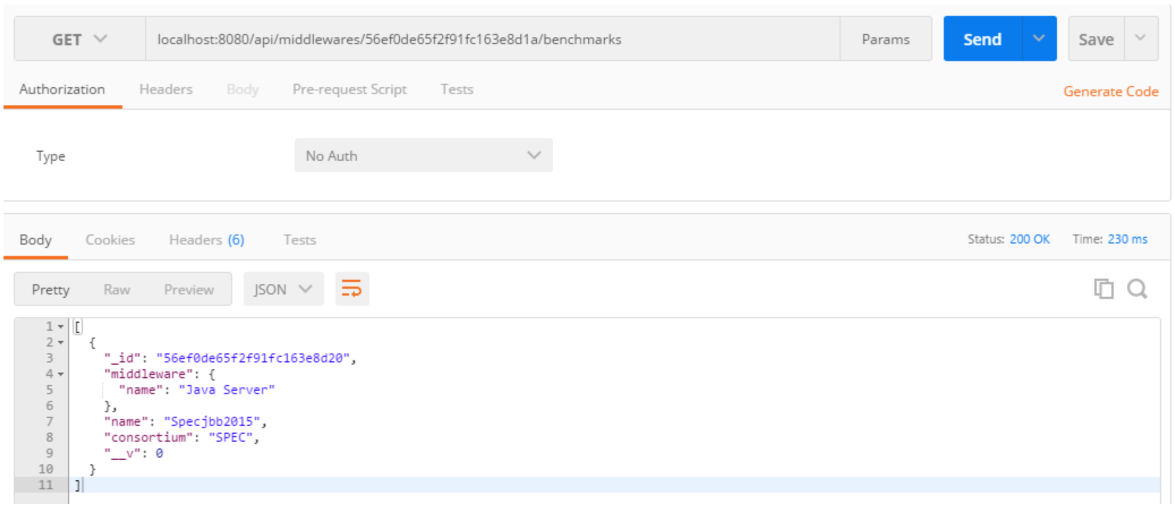
localhost:8080/api/middlewares/56ef0de65f2f91fc163e8d1b/workloads?populate=1&fields[]=type&fields[]=r

Type: No Auth

Status: 200 OK Time: 104 ms

```
[
  {
    "_id": "56ef0de65f2f91fc163e8d23",
    "measure": [
      "Transaction-per-minute",
      "Transaction-per-second"
    ],
    "type": [
      "TpmC",
      "tpsE"
    ]
  }
]
```

Figure 7.2: Validation of Use Case 2



localhost:8080/api/middlewares/56ef0de65f2f91fc163e8d1a/benchmarks

Type: No Auth

Status: 200 OK Time: 230 ms

```
[
  {
    "_id": "56ef0de65f2f91fc163e8d20",
    "middleware": {
      "name": "Java Server"
    },
    "name": "Specjbb2015",
    "consortium": "SPEC",
    "__v": 0
  }
]
```

Figure 7.3: Validation of Use Case 3

7.2 Validation of CRUD operations

The operations to create, update and delete resources are also validated using the POSTMAN API client. In the case of creating a resource, a POST method is used where it is necessary to define the location of the new resource and include its content in the body section. When the resource is successfully created, the status code 200 OK is displayed. In the case of updating a resource, a PUT method is used together with the URL of the resource to be updated and the content of required changes. Lastly, upon deletion of a resource, it is sufficient to define the URL of the required resource to be deleted.

7.3 Validation of REST APIs

Different operations have been performed on the resources in order to investigate the responses. Therefore, the REST API Client postman [pos] is used for validation purposes (see figure 7.4). The use cases defined in section 4.2 have led to the following responses:

1. Add, delete and update a resource - For the functionality of adding a new resource, a POST method is required that contains the resource details along with a status code that indicates success or failure of the request. In the case of removing a resource, a DELETE method is requested and the status code. Similarly, for updating a resource, PUT method is requested.

   a) URL: `/api/benchmarks`
   Method: POST
   POST all parameters that are required for adding the corresponding resource
   Status Code: 201 CREATED

   b) URL: `/api/benchmarks/:id`
   Method: DELETE
   URL parameter includes the id of specific benchmark
   Status Code: 200 OK

   c) URL: `/api/benchmarks/:id`
   Method: PUT
   URL parameter contains the ID of the resource to be updated
   Status Code: 200 OK

