Linnæus University
Sweden

Degree project

Evaluation and Implementation of Machine Learning Methods for an Optimized Web Service Selection in a Future Service Market

Author: Philipp Karg
Supervisor: Prof. Dr. Welf Löwe
External Supervisor: Prof. Dr. Sven Martin, University of Applied Sciences Karlsruhe

Date: 2014-08-31

Course Code: 4DV01E, 60 credits
Level: Master

Department of Computer Science

Statement of Authorship

I certify that this master thesis and the research work to which it refers are the product of my own work and that it has not been submitted for a degree at any other university. Any ideas or quotations from the work of other people published or otherwise are fully acknowledged.

Place, Date	Philipp Karg

Abstract

In future service markets a selection of functionally equal services is omnipresent. The evolving challenge, finding the best-fit service, requires a distinction between the non-functional service characteristics (e.g., response time, price, availability). Service providers commonly capture those quality characteristics in so-called Service Level Agreements (SLAs). However, a service selection based on SLAs is inadequate, because the static SLAs generally do not consider the dynamic service behaviors and quality changes in a service-oriented environment. Furthermore, the profit-oriented service providers tend to embellish their SLAs by flexibly handling their correctness. Within the SOC (Service Oriented Computing) research project of the Karlsruhe University of Applied Sciences and the Linnaeus University of Sweden, a service broker framework for an optimized web service selection is introduced. Instead of relying on the providers' quality assertions, a distributed knowledge is developed by automatically monitoring and measuring the service quality during each service consumption. The broker aims at optimizing the service selection based on the past real service performances and the defined quality preferences of a unique consumer.

This thesis work concerns the design, implementation and evaluation of appropriate machine learning methods with focus on the broker's best-fit web service selection. Within the time-critical service optimization the performance and scalability of the broker's machine learning plays an important role. Therefore, high-performance algorithms for predicting the future non-functional service characteristics within a continuous machine learning process were implemented. The introduced so-called foreground-/background-model enables to separate the real-time request for a best-fit service selection from the time-consuming machine learning. The best-fit services for certain consumer call contexts (e.g., call location and time, quality preferences) are continuously pre-determined within the asynchronous background-model. Through this any performance issues within the critical path from the service request up to the best-fit service recommendation are eliminated. For evaluating the implemented best-fit service selection a sophisticated test data scenario with real-world characteristics was created showing services with different volatile performances, cyclic performance behaviors and performance changes in the course of time. Besides the significantly improved performance, the new implementation achieved an overall high selection accuracy. It was possible to determine in 70% of all service optimizations the actual best-fit service and in 94% of all service optimizations the actual two best-fit services.

Keywords

optimized web service selection, central service broker, service oriented computing, non-functional service characteristics, performances, machine learning, continuous service optimization

Table of Contents

List of Figures

List of Tables

1 Introduction

This first chapter serves as introduction into this thesis work as well as presentation of the thesis emphasis. For an understanding of the topic, first background details are given and the motivation is highlighted. Afterwards the thesis problem is defined and the derived objectives are presented. Finally, a brief overview of this thesis structure is given.

1.1 Background and Motivation

This Master Thesis is part of the research cooperation with the Linnaeus University in Växjö, Sweden and the Karlsruhe University of Applied Sciences, Germany. The joint work takes place in a service-oriented setting with the assumption that there will be a selection of service providers with functionally equal services in future service markets.

> *"With Service-Oriented Computing (SOC), Software as a Service (SaaS), and Cloud Computing, visions of a market of services where functionalities can be dynamically and ubiquitously consumed have emerged. Service consumers bind and use services on a dynamic basis." [1]*

Services offering the same functionality only distinguish between their non-functional characteristics (e.g., response time, price and availability). The service providers commonly capture these quality attributes in so-called Service Level Agreements (SLAs). Within the research of Kirchner, et al. 'Service Level Achievements – Distributed Knowledge for Optimal Service Selection' [1] it is claimed that a service selection based on the quality promises of the service providers defined as SLAs, cannot be trusted in general. Due to the fact that service provider are profit-oriented a certain tendency evolves to embellish the SLAs and to flexibly handle the correctness of the defined service quality characteristics. Furthermore, the static SLAs do not consider the dynamic behavior and quality changes of services in a service-oriented environment. For example, when services become more popular they are consumed more often and therefore, they possibly show availability problems. Such behaviors are hard to capture within SLAs. Overall, consulting SLAs for service quality provision is not optimal because they do not always reflect the current non-functional service characteristics. The approach of Kirchner, et al., addresses this problematic by monitoring and dynamically measuring the service quality during service consumption. Hereby the development of a distributed knowledge allows a detailed insight in the real and current service characteristics [1].

As mentioned before, the existence of multiple services with the same functionality requires a distinction between the non-functional characteristics. The evolving challenge is to find services with best-fit quality for a unique consumer. The approach of the mentioned research aims at automatically selecting best-fit services based on the measured service quality performances. Objective hereby is the extension of the classical service binding of consumer and provider. The participants in this scenario are service consumers and service providers as well as a central service broker. The broker serves as mediator with the core task automatically binding service consumer and provider. The term service is used for the general service functionality, whereas the service instance defines the service implementation having a certain service functionality. For example, a weather service provides weather predictions as a functional service. The service instance would be a specific web service from a service provider such as Yahoo!.

Figure 1.1 outlines the core features of the overall framework with the process for optimizing the service selection within a dynamic service binding.

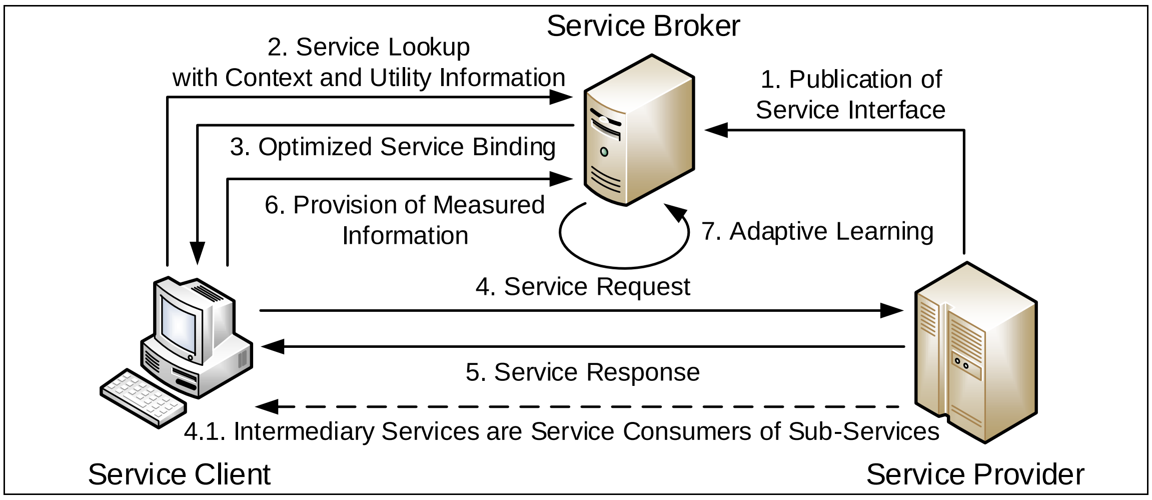

Service Broker
2. Service Lookup with Context and Utility Information
1. Publication of Service Interface
3. Optimized Service Binding
6. Provision of Measured Information
7. Adaptive Learning
4. Service Request
5. Service Response
4.1. Intermediary Services are Service Consumers of Sub-Services
Service Client
Service Provider

Figure 1.1: Extended dynamic service binding within a service oriented architecture aiming at optimizing the service selection (Kirchner, et al. [1])

The following paragraph gives a brief description of the single process steps. The service providers have to initially publish their service interfaces to the broker (step 1). When service consumers lookup a service (step 2) the service broker provides an optimized service binding (step 3). After the service is requested and consumed (step 4 and 5) the clients transmit their measured end-user quality characteristics to the service broker (step 6). Finally, the broker adapts the learning process based on the new measured information (step 7).

This approach for an optimized web service selection significantly distinguishes from a classical service binding. In classical service binding there is no knowledge of measured service performances in general, whereas this new approach aims to optimize the service selection for a unique consumer based on the knowledge of past service calls. The consumer defines the so-called utility function, which are user specific preferences for the service quality characteristics. Within the service lookup (step 2) corresponding to the consumer's context and the utility, the broker provides a best-fit service. The central service broker has the leading part in optimizing the service selection. The input of the optimization is the measured end-user performances, which are monitored on the client-side and transferred as feedback ticket to the service broker (step 6). These tickets include call context attributes (e.g., location, call time), the consumers' preferences (utility function) and the performances of the service call (e.g., response time, availability). With this information, the service broker is capable of selecting the services with best-fit quality in respect to the consumer's context and the consumer's preferences. [1]

1.2 Problem Definition

This section analyzes the objectives of the presented SOC framework, described more general before, with respect to the machine learning part. The thesis focuses on the overall optimization of the service binding (step 3) with the adaptive machine learning (step 7) on part of the central service broker.

For the problem definition the well-known process model *CRISP-DM* (Cross Industry Standard Process for Data Mining), for approaching and solving typical data mining problems, is used. *CRISP-DM* includes several phases, which are as follows: business understanding, data understanding, modelling, evaluation and deployment [2]. The first two phase's business and data understanding are examined in this section, whereas the other phases are direct or indirect objectives of the design, implementation and evaluation chapters.

Within the initial phase, focuses are the project objectives and requirements from a business perspective. Transferred to the SOC setting, the initial situation is an existent service market with a selection of multiple services offering the same functionality but with different service quality characteristics. The consequent selection problem, which service best-fit the service quality preferences of the consumer, needs to be solved. The main objective from business perspective is the improvement of the service quality for a unique consumer by optimizing the service selection. The supporting technical side for achieving an optimized selection consists of two mandatory steps. First, continually learn from past service calls to predict the service performances for similar contexts and secondly, finding a best-fit service considering the unique service quality preferences of a consumer. The subsequent part further describes those two steps.

The learning and prediction takes place based on historically collected data from past service calls. For an understanding of the existent data, the attributes are briefly described (see Section 3.3.3 for a detailed description of the attributes). The collected data represents the so-called feedback tickets that are submitted by a consumer after each service consumption. Those tickets contain information of the consumers call context as well as information of the performance measurements. The call context attributes such as the call time, the call location, etc. are input attributes for the learning, whereas the performance measurement such as the response time, the availability, etc. are the label attributes, which the machine learning aims to predict.

Within the second part, selecting a best-fit service to the corresponding consumer preferences, the concept of utility functions is applied. This approach allows the definition of the consumer's service quality preferences and therefore, the determination of an optimization goal. The consumer sets the preferred weight of each performance attribute and with simple additive weighting, a consumer's utility can be determined. As an example, here the definition of a consumer's utility function:

$$U_{(\textit{response time, availability})} = 0.7 \times [\textit{response time}] + 0.3 \times [\textit{availability}]$$

Within this utility function, the needs for a good response time are greater than the needs for a high availability. Overall the utility function allows to calculate the utilities of each possible service and accordingly rank the service to get a best-fit service [1].

1.3 Objectives

The main objectives of this Master's thesis are the design, implementation and evaluation of appropriate machine learning methods for an optimized web service selection. The current learning approach of the service broker is based on a classical batch decision tree learning algorithm that has some restrictions in regards to the service-oriented, dynamic and performance-critical setting. This Master's thesis is tackling those drawbacks.

The preliminary objectives are, first the determination of the broker restrictions within the current implementation, second the assessment of the SOC challenges with the definition of corresponding evaluation criteria, third the pre-selection of appropriate machine learning methods and libraries, which are addressing those restrictions and challenges. The core challenge is the speed and the scalability of the machine learning methods with regard to the performance-critical path from a consumers request to the response for an optimized service. Other aspects such as the ability of incremental learning also have to be examined. The final preliminary objective is the creation and preparation of appropriate test data for training and testing the learning algorithms.

After the preliminary, the following final objectives are pursued to obtain the optimized web service selection. First, a conceptual design has to be outlined concerning the challenges of the SOC setting. Second, the appropriate learning methods have to be conceptually implemented and integrated. Third, the new learning approach has to be evaluated with respect to the defined evaluations criteria.

In sum, the objectives of this Master's thesis are as follows:

- Scientific search for alternative machine learning methods, which are addressing the current challenges
- Definition of appropriate evaluation criteria in context of the SOC setting
- Creation and preparation of appropriate test data
- Design of a concept for the machine learning and the web service selection
- Implementation of the machine learning methods and the best-fit web service selection and integration into the SOC framework
- Evaluation of the machine learning with regards to the defined evaluation criteria

1.4 Goal Criteria

This section presents measurable/observable success criteria for this thesis work concerning the optimized web service selection. For each criteria, a corresponding assessment method is listed.

<u>High-Performance Service Determination</u>
This work aims at optimizing the critical path for achieving a high-performance service determination.
Assessment: time duration from requesting to receiving the broker's best-fit service selection (in milliseconds)

<u>High Service Selection Accuracy</u>
The optimized web service selection should achieve a high service selection accuracy.
Assessment: percentage of correctly determined actual best-fit services (in percent)

<u>Continuous Machine Learning Process</u>
The service selection should be based on a continuous machine learning process.
Assessment: capability of the machine learning continuously processing and learning from past measured service calls

Performance Change Adaption
The machine learning should be capable detecting performance changes in the course of time.
Assessment: capability of the machine learning detecting performance changes

1.5 Ethical Considerations

This subsection discusses ethical aspects considering potentially threads or negative impacts on the individuals of the SOC framework and means to avoid such threads or impacts.

In the literature, the common research field *ethics of artificial intelligence* can be divided into *roboethics* and *machine ethics*. For this thesis work the field of *machine ethics* is more suitable concerning to ensure that the *'behavior of machines toward human users, and perhaps other machines as well, is ethically acceptable'* [3]. Other research fields such as the *ethics of data mining* focus on the input data and the ethical usage of the retrieved results. Data mining, when applied to people, is frequently used to discriminate. Especially when raising racial, sexual, religious, etc. data attributes about a person [4]. The next part discusses the collection of data and the behavior of the central broker from an ethical point of view.

Within the framework the collected data are non-functional properties which can be seen as the real performance achievements of the providers. From provider's perspective, the collected performance details are not sensitive since the providers generally publish these details within their SLA's. On the other hand, from consumer's perspective a complete insight on the service consumption can be applied. Thereby not only consumer information about the achieved service performance but also about when a service was consumed, how often and from where, can be retrieved. Since the current broker implementation does not run in a productive environment and the test data is simulated and not collected from real consumers, no ethical concerns are involved within this thesis. In a later stadium of the SOC project, the underlying data could be anonymized or the data could be only stored for a defined period and thereafter be deleted.

The service oriented architecture is based on a machine to machine communication and human users are not directly involved. The central service broker continually learns and takes autonomous decision on the service selection. It cannot be excluded that the service broker makes wrong decisions. In the case of a wrong decision, there would be a possibly negative impact on the achieved service quality but not on the service functionality. Within the open market scenario a market-oriented competition is omnipresent. Services with best-fit quality are preferred and recommended. The discrimination or moreover ranking of service providers thereby is a legitimate approach. In the SOC setting the end consumer, have to trust the central service broker. However, since the broker is an observing instance and operated separately from the service providers, the objectivity and independency is guaranteed. Overall, it can be stated that the brokers behavior towards the framework participants and general approach is ethically acceptable.

1.6 Structure

The overall structure of the study takes the form of six chapters, including this introductory chapter. Chapter Two begins by laying out the current restrictions of the machine learning, and examines the core challenges given from the central broker model. For each challenge, corresponding evaluation criteria are defined. Afterwards a pre-selection of machine learning methods and libraries, with respect to defined criteria, is applied. Within a proof of concept, the general usage and potential of the pre-selected methods and libraries are tested. The third chapter presents the conceptual design with the underlying architecture of the optimized web service selection. Chapter Four presents the implementation specifics and objectives of the overall machine learning process. In the fifth chapter, an overall evaluation of the machine learning methods as well as of the optimized web service selection is applied. The final chapter gives the conclusion with a brief summary of the implementation and the resulting benefits as well as a critique analyses and given current restrictions.

2 SOC Project Machine Learning

In this chapter, first the current restrictions of the central broker implementation are examined. Second, corresponding challenges, derived from the general concept of the central service broker as well as from the current restrictions, are defined. Furthermore, for each challenge appropriate criteria are defined for the later evaluation. Finally, a pre-selection of appropriate machine learning methods and libraries is applied and within a proof of concept, the usage and potential is tested in advance.

2.1 Current Broker Restrictions

The current central broker machine learning is implemented within the data mining environment RapidMiner. For predicting the service performances several RapidMiner processes were modeled for pre-processing, training and testing the decision tree learning. Loading and executing these RapidMiner processes via the central broker Java project integrates the RapidMiner machine learning. For evaluating the decision tree learning specific test data was generated from real-world web services [5]. The following part examines the restrictions of the current broker implementation.

<u>Batch Learning</u>
The major restriction of the current approach is the classical batch learning. This type of learning requires to train on the overall input data and therefore, has a huge impact on the time consuming training of the prediction model. The batch learning is challenged with strong linear increase of computation time by increasing data volume and increasing service candidates. In the SOC setting it has to be assumed, that there will be produced a high volume of feedback measurement tickets. Therefore, this learning has to be questioned in general and other learning approaches in the area of incremental learning have to be considered.

<u>Pre-Processing</u>
In some cases, the discretization of the current implementation is not optimal regarding the value range division. The attribute 'daytime', for example is aggregated to four hours ranges. If there are many feedback tickets submitted within the four hours range, the data basis getting too broad and the prediction less meaningful.

<u>Automation</u>
In general, the handling of the RapidMiner processes is not flexible enough and the possibilities for automating the machine learning process are limited. The processes are modeled inside RapidMiner, saved as XML and loaded via Java. A solution for the automation, and especially the handling of new service candidates, would be to duplicate an existing process and afterwards accordingly manipulate the XML. This would not be a proper solution, moreover causing a significant overhead especially if there are many service candidates.

<u>Test Data</u>
For evaluating the decision tree learning in the current broker implementation, an *'initial set of measurement tickets (...) [were] created within a real world scenario of public and private web service providers'* [5]. These web services were implemented to cover different functionality. For example, the public Stock Quote service responses with the current stock quote of a specific stock and the public Weather

service responses with the current weather. The dummy service File Digest is sorting all lines of the submitted input file alphabetically and the Salutation service responses with a salutation depending on the current time of the day. In this data generation scenario a total of approximately 110 thousand feedback tickets, including call context attributes (call time, consumers external IP and message size of the request) as well as quality attributes (response time and availability), were created during a 25-day time frame and with a total of 15 unique service candidates. [5]

Service Candidate	Records	Response time in ms		Message size in ms		Availability in %
		Mean	Deviation	Mean	Deviation	
File Digest	9786	4477	3567	453153	201929	99,97
Salutation	30350	479	1380	305	0,6	88,72
Stock Quote	29625	1051	2888	1003	9,2	93,19
Weather	42405	756	2172	699	4,2	98,90

Table 2.1: Data statistics of the current test data

Table 2.1 outlines the statistics of these feedback tickets. The data for each service instance was aggregated and then summarized within the service functionality. Noticeable is the significant deviation of the attribute response time. The deviation of each service (except File Digest) is almost three times as big as the average response time. Because of the many outliers, there is a certain arbitrariness. The message size of the services Salutation, Stock Quote and Weather has a steady development and no remarkable deviation. The attribute availability is going from 85% up to 99%. Although the test data is from collected real-world dummy services there could not be found any patterns or cyclic behavior nor a performance change over time.

A closer look into the record count shows that the data volume is in general low. The Weather service for example has only 8.5 thousands records per service instance (in mean), which is the maximum instance count trained for a single decision tree learning model. For more details, see Appendix A.1.

This chapter shows that the current central service broker has some significant limitations regarding both the machine learning and the existing test data.

2.2 SOC Project Challenges

The SOC setting is situated in a dynamic and performance-critical environment. The machine learning approach for an optimized web service selection is confronted with several challenges. For an understanding of the problem situation, the challenges have to be examined first.

According to Han et al., ([6, 7]) there are several aspects for evaluating machine learning methods: speed, accuracy, scalability, robustness and interpretability. In the next subsections, these aspects (except interpretability[1]) were further discussed, not only concerning the method evaluation, but also as challenge for the

[1] The aspect interpretability is not further considered because within an automated machine learning process an interpretation of the prediction model does not play an import role, but the performances and results does.

overall machine learning within the optimized web service selection. For each challenge first, a definition is given and thereafter the specific requirements for the SOC setting are outlined. Afterwards corresponding criteria are listed, which are applied for the later evaluation.

2.2.1 Speed

This challenge describes how efficient the machine learning method performs concerning the training and prediction time. Furthermore, this aspect also concerns the overall machine learning process as 'critical path' from end-user side, meaning the time from the consumers request for a best-fit web service until receiving the response.