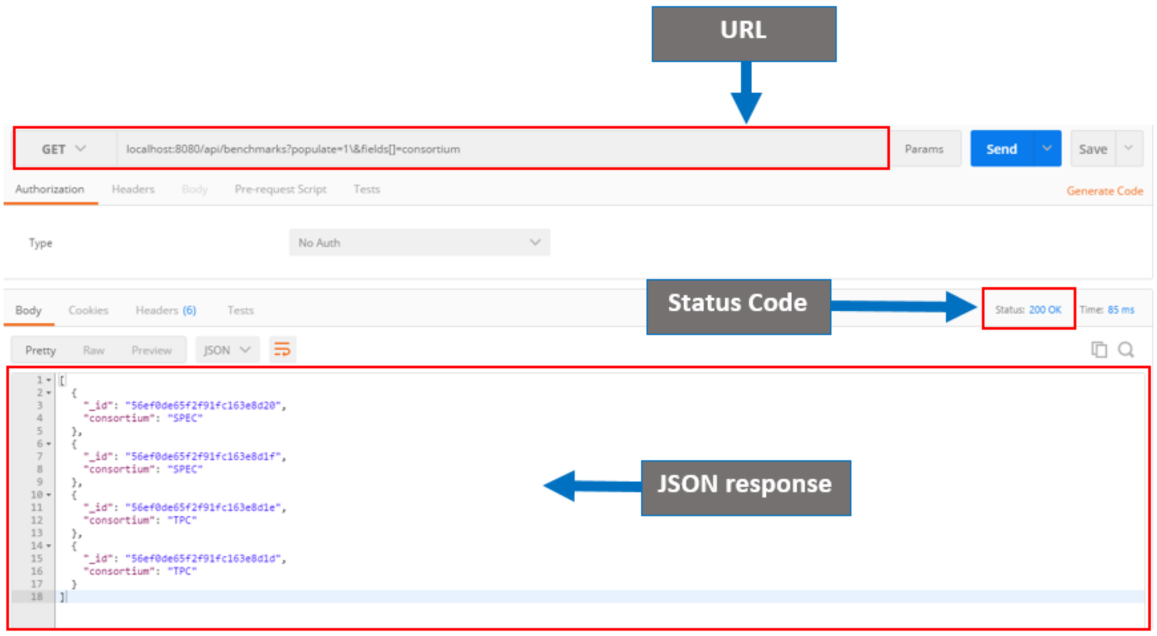
1. Retrieving results - For enabling to view results after selecting different criteria.

   a) URL: `/api/benchmarks?populate=1&fields[]=consortium`
   Method: GET
   The response displays the information that is contained in the URL applied in JSON format (see listing 7.1).
   Status Code: 200 OK

b)

```
"_id": "56ef0de65f2f91fc163e8d20",
"consortium": "SPEC"
},
{
"_id": "56ef0de65f2f91fc163e8d1f",
"consortium": "SPEC"
},
{
"_id": "56ef0de65f2f91fc163e8d1e",
"consortium": "TPC"
},
{
"_id": "56ef0de65f2f91fc163e8d1d",
"consortium": "TPC"
```

Listing 7.1: JSON Response format



```
localhost:8080/api/benchmarks?populate=1\&fields[]=consortium
```

Type: No Auth

Status: 200 OK Time: 85 ms

```
[
  {
    "_id": "56ef0de65f2f91fc163e8d20",
    "consortium": "SPEC"
  },
  {
    "_id": "56ef0de65f2f91fc163e8d1f",
    "consortium": "SPEC"
  },
  {
    "_id": "56ef0de65f2f91fc163e8d1e",
    "consortium": "TPC"
  },
  {
    "_id": "56ef0de65f2f91fc163e8d1d",
    "consortium": "TPC"
  }
]
```

Figure 7.4: Request and Response Using POSTMAN API Client

8 Conclusion and Future Work

The application of benchmarks, in a specific sector, is a vital approach for continuous improvement in regards to the effectiveness. Not only does it provide the means to understand the performance relatively close to competitors, it also enables the user to filter down into performance gaps and identify areas for improvements. There are numerous benchmarks on the market that are built for different softwares and hardwares. In terms of middleware benchmarking, there are standardised and open-source benchmarks available. However, the right selection of the required benchmark comes with the need of extensive research and crucial design decisions. In this thesis we focus on providing the means to assist benchmark users choose the suitable benchmark. We started by analysing standardised middleware benchmarks in order to identify critical decision points. After identifying the decision points, a taxonomy of the features and components in a benchmark was constructed. Lastly, a web-based Decision Support System (DSS) was implemented on the basis of this taxonomy. The DSS supports decision-making in terms of choosing the right features in the context of middleware benchmarking. There are many functionalities that can be used in a combination with web services, however, the main focus of easing the use of an application can be achieved by using an architectural style which is modifiable, portable and simple. The REST paradigm fulfills these aspects by allowing the user to identify the resources by using HTTP methods along with the required URLs. Below the answers and notable findings to the Research Objectives (RO) from Section 1.1 are stated.