In the SOC setting the speed in general and the processing of requests in specific, is a core challenge. The optimized web service selection is requested in real-time before a service binding takes place and therefore, the response time plays an important role. For example, if the consumer's preferences are strongly based on performance attributes and the request for a best-fit service takes longer than the actual service consumption, the optimized web service selection does not have an additional benefit and the existence has to be questioned. This example shows the significant importance of the performance.

As described in 2.1 the restriction given with the classical batch learning has to be solved. Incremental learning should therefore be considered for a continual real-time processing and learning without training on the overall data set.

Evaluation Criterion

- How long does the training and prediction take?

2.2.2 Accuracy

This aspect describes how effective the machine learning method performs and therefore, the ability to correctly classify a class label or predict a class value. [7]

The predictive accuracy plays also an important role in the SOC setting but compared to the criteria speed it has a lower priority. A model is not useful if it has the best accuracy but at the same time the worst computation time. The optimum is always a tradeoff between a good accuracy and a good speed.

Evaluation Criterion

- How accurate is the machine learning method?

2.2.3 Scalability

This criterion describes the ability of the machine learning method to scale up, in other words, the ability to build the method efficiently on a large amount of data [7].

As mentioned in the introductory chapter, for each service consumption the central broker framework receives a feedback ticket. It is assumed that the framework will produce a high amount of data. Table 2.2 shows an overview of the predicted number of tickets created with different number of participants.

Participants [2]	# Tickets per week	# Tickets per month	# Tickets per half year
100	14.000	56.000	336.000
500	70.000	280.000	1.680.000
2000	280.000	1.120.000	6.720.000
5000	700.000	2.800.000	16.800.000
15000	2.100.000	8.400.000	50.400.000

Table 2.2: Predicted number of submitted feedback tickets for corresponding number of participants

The prediction shows that there possibly will be created more than 2 million tickets per week with 15 thousand participants and 20 service consumptions per participant. This will continue up to over 50 million tickets during a six months' time period. Because of this high volume of data, the machine learning process should be capable to handle such large amount of data as well as process the data in a foreseeable amount of time. Therefore, it is mandatory to examine incrementally learning models, where each instance is learned at a time incrementally.

Evaluation Criteria

- How does the algorithm perform on large data sets?
- Is the machine learning method capable of incremental learning?

2.2.4 Robustness

This criterion describes the ability to make correct classifications and predictions given noisy data or data with missing values [7]. Furthermore, this criterion also includes the ability of the machine learning to automatically run in a changing environment.

The machine learning process in the SOC project is situated within a dynamic and changing environment. New service candidates and new attributes continually come and go.

Evaluation Criteria

- How is a continuously changing environment with new service candidates and new attributes handled?
- Is the machine learning method able to handle noisy data such as outliers and attributes with missing values?

2.3 Selection of Methods and Libraries

A preliminary objective of this thesis was the search for machine learning methods, which addresses the current challenges and restrictions described in the previous sections. Because there exist a significant number of machine learning methods as well as data mining libraries it is hardly possible to evaluate all of them. Based on the defined requirements a pre-selection is applied to limit the number of methods and libraries and therefore, reducing the time of the evaluation and the testing. Within a proof of concept, the potential and usage of the selected machine learning methods and libraries are pre-evaluated.

[2] 20 Consumptions per participant and day

2.3.1 Method Pre-Selection

In the area of machine learning there are two main categories for learning methods: *supervised* and *unsupervised* learning methods. *Unsupervised* learning methods are referred to *clustering*. Hereby, the aim is to discover new classes and hidden structure since the input examples are not class labeled. Whereas *supervised* learning is a synonym for *classification*, classifying nominal class labels or *prediction*, predicting numeric class values [6]. In literature ([7]), the term *prediction* tends to be used for predicting numerical class values. The term *regression* is also synonymic used for predicting numerical class values, whereas the term *classification* is used for predicting the class label. Throughout this thesis, the term prediction will be used in its broadest sense referring to predict class values as well as class labels and only in cases of misunderstanding described in detail. As described in Section 1.2, the aim of the machine learning is to predict the non-functional characteristics of web services. Accordingly, supervised learning methods are therefore selected.

While there has been much research in the subject of single machine learning algorithms for certain field of problems, there has been less empirical research comparing all the main machine learning methods together. This is because of the variety of the machine learning methods and the different implementations within a method group. Therefore, it is a challenge to generalize the methods on certain criteria in general.

However, S. B. Kotsiantis [8] has published a review of established supervised machine learning classification techniques giving a comprehensive overview of the main machine learning methods evaluation.

	Decision Tree	**Naïve Bayes**	**Neural Networks**	**kNN**	**SVM**	**Rule Learners**
Speed **- Speed of Learning**	* * *	* * * *	**(*)**	* * * *	**(*)**	* *
Speed **- Speed of Classification**	* * * *	* * * *	* * * *	**(*)**	* * * *	* * * *
Scalability **- Incremental Learning**	* *	* * * *	* * *	* * * *	* *	**(*)**
Accuracy **- Accuracy in general**	* *	**(*)**	* * *	* *	* * * *	* *
Robustness **- Tolerance to missing values**	* * *	* * * *	**(*)**	**(*)**	* *	* *
Robustness **- Tolerance to noise**	* *	* * *	* *	**(*)**	* *	**(*)**
Overall Rating	**16**	**20**	**14**	**13**	**15**	**12**

Table 2.3: Comparison of the main machine learning methods (**** represent the best and * the worst performance) (based on [8], p. 263)

The aim of the publication is to give an orientation for a method selection with pros and cons to certain aspects, helping to find a best-fit method and not selecting inappropriate methods. Table 2.3 shows the results with corresponding star rating, the evaluation criteria on the left column and the methods on the top row.

In section 2.2, the challenges speed, accuracy, scalability and robustness were introduced. The evaluation criteria within this pre-selection are accordingly grouped to these defined challenges.

It has been revealed by consulting the supervisors, that those results are generally also reflect their assessment and experience. Overall, there is no single method performing better than all others. The comparison strongly depends on the corresponding single criterion. However, for a more general view, an overall rating as sum of each single rating has been applied, as seen in the last row. Thereby, the Naïve Bayes with an overall rating of 20 outperforms all others, and the Decision Tree follows on the second rank.

For an appropriate pre-selection, the approach of dismissing the worst methods first, was applied. As seen in Table 2.3, the worst rating within each criterion is highlighted in brackets. The methods Neural Networks, kNN and Rule learners show two or more of those worst ratings. They also show the lowest overall rating in comparison to the other methods. This more general evaluation is briefly explained with respect to the single criterion in the following part. **Neural Networks** and **kNN** are dismissed, because either of their worse speed in learning or classification and their low tolerance regarding missing values and noise. **Rule learners** are also dismissed because of their low rating in speed of training and because they are not optimal for incremental learning.

Considering the fact that there exist many implementations within a method class, as already mentioned before, the best two machine learning methods naming **Decision Tree** and **Naïve Bayes** are considered for the further evaluation. A brief explanation for this pre-selection is given in the following paragraph.

Because the prediction takes place in a performance critical setting, the speed of learning and classification plays an important role. Here both methods show a high overall rating compared to the other methods. Furthermore, it is important to have a certain ability for incremental learning. Naïve Bayes has a low rating in the criterion accuracy. As mentioned in section 2.2, the accuracy is important but speed and scalability is in general more important. It has to be mentioned that Naïve Bayes has the restriction of dealing with continuous attributes [8], whereas Decision Tree can handle both continuous and categorical attributes.

For the selected machine learning methods Decision Tree and Naïve Bayes, there exist many different implementations. Taking only the Decision Tree method there are implementation for classification, regression, incremental or batch learning, etc. A concrete example is the Hoeffding Tree, an incremental, any time decision tree, but there are also classical implementations such as the C4.5 or ID3 decision tree.

2.3.2 Library Pre-Selection

In this section first, the requirements for selecting a machine learning library with regards to the SOC setting are defined and second, corresponding libraries from search are presented. Finally, a pre-selection is applied.

The requirements for selecting the libraries were as follows:

- Integration: There should exist an easy way of integrating the library into a Java environment. The library should be capable of real-time processing within a machine learning workflow.
- Automation: The library should provide a high automation degree to be able to adapt changes in the learning environment (e.g., changing service instances, service consumer with new call context).
- Usage: There is a strong dependency between the library and the methods itself. Often libraries are created for specific purposes and only contain certain specific learning methods. Therefore, the general usage for solving real world problems should be considered as well as the overall method selection.
- Open Source: The library should be open source because of the non-commercial research project usage.

For selecting appropriate libraries, different sources were searched. Hereby, KDnuggets [9], a well-known site for Analytics, Big Data and Data Mining was a helpful resource finding the most common and proven libraries for machine learning tasks. The comprehensive list 'Libraries and Development Kits for Data Mining' was examined and corresponding libraries were searched [10]. Furthermore, also experts and other specialized sites for data mining and machine learning were reviewed for search. For example Albert Biffet, one of the main contributor of the Weka framework, put together today's most popular Big Data Mining Tools in the Open Source market in one of his blog posts [11].

In KDnuggets newest annual survey 3'000 voters answered which Analytics, Data Mining, Data Science software/tools they used in the past 12 months for real project and not just evaluation [12]. Here are the top ten results of the survey:

1) RapidMiner, 44.2% share (39.2% in 2013)
2) R, 38.5% (37.4% in 2013)
3) Excel, 25.8% (28.0% in 2013)
4) SQL, 25.3% (na in 2013)
5) Python, 19.5% (13.3% in 2013)
6) Weka, 17.0% (14.3% in 2013)
7) KNIME, 15.0% (5.9% in 2013)
8) Hadoop, 12.7% (9.3% in 2013)
9) SAS base, 10.9% (10.7% in 2013)
10) Microsoft SQL Server, 10.5% (7.0% in 2013)

This list includes all kinds of tools, software, languages as open source or commercial, etc. The top ten list shows that there are a variety of tools with specific features, which are used for different fields of problems. For example, Excel, SQL, Python and all commercial tools are no appropriate solutions for the SOC setting with respect to the previously defined criteria. The Hadoop framework used for parallel batch processing of large-data sets is also not a suitable solution for real-time processing. Furthermore, the list has to be viewed critical because the survey was publicly opened so that everyone could attend. Therefore, some vendors pushed the participation on the survey. Nevertheless, the list gives a good overview of current used Data Mining and Analytics tools within real projects.

The next part covers a selection of all examined libraries that are appropriate for the SOC setting. Hereby a short presentation of the tools is given not aiming at comprehensive listing all features, but rather to give an explanation why or why not the corresponding tool is selected.

RapidMiner

RapidMiner is an easy-to-use desktop application for solving a variety of Machine Learning, Data Mining, Text Mining and Predictive Analytics tasks. It has a comprehensive selection of machine learning methods and integrated third-party libraries [13]. As seen in the top ten list from above, RapidMiner is a well-known and widely used data mining tool. In the past, there has been a big discussion about the commercialized licensing model. However, the RapidMiner Company announced that everything that is currently free stays free. Only the newer versions will be commercial and when a new release is published the previous version gets open source [14].

Despite the popularity and established usage, RapidMiner is not selected for further evaluation. The main reason for the exclusion is the missing capability for incremental learning. No machine learning method or external library could be found for incremental learning. That is because the RapidMiner comes from the area of classical batch processing and analytics. Another point, as examined in section 2.1, is the integration into Java. It is generally practicable but the handling with processes especially for the automation takes much effort.

R project

R is an open source software environment for statistical computing and graphics. It has its own programming language and an extensive selection of functions and extensions [15].

The main reason for not selecting R is the high entry barrier. R has its own programming language, which causes an additional effort to learn. Although there are libraries for connecting R with Java such as the 'Java GUI for R' [16], but with regards to automation and integration it seemed not an appropriate solution. Furthermore, R has a statistical background, which means that modern machine learning methods and concepts especially in the area of incremental learning could not be found or are not that popular.

Weka + MOA

Weka is an open source library with a collection of machine learning algorithms for data mining tasks from the University of Waikato. The library contains different methods and algorithms for pre-processing, classification, regression, clustering, association rules and visualization. It is written in Java and can be natively integrated into Java projects [17].

MOA, which stands for Massive On-line Analysis, is also an open source framework from the University of Waikato. The framework is designed for testing and implementing online algorithms with focus on data streams. It includes several state of the art algorithms for classification, regression and clustering [18].

The Weka and the MOA libraries are selected for the evaluation because of their extensive collection of classical machine learning methods as well as new algorithms with state of the art concepts for incremental learning. The bi-directional interaction of the libraries also allows using them together. Both libraries are implemented in native Java and therefore, easy to integrate into the central broker Java framework. In addition, the automation of the machine learning process can be handled without restrictions because of the native integration. As seen in the top ten ranking from KDnuggets, Weka is a well-known and an established Data Mining tool with a widespread usage for solving real-world problems. MOA is quite new in the area of

Data Mining tools but already found recognition in the expert groups of data stream mining. Overall, they seem to be a perfect combination handling the given machine learning challenges within the SOC setting.

Apache Mahout
Apache Mahout is a suite of machine learning libraries with algorithms for clustering, classification and collaborative filtering on top of scalable and distributed systems. It became a top level project of the Apache foundation in 2010 [19].

Despite the overall advantages, it was not selected for the further evaluation because of the little selection of machine learning methods and the specific use cases. Mahout focuses on scenarios in the area of text mining and is generally used in combination with the complex Hadoop framework.

Other libraries
Libraries, which were reviewed but not further considered with regards to the defined criteria are: Apache Spark, KNIME, Shogun, Shark, scikit-learn, Vowpal Wabbit.

2.3.3 Proof of Concept

Prior to the final pre-selection a proof of concept was applied verifying the potential of the machine learning methods and proofing that the libraries can be integrated and used with a manageable effort.

As presented in the section above, the selected Weka and MOA library are used within this proof of concept. Furthermore, also RapidMiner is used for comparing the current implementation with the new selected machine learning algorithms. Hereby, implementations for Naïve Bayes and Decision Tree, in MOA the Hoeffding Tree and in RapidMiner the C4.5 were selected. The C4.5 Decision Tree classifier within RapidMiner is used in the current implementation of the central broker framework. The methods in the MOA library are based on incremental learning, whereas in RapidMiner the C4.5 decision tree is based on a classical batch learning. The benchmarks were executed within the integrated development environment (IDE) Eclipse [20]. Therefore, the two libraries had to be integrated and additionally RapidMiner processes had to be created.

For training and testing the methods, three different data sets were used: Airlines (540K records - The airlines data set is used for evaluating regression models [12]), Response time[3] (104K records - this data set was used from the current feedback tickets of the SOC framework) and Availability (14K records). For the evaluation, the holdout approach was used with stratified sampling to have equally distributed training and test data sets. Table 2.4 shows the results obtained from the methods evaluation. The results do not include the computation time for data pre-processing. Each benchmark was executed ten times and the average was taken.

The results show an overall improvement of the new machine learning algorithms (MOA Hoeffding Tree and MOA Naïve Bayes) in respect to time of training, testing and the overall accuracy. The Naïve Bayes classifier performs best regarding the time of training and testing. At the same time, the Naïve Bayes has either the best or almost the best accuracy.

[3] This data set was used from the current feedback tickets of the SOC framework.

ML Method	Training (ms)	Testing (ms)	Accuracy (%)
Airlines (Training: 270K, Testing: 270K)			
Hoeffding Tree (MOA)	3049	836	64.53
Naïve Bayes (MOA)	709	1044	63.22
C4.5 (RM)	1448	775	55.52
Airlines (Training: 512K, Testing: 27K)			
Hoeffding Tree (MOA)	4677	110	65.49
Naïve Bayes (MOA)	1080	96	63.66
C4.5 (RM)	2420	104	56.03
SOC Response Time (Training: 52K, Testing: 52K)			
Hoeffding Tree (MOA)	255	138	14.82
Naïve Bayes (MOA)	91	164	15.26
C4.5 (RM)	102	284	4.81
SOC Availability (Training: 7K, Testing: 7K)			
Hoeffding Tree (MOA)	21	11	91.18
Naïve Bayes (MOA)	10	11	91.18
C4.5 (RM-Cross-Valid.)	247	50	91.18

Table 2.4: Evaluation results of the pre-selected machine learning methods and libraries

These benchmarks demonstrate that on the one hand there is a significant performance improvement of the new machine learning methods. On the other hand, it shows the low entry level of using and integrating these libraries. Further has to be mentioned that the effort within RapidMiner is much higher, because the processes have to be modeled within the RapidMiner user interface and afterwards integrated into Java.

3 Design

This chapter introduces the conceptual design for an optimized web service selection. Thereby the overall machine learning activities from the data pre-processing, the performance prediction, the service optimization, up to the recommendation of a best-fit service are illustrated and described in detail.

3.1 Architecture

Figure 3.1 outlines the architecture for an optimized web service selection. In comparison to the process of a dynamic web service binding, described in the introductory chapter, this conceptual design moreover focuses on illustrating the activities of the adaptive machine learning.

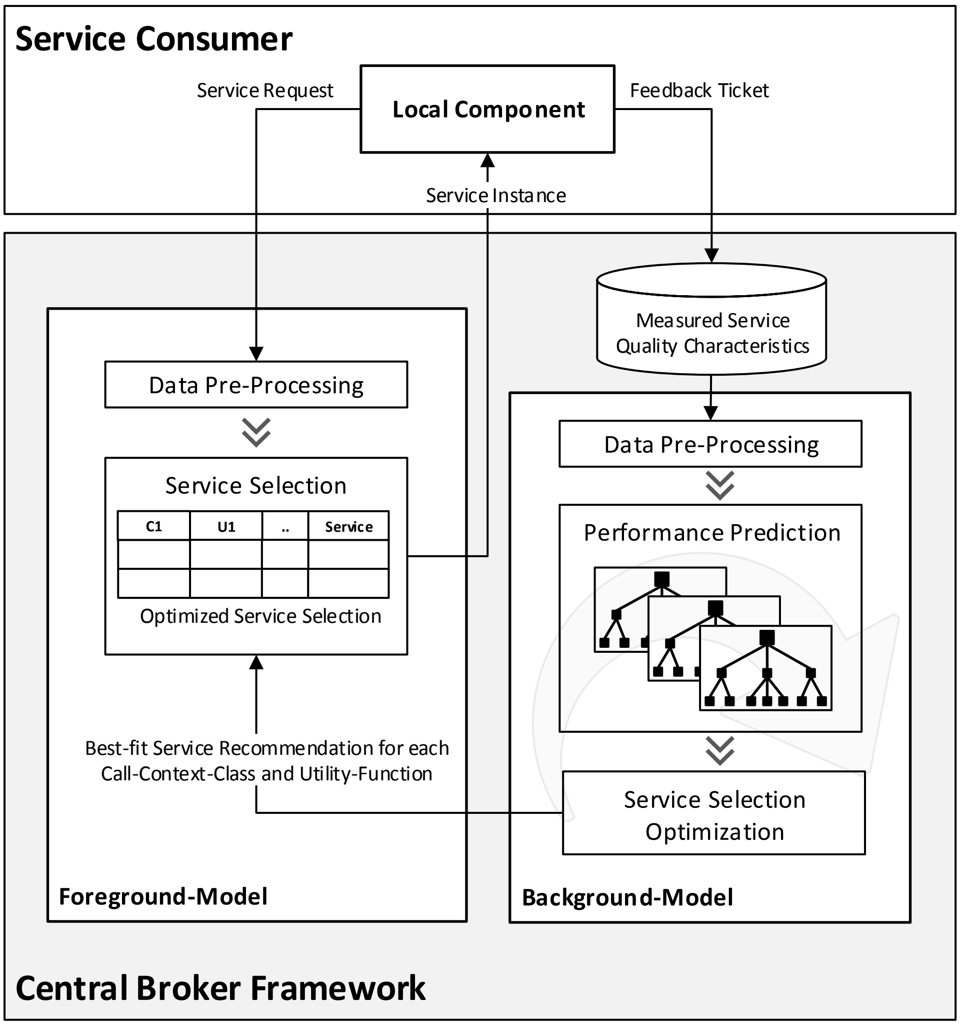


Figure 3.1: Architecture of the central broker framework aiming at optimizing the service selection for a unique service consumer

The involved participant on the one side is the service consumer, represented by the local component, and on the other side the central broker framework with the so-called foreground-/background-model. Those participants interact with each other via several interfaces illustrated as arrows. The next part describes the general procedure of optimizing the service selection.

The interaction of the service broker and the consumer is triggered when a best-fit web service is requested. The request of the consumer is received by the foreground-model and pre-processed. According to the call context attributes and the consumer preferences, the best-fit service instance is selected and passed to the service consumer. After a service consumption, the local component submits the measured performances and call context attributes of the service call to the central broker, where it is persisted. The background-model has an asynchronous, dynamic and repetitive characteristic. When new feedback data is submitted, the data is pre-processed for the prediction and optimization activities. These are controlled and executed based on certain strategic rules within a repetitive cycle. When the performance prediction for certain call contexts is triggered, all current performances of the possible service instance selection are predicted and passed to the service selection optimization. Within the optimization process, the performances are normalized and thereafter the utility is calculated. Finally, the best-fit service for the specific call contexts is determined and submitted to the foreground-model, where the service recommendation is persisted.