Chapter 3 discussed the RO 1 1.1, since Section 3.1 described the benchmarking of middleware in general. Section 3.2 outlined the current trends with middleware benchmarking where the existing standard and open-source middleware benchmarks were addressed. Section 3.3 presented the need and features of DSS in general. Different types of DSS are depicted, whereas the *DSS4MiddlewarePBenchmarking* is categorised as a *knowledge-driven* DSS, based on the data in its knowledgebase. In regards to decision support systems in the field of middleware benchmarking, a brief statement is provided was section 3.3 that described the lack of this approach in the State of the Art.

In regards to the RO 2 1.1, Section 4.3 extensively covered all decision points which are relevant for decision-making. Firstly, the overall taxonomy described the skeleton on which the DSS is based on. Later, the focus laid on all relevant nodes inside the taxonomy. Each decision point was considered with a graphical diagram and a description. Moreover, possible outcomes of each decision point were also covered.

The answer to the RO 3 1.1 was discussed in Section 5.3 where the design rules for putting the RESTful paradigm into practice were considered. All APIs included in the system were designed according to the rules mentioned in that section. Furthermore, Section 4.1 investigated all functional and non-functional requirements that needed to be considered in the implementation of the system.

Finally, in regards to the RO 4 1.1, Chapter 6 covered all relevant aspects that needed to be considered for the prototypical implementation. All technologies and frameworks used for the system were described in Section 6.1. Alternatives that could have been considered and why they were not applied are also listed. Section 6.2 included all main parts of the implementation including screenshots. In Section 6.2.5 some functionalities provided by the system were also visualised using three use cases where queries along with their query results were shown.

8.1 Further Research

Based on this Master's thesis further research aspects regarding the decision support system for middleware benchmarking can be considered. The prototype developed in this thesis is a first step towards the establishment of an actual decision support framework.

In future, a more comprehensive and refined development could be considered. Due to the fact, that there was a literature gap in regards to middleware benchmarking as well as DSS for benchmarking in general, it was a challenge to assemble the contrasting data into a single knowledge base. However, we recommend to consider the expansion of the current taxonomy into a complex taxonomy where information regarding hardware configuration of the benchmark environment is also included. This means that information about concrete platforms is also added, despite the fact that these platforms change over time, the system could adapt manually to changing features. Later versions could also include more middleware systems, in order to provide a richer decision-making tool. Furthermore, to improve the data visualisation aspect the user interface can be designed in a more user-friendly way. The output of query results can be implemented in a table format. Moreover, the process of this work showed that selections made for certain decision points may constrain the selection in other decision areas. These constraints have a possible impact on the overall decision-making in the *DSS4MiddlewarePBenchmarking*. This problem can be addressed by further improving the dynamic and multi-criteria selection approach.

Bibliography

[A$^+$15] J. A. Arnold et al. How to Build a Benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE '15)*, pages 333–336. ACM, 2015.

[Act06] JMeter Performance Test, 2006. `http://activemq.apache.org/jmeter-performance-tests.html`.

[Agr15] S. Agrawal. *A service-oriented and cloud-based statistical analysis framework*. PhD thesis, Stuttgart, Universitaet Stuttgart, Masterarbeit, 2015.

[ASB10] S. Appel, K. Sachs, and A. Buchmann. Towards Benchmarking of AMQP. In *4th ACM International Conference on Distributed Event-Based Systems (DEBS' 10)*, July 2010.

[AW96] A. Avritzer and E. J. Weyuker. Deriving Workloads for Performance Testing. *Softw. Pract. Exper.*, 26(6):613–633, 1996.

[AYV10] T. L. Alves, C. Ypma, and J. Visser. Deriving metric thresholds from benchmark data. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10, 2010.

[B$^+$05] P. Brebner et al. Middleware benchmarking: approaches, results, experiences. *Concurrency and Computation: Practice and Experience*, 17(15):1799 —1805, Dec 2005.

[BC04] P. Bharati and A. Chaudhury. An empirical investigation of decision-making satisfaction in web-based decision support systems. *Decision support systems*, 37(2):187–197, 2004.