The design is derived from the defined goal criteria (1.4), the problem definition (1.2), the challenges given from the machine learning (2.1) and the restrictions of the current implementation (2.2). The core feature of this introduced architecture is the separation of the service optimization, including learning, predicting and calculation activities, from the performance-critical request of a best-fit web service. This leads to the two-part central service broker with the foreground-/background-model. This asynchronous model enables that the time-consuming prediction and optimization take place in the background and does not affect the real-time request for a best-fit web service. Thereby the core approach is the pre-calculation of best-fit web services for certain unique call context classes. Any kind of performance issues within the critical path from requesting a best-fit web service to the point of the service recommendation were thereby eliminated. In the subsequent sections, the design of the single activities within the foreground-/background-model are further described.

3.2 Foreground-Model

The foreground-model, compared to the background-model, is less complex and only exists of the two process steps: pre-processing of the requested call context and best-fit web service selection.

3.2.1 Pre-Processing

Main objective of the pre-processing is the determination of the call context class given from the information of the service request.

Call Context Class

Having the overall foreground-/background-model, a concept was required for identifying a consumer with corresponding call context attributes and their unique preferences. This was not only needed for controlling the continuous learning in a systematic way, but also for a high-performance access of the service recommendations.

A call context classes is a unique combination of all call context attributes, functional services and utility functions. Table 3.1 shows an example of several call context classes with corresponding attributes.

id	external ip	weekday	daytime	service	utility function
34	DE	Sat	11	ServiceA	0.5*responsetime+0.5*availability
35	DE	Sat	12	ServiceA	0.8*responsetime+0.2*availability
36	US	Mon	13	ServiceA	0.5*responsetime+0.5*availability
37	US	Mon	13	ServiceB	0.5*responsetime+0.5*availability

Table 3.1: Example of call context classes with corresponding attributes

When a service consumer requests a best-fit service the IP, the timestamp of the request and corresponding utility function is submitted to the central service broker. The IP is mapped to a location and information of the timestamp such as the weekday and daytime hour are extracted. Section 4.2 describes the specifics of the pre-processing implementation. After the pre-processing, the call context class id is determined and the next process step, the selection of the best-fit service selection, is applied.

3.2.2 Service Selection

Within the time-critical request for a best-fit service, the performance of accessing and receiving a best-fit service plays an important role. Therefore, the simple approach of a lookup table was applied. The recommendation table contains the current best-fit services for a specific call context class. As seen in Figure 4.3, the table contains two service recommendations for the call context class *34* (the value of the attribute call context class is a foreign key to the id of the call context class table, as seen in Figure 3.2). In the third column, the best-fit service is listed with the functional service identifier. In this example, *ServiceB* replaced *ServiceA* on the *22.02.2014* at *13:34:46*. The attribute instances contains the number of learned feedback tickets until a service change for the specific call context class was applied. Here the service change took place after 16 new feedback tickets were learned. The attribute version is an index counting the number of service changes for the corresponding call context class. The highest value of the version attribute represents the current best-fit service.

id	call context class	service	date	instances	version
210	34	ServiceA	2014-02-22 11:02:12	10	7
211	34	ServiceB	2014-02-22 11:34:46	16	8

Table 3.2: Service recommendation table for determining best-fit services

The advantage of the presented technical selection approach on the one hand, is the simple and quick determination of a best-fit service for a specific call context class. On the other hand, the attributes date, instances and version allow an extensive insight and traceability into the behavior of the central service broker. For example, when the service recommendation for certain call context classes are changing often, this could be either an indicator for a close competition or a significant continuous service quality change of the involved services. It is also possible to make statements, which services for certain call contexts and consumer preferences are best.

In addition, this approach could be extended by also persisting the predicted performances of the quality attributes. Then the performance effect should be further analyzed and decided, if a benefit is a given.

3.3 Background-Model

The background-model is the core of the optimized web service selection and synonym for the learning, prediction and optimization activities. The overall component is designed as a continuous and repeatable process.

3.3.1 Pre-Processing

As seen in Figure 3.1, the measured service quality characteristics submitted from the local component are input for the pre-processing activities within the background-model. The input consists of two attribute types: the call context attributes and the measured quality attributes.

Within the machine learning methods for classification, the class labels can only be handled as nominal values (e.g., Naïve Bayes or classical Decision Tree). Therefore, the values of continuous attributes have to be transformed into discrete class values. This process is called discretization and a general notion in the area of data mining [6]. For example, the numeric response time value of 739 is discretized into the nominal class '700-750'. There exist different approaches for the discretization, which are discussed in the implementation part in subsection 4.2.2.

3.3.2 Strategically Learning

The idea behind the foreground-/background-model was to detach the real-time request for a best-fit web service selection from the overall time-critical prediction activities and the service determination. The determination of the best-fit services automatically and continuously takes place in the background. The pre-determined best-fit services therefore can be quickly provided. This overall model required a mechanism for controlling the continuous process of learning and predicting the performances, calculating the utilities and determining the best-fit services. Thereby, the term *strategically learning* was introduced, which not only is addressing the learning activity, moreover can be referred to this overall continuous optimization process.

As control mechanism, the approach of strategically learning with respect to defined thresholds was created. A threshold is a value level that when exceeded, triggers an action. In general, two main threshold types were defined: count and time thresholds. For example, when defining a count threshold, the background-model learns and counts the number of new feedback input within a call context class. After the defined threshold number is exceeded, the count is reset and the process for optimizing the service selection and determining the current best-fit service is triggered. The time threshold works similar, but instead of counting the new feedback tickets, the minutes after the last service optimization are counted. In the following, all threshold types are listed. Within a call context class the performance prediction and service optimization is triggered:

- *Count* - when a defined number of instances is exceeded.
- *Time* - when a defined number of minutes is exceeded.
- *CountANDTime* - when both, a defined number of instances and minutes are exceeded.

- *CountORTime* - when either a defined number of instances or minutes is exceeded.

Figure 3.2 illustrates the three main activities within the background-model. When a defined threshold is exceeded, first the performances of the quality attributes are predicted (step 1). Thereafter, the performances are normalized into a certain value range and corresponding utilities are calculated (step 2). Finally, the service with the highest utility is determined and selected as best-fit service (step 3). All these activities are part of the strategically learning process and core of the optimized service selection.

The activities of the performance prediction are described in the next subsection, whereas the other activities of step two and three are further described in Section 3.3.4.

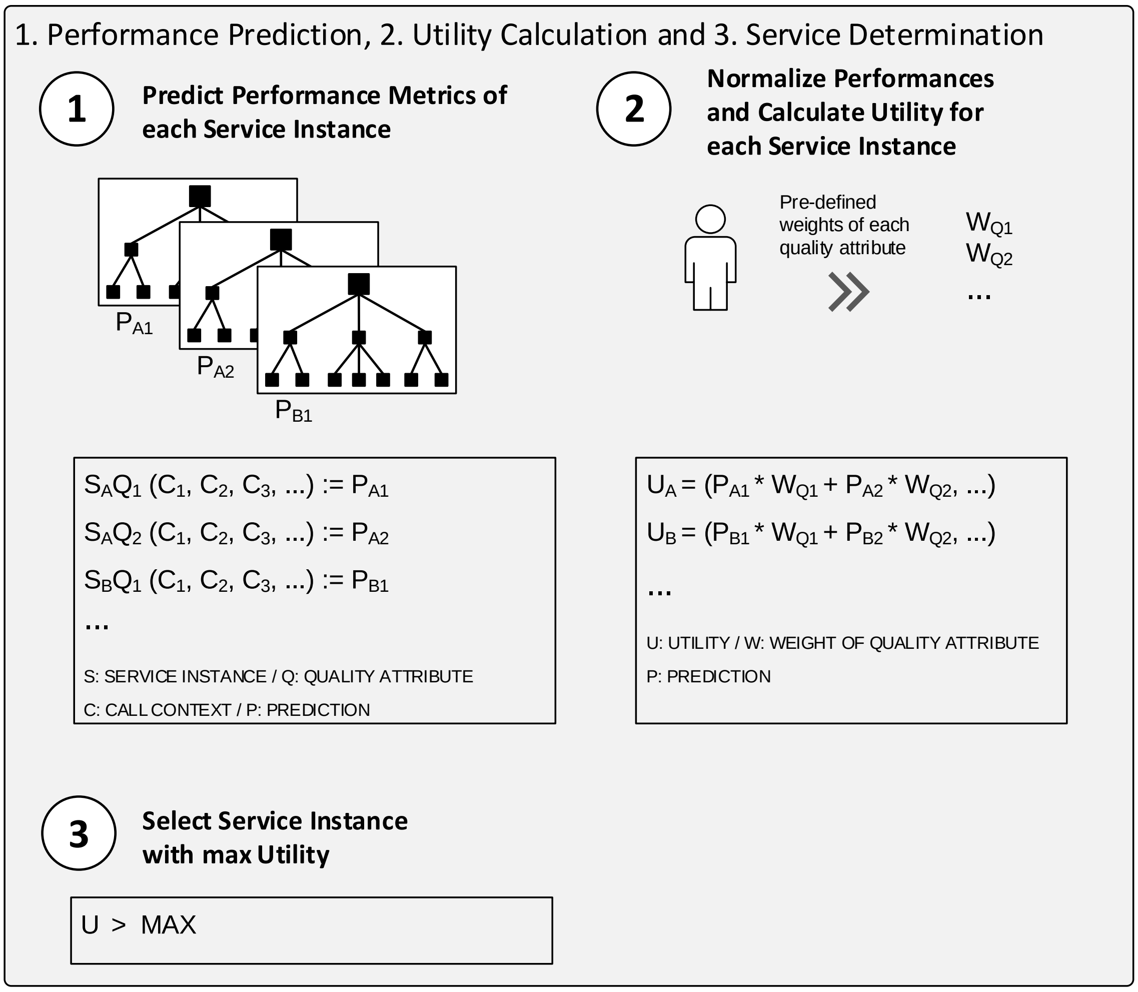

1. Performance Prediction, 2. Utility Calculation and 3. Service Determination
1
Predict Performance Metrics of each Service Instance
2
Normalize Performances and Calculate Utility for each Service Instance
P_A1
P_A2
P_B1
Pre-defined weights of each quality attribute
W_Q1
W_Q2
...
S_AQ_1 (C_1, C_2, C_3, ...) := P_A1
S_AQ_2 (C_1, C_2, C_3, ...) := P_A2
S_BQ_1 (C_1, C_2, C_3, ...) := P_B1
...
S: SERVICE INSTANCE / Q: QUALITY ATTRIBUTE
C: CALL CONTEXT / P: PREDICTION
U_A = (P_A1 * W_Q1 + P_A2 * W_Q2, ...)
U_B = (P_B1 * W_Q1 + P_B2 * W_Q2, ...)
...
U: UTILITY / W: WEIGHT OF QUALITY ATTRIBUTE
P: PREDICTION
3
Select Service Instance with max Utility
U > MAX

Figure 3.2: Procedure of a best-fit service selection within the background-model and corresponding core activities (1. performance prediction, 2. utility calculation and 3. service determination)

3.3.3 Performance Prediction

The performance prediction is the overall term for the prediction as well as the prior learning. After the submitted feedback tickets are pre-processed, they are learned within the prediction models and underlying machine learning methods. For each service instance and each quality attribute (seen as S_AQ_1, S_AQ_2, etc. in Figure 3.2, step 1) a single prediction model is used (seen as P_{A1}, P_{A2}, etc. in Figure 3.2, step 1).

As described before, when a defined threshold within a call context class is exceeded, the prediction process is triggered (step 1 in Figure 3.2). Then all performances of each service instance within a functional service group are predicted. For example, all current performances of the functional equally services *A1*, *A2* and *A3* are predicted after ten new feedback tickets from service *A1* and call context class '*DE, Mon, 14*' were submitted.

The input of the prediction are the call context attributes such as the call time (timestamp) and call location (consumers IP) of the measured past service calls. The label attributes, which are to be predicted, are the quality attributes. A quality attribute stands for the non-functional service characteristics. In the research paper of Kirchner, et al. [1] as well as within the thesis work of Markus Geisler [5] and Catherine Catherine [21], several non-functional characteristics were introduced within the SOC framework such as response time, throughput, availability, monetary costs, reliability and reputation. The attributes generally have different characteristics and are either dynamic evolving or static attributes. They can be assessed automatically or manually within a human interaction. This work focuses on the two main attributes namely **response time** and **availability**. These performance attributes can be assessed automatically and show a dynamic behavior. The following part further describes those two attributes and outlines how they are determined.

Response Time

Within the SOC framework, the response time is a performance metric and expressed as the duration of a service call from end consumer perspective. The duration includes the time starting from the client's request, to the processing within the service instance, until the receiving of the response on client side. For predicting this numeric quality attribute, the pre-selected machine learning algorithms are used. The classification models can only handle nominal values, which is why the response time has to be discretized. Others such as the regression models can handle continuous values.

Availability

In this work the availability, when referring to a service call, is expressed with the Boolean values 'true' and 'false'. If the service instance was available at the time of the request and successfully responded, the availability is 'true'. When the service did not successfully answer, then the availability is 'false'. The availability attribute is not handled within the pre-selected machine learning methods. This is because predicting the availability as a nominal value attribute with 'true' or 'false' is not accurate enough. Instead of using a prediction model, the statistical approach of building the probability that a service will be available is chosen. Therefore, within a call context class the relation between the number of available and total service calls is built from the observed past service calls. For example, the call context class for a service instance contains an overall of 74 feedback tickets with a total of two non-available service calls. In this example, the availability would be 97.3% (72/74).

3.3.4 Service Selection Optimization

As seen from Figure 3.1, the optimization of the service selection exists of step two, the utility calculation, and step three, the service determination.

Utility Calculation

The concept of calculating the service utilities was already briefly described in the introductory chapter. The selection of multiple services offering the same functionality but only distinguishing themselves between their service quality characteristics results in a multi criteria decision making problem. Within the research of Kirchner, et al. [1] the selection problem is approached by defining a utility function as an optimization goal of the unique consumer. The utility function follows the scoring method of simple additive weighting (SAW) [22]. For each alternative thereby an evaluation score is calculated, the so-called utility. The utility is determined by numerically combining the non-functional characteristics (quality attributes) with the pre-defined weights of the unique consumer (the summed weights have to be one). Before the calculation, the predicted quality attributes, which are from different domains, have to be converted into a similar value range to be comparable. This process is called normalization or standardization approaching to give all attributes an equal weight [6]. For the response time low values, which are desired, should show a high benefit, whereas for the attribute availability high values are desired and therefore should show a high benefit.

Service Determination

The core approach of the foreground-/background-model is to have all current best-fit services listed in the foreground lookup table. Therefore, a continuous adjustment takes place to keep the recommended best-fit services up to date.

The service determination aims at maximizing the utilities. Thereby simply that particular service, which has the highest utility, is selected. When the optimization process for a certain call context class is triggered and the best-fit service is determined, a matching with the recommendation table of the foreground-model is applied. If the best-fit service, listed in the recommendation table, differs from the determined and just predicted best-fit service, a new service recommendation is inserted into the table.

Summary

The architecture for an optimized web service selection is designed as a continual machine learning process aiming to address the current restrictions and challenges within the SOC setting. The concept eliminates the current performance problematic, so that requesting a best-fit web service selection is not time-critical anymore.

4 Implementation

This chapter concerns the specifics of the central broker implementation. First, the derived requirements for creating appropriate test data and the general simulation approach are introduced. Thereafter, the consolidation and discretization activities within the pre-processing are described. Finally, the machine learning specifics for the optimized web service selection are outlined.

4.1 Test Data

Having meaningful and sufficient test data plays an important role when evaluating machine learning methods. If the data input of the learning model is bad, the output and therefore the prediction does not get better. Markus Geisler [5] and Catherine Catherine [21] also examined the challenge of appropriate test data in their thesis work within the SOC research project. Both implemented concepts and techniques for creating test data. As described in 2.1, the current available test data is insufficient in respect to data volume. Furthermore, the data does shows neither traceable web service characteristics nor performance changes over time, which all is necessary for a representative evaluation. Due to this fact, an initial objective of this work was to create test data fitting the needs for the evaluation. In the next subsections, the requirements for appropriate test data are described and the approach for simulating the test data is presented.

4.1.1 Requirements

In the SOC setting, the data used for testing the machine learning methods are the submitted feedback tickets containing the call context and the performance measures from each service consumption. The requirements for creating this data were defined in the categories data volume and data characteristic, which are further described in the subsequent part.

<u>Data characteristic</u>
Real-world web services can have cyclic performance behaviors such as day and night cycles or weekly and monthly cycles, etc. They also can show workweek and weekend behaviors. Other non-cyclic behavior can be unexpected hardware failures resulting in a service breakdown [5]. Geographical network latency is another significant and common impact on the performance characteristic of web services. More general service behaviors are arbitrary performance volatilities or performance outliers.

All these mentioned examples for web service behaviors should be reflected in the test data characteristics. For summarizing the above, the following requirements are given for simulating and creating appropriate test data:

- Cyclic performance behavior
- Performance change over time
- Geographical network latency
- Performance volatility and outliers

<u>Data volume</u>
As discussed in section 2.2.3 with increasing number of participants, the number of data records can rises into millions. Therefore, the scalability and the handling of a high data volume have to be evaluated. Especially the incremental learning algorithms need an appropriate data volume to outperform the classical batch

algorithms. According to the vendor of the MOA library, several millions of instances are needed to show better performance than classical methods [23]. The proof of concept (see Section 2.3.3) indicated that already several hundred thousands of instances resulting in a better performance of the incremental algorithms. Furthermore, considering weekly cycle behaviors, an appropriate time period should be chosen. To summarize, the defined requirements are as follows:

- Number of records: several 100K records
- Time period: at least three weeks

4.1.2 Simulation

The first idea for creating test data was to design several dummy web services with a certain logic to imitate real-world web services. They would being called from different geographical located clients with a certain time behavior and call frequency. The same approach was applied for creating the current available test data of the previous thesis work. As it turned out the implementation, deployment and scheduling of the services would be complex and difficult to handle. Furthermore, they had to run in real-time, which would be very time-consuming since there is the requirement for a high data volume. The missing flexibility to have several test runs and adapt the defined service logic would not be given.

Instead of imitating real-world web services with dummy services, the approach of simulating the web service quality characteristics was considered. The first implementation of a simulation tool showed an easy and comprehensive approach for generating appropriate test data. According to the previously described data characteristics requirements, corresponding simulation parameters had been defined. The subsequent part describes those defined simulation parameters.

Table 4.1 lists all parameters for the test data generation with corresponding attributes. As described in 3.3.3, the non-functional characteristics response time and availability were selected for simulating the quality characteristics.

Parameter	Attributes
Call Context	
Call Time	• Simulation start (DD-MM-YYYY HH:MM:SS) `[start]` • Service call frequency (ms) `[periode]` • Duration of Days `[days]` • Duration of Hours `[hours]`
Consumer	• Consumer ID `[consumerId]` • Consumer IP address `[externalIp]`
Service Instance	• Service instance name `[serverName]` • Service instance IP `[calledServiceInstanceAddress]`
Service	• Service name `[functionalServiceIdentifier]`
Utility	• Consumers utility function `[utility]`
Response time	
Response time	• Response time normal (ms) `[respNormal]` • Response time volatility (upward) (ms) `[respDeviation]` • Response time simulation (ms) `[respSimulation]`
Shift	• Begin (% of the overall records) `[respOverallBeginPerc]` • End (% of the overall records) `[respOverallEndPerc]`

Daytime	• Begin (HH-MM) `[respDaytimeBegin]` • End (HH-MM) `[respDaytimeEnd]`
Weekday	• Weekday name(s) (EEE) `[respWeekday]`
Outlier	• Response time outlier (ms) `[respOutlierResponse]` • Response time deviation (ms) `[respOutlierDev]` • Rate (% of the overall records) `[respOutlierPerc]`
Availability	
Availability	• Non-availability (% of the overall records) `[nonavailOverallPerc]` • Begin (% of the overall records) `[nonavailOverallBeginPerc]` • End (% of the overall records) `[nonavailOverallEndPerc]`
Daytime	• Begin (HH-MM) `[nonavailDaytimeBegin]` • End (HH-MM) `[nonavailDaytimeEnd]`
Weekday	• Weekday name(s) (EEE) `[nonavailWeekday]`
Outlier	• Rate (% of the overall records) `[nonavailOutlierPerc]`

Table 4.1: List of simulation parameter for simulating non-functional service characteristics

In general, there are three categories of parameters: call context, response time and availability. The call context parameters allow scheduling the time and frequency of the service call, but also allow defining the consumer and the provider details. The response time and availability parameters are for the simulation of the service quality characteristics. Within the response time simulation, the expected *normal response time* has to be defined with an additional upward *deviation*, simulating the service volatility. The attribute *response time simulation* is used for simulating a *response time shift* or a *daytime* and *weekday* service behavior. Having a response time shift simulation, the *start* and *end* percentage referring to the overall records have to be defined. For the daytime simulation, the *begin* and *end time* and for the weekday, the *weekday names* have to be set. Furthermore, response time outliers can be simulated by defining the *response time outlier* attribute with a *deviation* and percentage *rate* of the overall records. Within the simulation of the service availability, similar attributes are used as described before. Instead of numerical values, nominal attribute values with 'true' for availability and 'false' for non-availability are simulated.