[BHM$^+$04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture, W3C Working Group Note. *World Wide Web Consortium*, Feb 2004.

[BPS07] H. K. Bhargava, D. J. Power, and D. Sun". Progress in Web-based decision support technologies. *Decision Support Systems*, 43(4):1083 – 1095, 2007. Special Issue Clusters.

[C$^+$15] I. K. Chaniotis et al. Is Node.Js a Viable Option for Building Modern Web Applications? A Performance Evaluation Study. *Computing*, 97(10):1023–1044, Oct 2015.

[CMM96] J. Czyzyk, M. P. Mesnier, and J. J. Moré. The Network-Enabled Optimization System (NEOS) Server, 1996.

[DAM01] V. A. A. Daniel A. Menasce. *Capacity Planning for Web Services: Metrics, Models, and Methods*, chapter Benchmarks and Performance Tests, pages 261–303. Prentice Hall, 2001.

[Dic15] J. Dickey. *Write Modern Web Apps with the MEAN Stack: Mongo, Express, AngularJS, and Node.js (Develop and Design)*. 2015.

[EK97] D. Elmuti and Y. Kathawala. An overview of benchmarking process: a tool for continuous improvement and competitive advantage. *Benchmarking for Quality Management & Technology*, 4(4):229–243, 1997.

[Erl05] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[F$^+$99a] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. June 1999. `http://www.hjp.at/doc/rfc/rfc2616.html`.

[F$^+$99b] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1, 1999.

[Fie00] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[GGP02] D. G. Gregg, M. Goul, and A. Philippakis. Distributing decision support systems on the WWW: the verification of a DSS metadata model. *Decision Support Systems*, 32(3):233 – 245, 2002.

[git] git. `https://git-scm.com/`.

[GMMS07] U. Guentzer, R. Mueller, S. Mueller, and R.-D. Schimkat. Retrieval for decision support resources by structured models. *Decision Support Systems*, 43(4):1117–1132, 2007.

[HSC99] K. Hemant, S. Suresh, and H. Craig. Beyond Spreadsheets: Software for Building Decision Support Systems. *IEEE Computer*, 32(3):31–39, 1999.

[Hup09] K. Huppler. *Performance Evaluation and Benchmarking: First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, chapter The Art of Building a Good Benchmark, pages 18–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Inc16] W. Inc. benchmarking, 2016. `http://www.businessdictionary.com/definition/benchmarking.html`.

[JBo06] JBoss. JBoss JMS New Performance Benchmark, 2006. `https://developer.jboss.org/welcome`.

[Kou06] S. Kounev. *Performance Engineering of Distributed Component-based Systems*. Shaker Verlag, 2006.

[Kra03] S. Krakowiak. What is Middleware. *OW2 Consortium*, 2003.

[KS09] S. Kounev and K. Sachs. Benchmarking and Performance Modeling of Event-Based Systems. *IT - Information Technology Heft 5 / 2009*, 5, Sep 2009.

[Law05] G. Lawton. LAMP Lights Enterprise Development Efforts. *Computer*, 38(9):18–20, 2005.

[LL09] K. B. Laskey and K. Laskey. Service Oriented Architecture. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):101–105, 2009.

[LR01] A. Leff and J. T. Rayfield. Web-Application Development Using the Model/View/Controller Design Pattern. In *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing (EDOC '01)*, pages 118–127, Washington, DC, USA, 2001. IEEE Computer Society.

[Ma05] K. J. Ma. Web services: what's real and what's not? *IT Professional*, 7(2):14–21, Mar 2005.

[Mar99] D. Marshall. Remote Procedure Calls, May 1999. `https://www.cs.cf.ac.uk/Dave/C/node33.html`.

[Mas11] M. Masse. *Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, 2011.

[Mea14] Mean. The Friendly and Fun Javascript Fullstack for your next web application, 2014. `http://mean.io`.

[Men] F. Menge. Enterprise Service Bus. `https://programm.froscon.org/2007/attachments/15-falko_menge_-_enterpise_service_bus.pdf`.

[Mon15] Mongo. MySQL vs. MongoDB: Choosing a Data Management Solution, 2015. `https://www.javacodegeeks.com/2015/07/mysql-vs-mongodb.html`.

[Mon16] Mongo. MongoDB and MySQL Compared, 2016. `https://www.mongodb.com/compare/mongodb-mysql`.

[NW05] E. W. Ngai and F. Wat. Fuzzy decision support system for risk analysis in e-commerce development. *Decision support systems*, 40(2):235–255, 2005.