The simulation parameters are defined via a configuration CSV file (as seen in the table, the naming of the CSV parameters are in square brackets). Beside the listed attributes from Table 4.1, the configuration file contains additional attributes, needed for selecting only those simulation attributes, which are desired. For example, there are Boolean flag parameters ('true' and 'false') for selecting or deselecting the response time shift simulation and another Boolean parameter for combining the daytime and weekday simulation. Within a single simulation run, multiple simulation cycles can be executed at the same time. Each row in the configuration file corresponds to one simulation cycle. After a simulation run the created data can be either exported to a CSV file or directly submitted to the central broker framework via the feedback SOAP interface (SOAP is a protocol intended for exchanging structured information [24]).

The following part outlines the advantages of the test data generation tool. The main benefit is the general usage with the ability executing and repeating the simulation runs easily. This can be very favorable especially in the beginning, when defining service behaviors and trying to determine an appropriate test data composition. The creation of the data is completed within seconds, which compared to measuring real-world web services would take days or weeks. The generation approach allows simulating any specified time period with no limits in respect to data volume. The extensive selection of parameters allows creating sophisticated test data scenarios in any degree of complexity under stable and predictable conditions.

Figure 4.1 shows the definition of a simulation cycle with a short description on how the data is created. In this simulation cycle the consumer with the discretized IP *DEConsumer*, consumer id *1* and the utility *0.5*responsetime+0.5*availability* requests the service *Scenario1* with the address *DE1Service* starting at *22.02.2014* for *30* days every minute (call frequency of *3.600.000* milliseconds).

The service has an expected response time between *250* and *270* milliseconds (plus deviation), which is decreasing between *360* and *380* milliseconds in the time course (*50*% to *100*%) from day *15* to day *30*. Furthermore, the service performance is decreasing weekly, each *Friday* from *15.00* to *17.00* pm between *360* and *380* milliseconds. *Two* percent of all service calls are outlier with a response time between *600* and *620* milliseconds.

The service has a *95*% availability (*5*% of all service calls are non-available) from day *27* to *30* (*90*%-*100*%). Additionally every morning from *08.05* to *08.30* am the service is non-available. *Four* percent of all service calls are outliers (non-available service call).

Call Context
start=22-02-2014 00:00:00, periode=3.600.000, days=30, hours=0, consumerId=1, externalIp=DEConsumer, calledServiceInstanceAddress=DE1Service, functionalServiceIdentifier=Scenario1, utilityFunction=0.5*responsetime+0.5*availability

Response Time
respNormal=250, respSimulation=360, respDeviation=20, isRespOverall=TRUE, respOverallBeginPerc=50, respOverallEndPerc=100, isRespDaytimeAndWeekday=TRUE, isRespDaytime=TRUE, respDaytimeBegin=15-00, respDaytimeEnd=17-00, isRespWeekday=TRUE, respWeekday=Fri, isRespOutlier=TRUE, respOutlierResponse=600, respOutlierPerc=2, respOutlierDev=30

Availability
isNonavailOverall=TRUE, nonavailOverallPerc=5, nonavailOverallBeginPerc=90, nonavailOverallEndPerc=100, isNonavailDaytimeAndWeekday=FALSE, isNonavailDaytime=TRUE, nonavailDaytimeBegin=08-05, nonavailDaytimeEnd=08-30, isNonavailWeekday=FALSE, nonavailWeekday=, isNonavailOutlier=TRUE, nonavailOutlierPerc=4

Figure 4.1: Example of a simulation cycle within the test data generation tool

In the following a short description of the outlier function, which is applied both for the response time and availability simulation in the same way. First, the number of outliers is calculated from the total calls and the outlier percentage. Thereafter, the outlier calls are split and randomly assigned to normal distributed ranges. From the example above, the outlier creation for response time is illustrated in detail: In total, there are *43.200* service calls (60min * 24hours * 30days) and an outlier percentage of *2*% resulting in *864* outliers. The outliers are now distributed randomly within the ranges 1-50, 51-100, 101-150, 151-200, etc. (range of 50: 43.200 / 864), which could result to a outlier distribution of 13, 52, 122, 196, etc.

The outlier value for response time is defined separately, whereas for availability the value ‘false’ is set.

4.2 Data Pre-Processing

This section outlines the implementation of the consolidation and discretization activities within the data pre-processing.

4.2.1 Consolidation

The submitted tickets via the feedback service interface are processed and stored in a very generic way. This is because within the SOC setting new attributes and measurements types can quickly change. The feedback ticket itself with all call-context attributes and performance measurements are saved into four different tables. [5]

Having the data available for the learning models, different approaches have been examined and tested. In the beginning, a classic SQL join was used for merging the data. Afterwards the more convenient PostgreSQL crosstab function, producing a pivot-table, was applied for achieving a consolidation. After all, fetching the records with a complex query caused too much overhead. Also in respect to the incremental learning and the high performance requirements, it appeared not to be an appropriate solution.

Instead, the approach of merging the data into a single table was chosen. Hereby for each of the four tables triggers were designed and implemented, fetching the inserted data and copying those into one table. All required data is located in a single table and therefore can be fetched very performant. When it comes to incremental learning, where for each record a single query is executed, this seemed the most appropriate solution. Furthermore, there could not be observed any significant performance issues caused from the additional triggers. The time differences, when submitting several 100 thousands tickets via the feedback service interface with and without the triggers, did only distinguish in range less than 100 milliseconds.

However, this approach also involves a drawback. The flexible and generic way of saving the submitted feedback data is restricted. If the framework is confronted with new attributes such as new call-context attributes or new performance measures, the triggers always have to be adapted.

4.2.2 Discretization

Table 4.2 shows the discretization activities for the corresponding call contexts and quality attributes. The different pre-processing approaches are further described in the subsequent part.

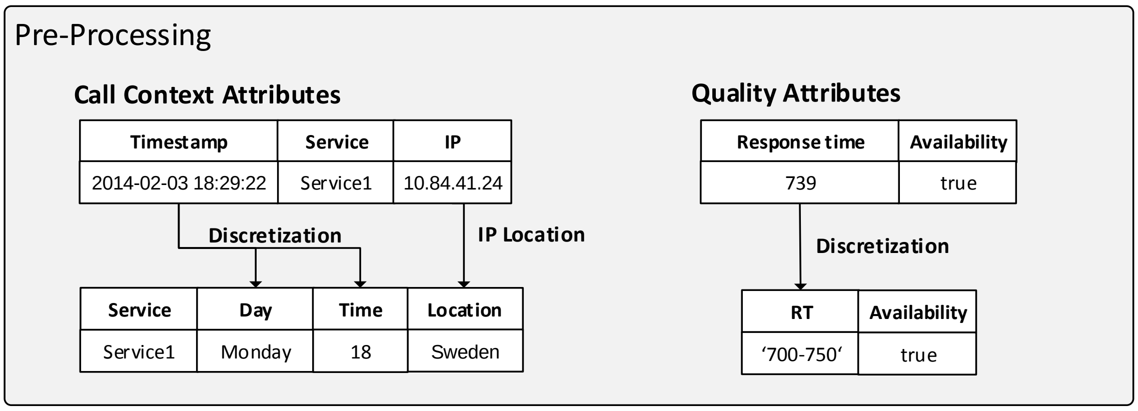


Figure 4.2: Pre-processing activities within the foreground-/background-model

<u>Timestamp and IP</u>
The timestamp of the service call, as continuous time value, is discretized and split into the attributes day and time. For the time attribute, the interval of one hour was selected. Furthermore, the IP is assigned to a location. In the current implementation, and the generation of the test data, the IP attribute already is discretized manually. In further implementation, corresponding services for mapping IPs to locations could be used [26].

<u>Response Time</u>
As already outlined in the design chapter, for classification models such as the Naïve Bayes or the classical Decision Tree learner, continuous values have to be discretized. Within the development phase, it has been shown that there are different discretization approaches, especially in the area of batch pre-processing. Discretizing numeric values on the fly within a continuous process, where data is arriving continuously, is a particular challenge. Here an overall value distribution is only known from the current data, but moreover can significantly change in the course of time.

Within the service broker, two discretization types were implemented: batch and incremental discretization. These are further described in the following:

- Batch Discretization: For this type, the method *equal-frequency* was applied with a number of 10 bins. The bin intervals were set to contain an equal number of instances. This method showed consistent better performance than the method *equal-width*, where the intervals are set to have an equal-sized interval width [25].
- Incremental Discretization: For an incremental discretization, or synonymous called online discretization, the simple approach of setting static intervals was applied. Within the central broker, the interval size of 50 milliseconds was selected. There exist only a few methods for discretizing continuous data streams, because the research in this field of online discretization just developed in the past few years. Christophe Salperwyck introduced an extension for the MOA library discretizing numeric values within a continuous stream [29]. These methods include a dynamic adaption of the interval sizes when the value distribution changes during the course of time.

Overall, the implementation of the pre-processing showed that consolidation and discretization activities are required before the training of the prediction models can take place. There exist different approaches for discretizing numeric values. The discretization within a continuous data stream thereby is a particular challenge. Furthermore, determining the right granularity and finding appropriate discretization intervals is difficult. The intervals can be selected either too broad, then information gets lost and the underlying data gets more insignificant, or the intervals are too precise, which can causes an overhead in computation but also in prediction accuracy.

4.3 Machine Learning

This section concerns the implementation specifics of the machine learning for an optimized web service selection. First, the implemented machine learning algorithms were introduced. Second, the specifics of the continuous machine learning are outlined. Third, the approach of the utility calculations is introduced. Finally, the selection of the best-fit service with the activities of the foreground-/background-model is further described and illustrated.

4.3.1 Naïve Bayes and Decision Tree

As outlined in Section 2.3.1, the Decision Tree and Naïve Bayes machine learning methods were pre-selected. Thereby the main implementations of these two method groups within the MOA and Weka libraries were selected for the further evaluation (see Section 5.2 and Section 5.3). The Naïve Bayes classifier is included in both libraries. The Decision Tree machine learning algorithms can be further distinguished between classification and regression trees. The differences, as described in section 2.3.1, are that within a classification a class label is predicted, whereas within a regression a numeric class value is predicted. There are implementations for classification and regression trees in both libraries. The subsequent part briefly describes all implemented algorithms with references to the vendor's information.

<ins>Naïve Bayes</ins>
The Naïve Bayes performs the classic bayesian prediction and is a classifier algorithm known for its simplicity and low computational cost [27]. The Weka and MOA library contain the standard implementation of the Naïve Bayes classifier but only distinguishing themselves regarding the type of learning. Within Weka the Naïve Bayes is implemented as batch classification model, whereas within MOA as incremental classification model.