[OvVSH16] W. Oude Nijeweme-d'Hollosy, L. van Velsen, R. Soer, and H. Hermens. Design of a web-based clinical decision support system for guiding patients with low back pain to the best next step in primary healthcare. In *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016)*, volume 5: HEALTHINF, pages 229–239, Portugal, February 2016. SCITEPRESS ? Science and Technology Publications.

[PJC15] A. J. Poulter, S. Johnston, and S. Cox. Using the MEAN Stack to implement a RESTful service for an Internet of Things Application. 2015.

[pos] postman. Postman REST API Client. `http://www.ibm.com/developerworks/library/ws-restful/`.

[Pow02] D. Power. *Decision Support Systems: Concepts and Resources for Managers*. Quorum Books, 2002.

[Pow04] D. J. Power. Specifying An Expanded Framework for Classifying and Describing Decision Support Systems. *The Communications of the Association for Information Systems*, 13(1):52, 2004.

[PSB15] D. J. Power, R. Sharda, and F. Burstein. *Decision Support Systems*. John Wiley and Sons, Ltd, 2015.

[resa] rest. `http://www.restapitutorial.com/httpstatuscodes.html`.

[RESb] REST. REST Architectural Goals. `http://whatisrest.com/rest_architectural_goals/index`.

[RHW14] C. Rock, S. Harrer, and G. Wirtz. Performance Benchmarking of BPEL Engines: A Comparison Framework, Status Quo Evaluation and Challenges. In *Proceedings of the 26th International Conference on Software Engineering & Knowledge Engineering (SEKE '14)*, 2014.

[Rod] A. Rodriguez. `http://www.ibm.com/developerworks/library/ws-restful/`.

[SJ80] R. H. Sprague Jr. A framework for the development of decision support systems. *MIS quarterly*, pages 1–26, 1980.

[SKAB09] K. Sachs, S. Kounev, S. Appel, and A. Buchmann. Benchmarking of Message-oriented Middleware. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS '09)*, New York, NY, USA, 2009. ACM.

[SKBB07] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Workload characterization of the SPECjms2007 benchmark. In *Formal Methods and Stochastic Models for Performance Evaluation*, pages 228–244. Springer, 2007.

[SKBB09] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Performance evaluation of message-oriented middleware using the SPECjms2007 Benchmark. *Performance Evaluation*, 66(8):410–434, Aug 2009.

[SKCB07] K. Sachs, S. Kounev, M. Carter, and A. Buchmann. Designing a Workload Scenario for Benchmarking Message-Oriented Middleware. In *Proceedings of the 2007 SPEC Benchmark Workshop*, Austin, Texas, USA, January 2007. SPEC.

[SPE95] SPEC. Standard Performance Evaluation Corporation, 1995. `https://www.spec.org/spec/`.

[SPE07] SPEC. SPECjms2007, 2007. `https://www.spec.org/jms2007/`.

[SPE15] SPEC. SPECjbb2015, 2015. `https://www.spec.org/jbb2015/`.

[SRL$^+$15] M. Skouradaki, H. D. Roller, F. Leymann, V. Ferme, and C. Pautasso. Technical Open Challenges on Benchmarking Workflow Management Systems. In U. of Stuttgart, editor, *Technical Report of the Symposum on Software Performance (SOSP) 2014*, pages 1–8. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Stuttgart, January 2015.

[T$^+$04] J. Tracey et al. Workflow Management System, Sep 2004. `https://www.google.com/patents/US6798413`.

[The97] The Open Group. CDE 1.1: Remote Procedure Call, May 1997. `http://pubs.opengroup.org/onlinepubs/9629399/chap6.htm`.

[Tig12] P. J. Y. Tigl. Middleware for Ubiquituous Computing, 2011-2012.

[TPC92a] TPC. TPC-C, 1992. `http://www.tpc.org/tpcc/`.

[TPC92b] TPC. TPC-E, 1992. `http://www.tpc.org/tpce/`.

[TPC92c] TPC. Transaction Processing Performance Council, 1992. `http://www.tpc.org/information/benchmarks.asp`.

[Vin02] S. Vinoski. Where is middleware. *Internet Computing, IEEE*, 6(2):83–85, Mar 2002.

[W3S] W3SCHOOLS. HTTP Status Messages. `http://www.w3schools.com/tags/ref_httpmessages.asp`.

All links were last followed on May 2, 2016

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, May 2, 2016 ______________________

(Name)