<ins>J48</ins>
The J48 algorithm is the Weka implementation of the C4.5 classical decision tree algorithm [28]. This algorithm corresponds to the machine learning method used in the previous implementation of the central service broker [5]. The decision tree algorithm is implemented as batch classification model within Weka.

<ins>Hoeffding Tree</ins>
The Hoeffding Tree is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams [27]. The prediction strategy used within the leaf is a classical Naïve Bayes. The Hoeffding Tree is implemented as incremental classification model within MOA.

<ins>FIMT-DD</ins>
The FIMT-DD stands for 'Fast Incremental Model Tree for drift detection'. This algorithm is an efficient and incremental decision tree for regression with the ability

to handle high-speed and time-changing data streams [30]. This algorithm is implemented as incremental regression model within the MOA library.

<u>M5P</u>
The M5P is a regression tree, which implements routines for generating M5 Model trees and rules. It was original invented by R. Quinlan and introduced in 1992 [31]. This algorithm is implemented as batch classification model within Weka.

4.3.2 Continuous Learning

After implementing the first workflow for an optimized web service selection, it proved that Weka has a much better data handling, whereas the MOA methods are more scalable. Because of their bi-directional connection, the decision was made to have both libraries implemented, but Weka mainly for the data management.

Beside the advantages, there are also some restrictions with respect to incremental learning. Although all algorithms within the MOA library are designed for incremental learning, but incremental loading of PostgreSQL is not yet fully supported in the newest developer version of Weka (3.7.11). As seen from the commits, the subject is paid much attention, so that it is likely to be supported in the near future. As workaround, instead of using the Weka database loading, a manual incremental loading approach could be implemented.

Within batch loading, all the data gets loaded into the main memory, whereas within incremental loading a single instance is loaded at a time as long as there are new instances. Because Weka does not support PostgreSQL for incremental loading at this time, a pseudo incremental method is used. *'In pseudo incremental mode the instances are read into main memory all at once and then incrementally provided to the user.'* [25] The learning happens to be incremental but the loading correspond to be batch. Given this restriction, the possible maximum data volume used for training the machine learning methods can always only be as large as the size of the main memory.

4.3.3 Utility Calculation

As described in Section 3.3.4, the utility functions are quality preferences defined by the unique consumers. For calculating the utility first, the predicted performances are normalized and second corresponding to the simple additive weighting, the utility gets determined.

<u>Normalization</u>
In the previous implementation the attributes were normalized within a range of 0 to 10, taking the min and max performance values (min-max normalization [6]). For example, the best attribute value were normalized with a value of 10, the worst with a value of 0 and all others in relation between the others [5]. However, the previous approach has some drawbacks. The major drawback is when having outliers, which is a data object that deviates significantly from the rest of the objects [6]. Thereby the normalization is distorted resulting in a lower distinction between the normal values. Furthermore, even when having a low performance distribution (when the values are very similar to each other), the worst performance is always normalized with zero. Therefore, the simple additive weighting, where the performances are multiplied with the defined weights, caused that services generally had been eliminated, when showing the worst performance in one of the attributes, even if the difference was little.

Therefore, instead of normalizing within the max and min performance values, new max and min thresholds were defined to solve the mentioned drawbacks. Table 4.2 presents the new normalization approach for the quality attributes response time and availability.

	Response Time	**Availability**
MAX Threshold	MAX := $(2 * \sigma) + \mu$	MAX := highest availability
MIN Threshold	MIN := lowest response time	MIN := $if\ (\mu - (2 * \sigma)) < 0\ then\ 0$ $else\ (\mu - (2 * \sigma))$
Normalization	NORM := $\left(\frac{(\text{MAX} - \text{RT})}{(\text{MAX} - \text{MIN})} * 10\right)$	NORM := $\left(\frac{(\text{AV} - \text{MIN})}{(\text{MAX} - \text{MIN})} * 10\right)$
μ = average or expectation / σ = standard deviation		

Table 4.2: Threshold definition within the normalization for the attributes response time and availability

For achieving the improved normalization, the max and min thresholds are set considering an outlier detection. In literature, a data object is defined as an outlier, when the value is exceeding two times the standard deviation from average or when falling below two times the standard deviation from average [6]. This concept for detecting outliers is applied within threshold definition. Thereby first, either an upper or a lower threshold is fixed corresponding to the best performance value. For example, for the quality attribute response time, where low values are desired, the MIN threshold is set with the best performance value, so that the best performance gets the maximum value of ten. For the quality attribute availability, where high values are desired, the MAX threshold is set with the best performance value, so that the best performance gets the maximum value of ten. The other upper or lower thresholds are accordingly set with two times of the standard deviation from average (as seen in Table 4.2). It has to be mentioned, that there is no upward limit for the attribute response time. On contrary, for the attribute availability there is a downward limit, because a negative availability in fact is not possible. Therefore, the MIN threshold for the availability is set to zero when it gets negative. The overall advantage of this presented normalization approach is that outliers are not distorting the normalization and the worst performance is not eliminated, except it is an outlier.

Table 4.3 illustrates the normalization process. For the corresponding response time and availability values the mean, standard deviation and the thresholds are listed. As seen in the table, the service *DE2Service* with the best response time is normalized with the highest possible value of ten. For the attribute availability, the service *DE1Service* with the highest availability gets the highest possible value of ten. All other values are relatively normalized to the defined MAX and MIN thresholds (corresponding to the normalization formula in Table 4.2).

Service	RT	RT NORM	AV	AV NORM
DE1Service	302	7.723	0.961	7.470
DE2Service	240	10.000	0.874	1.825
SE1Service	389	4.529	1.000	10.000
US1Service	457	2.032	0.922	4.939
Mean	347	/	0.939	/
Std. Deviation	82.67	/	0.047	/
MAX Threshold	512.34	/	1.000	/
MIN Threshold	240	/	0.846	/

Table 4.3: Normalization example of the service quality attributes response time (RT) and availability (AV)

4.3.4 Best-fit Service Selection

This subsection exemplary describes the process of determining a best-fit service from calculating the utilities, selecting the best-fit service, up to the adaption of the foreground-table and recommendation of the best-fit service.

Table 4.4 shows the determination of a best-fit service with corresponding utility functions and the normalized responses from the example above. The utility functions were defined as follows (the quality attributes in brackets represents the normalized values):

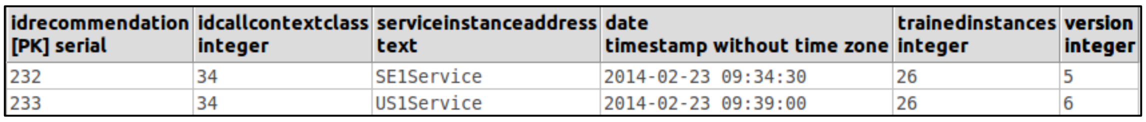
- *U1* = 0.5 x [*response time*] + 0.5 x [*availability*]
- *U2* = 0.2 x [*response time*] + 0.8 x [*availability*]

Within the utility function *U1* each of the attributes are weighted with 0.5 and no particular quality attribute is preferred over the other, whereas within the utility function *U2* a stronger preference for the attribute availability is predominant. This effects the service selection as shown in the table below. The service *DE1Service* within the utility function *U1* achieves the highest score of 7.6 and therefore is selected as best-fit service. On the contrary, when having a stronger availability preference (utility *U2),* the best-fit service with a score of 8.9 is service *SE1Service*.

Service	RT-Norm	AV-Norm	*U1*	*U2*
DE1Service	7.723	7.470	**7.600**	7.521
DE2Service	10.000	1.825	5.913	3.460
SE1Service	4.529	10.000	7.265	**8.906**
US1Service	2.032	4.939	3.486	4.358

Table 4.4: Determination of a best-fit service with corresponding normalized performances and utility function

After the best-fit service is determined a comparison and update of the foreground-model takes place. If the current determined best-fit service differs from the service listed in the foreground-table, a new service recommendation is inserted. As seen in Figure 4.3, the recommended service *SE1Service* is replaced for the call context class *34* with the service *US1Service* after 26 instances were trained within a time period of less than five minutes.



idrecommendation [PK] serial	idcallcontextclass integer	serviceinstanceaddress text	date timestamp without time zone	trainedinstances integer	version integer
232	34	SE1Service	2014-02-23 09:34:30	26	5
233	34	US1Service	2014-02-23 09:39:00	26	6

Figure 4.3: Example of best-fit service recommendations for corresponding call context classes

During the development, it appeared that in some cases, especially within discretized response times, two or more services had the same utility. To avoid a random selection several approaches can be applied. Either the service with the highest number of submitted feedback tickets within the call context class is selected, arguing that the data basis is more trustfully, or the other way around, arguing that a performance change is more unlikely because of the lower service consumptions.

As described in Section 3.2, the idea of the foreground-model is to have all current best-fit services listed in one table. When a consumer requests a service, the call context and utility function of the consumer are submitted within the request. After the pre-processing of the request, a single SQL SELECT query is executed to get the current-best fit service. Figure 4.3 shows the structure of the SQL SELECT query to determine the best-fit service instance address. The query includes a sub query with the pre-processed call context attributes to get the ID of the call context class. The query result is ordered descending by version and limited to one result to get only the latest service recommendation.

```
SELECT serviceinstanceaddress
FROM centralcomponentschema.recommendation
WHERE idcallcontextclass =
   (SELECT idcallcontextclass
   FROM centralcomponentschema.callcontextclass
   WHERE externalip = 'DEConsumer'
   AND weekday = 'Tue'
   AND daytime = '11'
   AND functionalserviceidentifier = '{}:Scenario1'
AND utilityfunction = '0.5*responsetime+0.5*availability')
ORDER BY version DESC
LIMIT 1;
```

Figure 4.4: SQL SELECT query for determining best-fit web service

5 Evaluation

This chapter presents the evaluation of this work. Thereby not only evidence is given, on which degree the machine learning and performance prediction is accomplished, but also on how the service selection could be optimized in general. In the first section, the created test data scenario used for the evaluation is described and illustrated. The second and third section focus on the overall evaluation with respect to the criteria defined in Section 2.2. The machine learning methods are evaluated separately (5.2), whereas the central service broker with the optimized service selection is evaluated on the whole (5.3).

5.1 Test Data Scenario

Within this chapter, the web service scenarios for evaluating both the machine learning methods and the overall central service broker are presented (for the simulation parameter definition, see Appendix A.2). The aim of the scenario creation was to imitate typical characteristics and behaviors of real-world web services. As described in 4.1, the test data generation tool offers an extensive selection of simulation parameters for creating sophisticated test scenarios with no limits regarding data volume. Because of the many input parameters and their resulting dependencies, the definition of test data scenarios is confronted with a certain complexity. Especially when dealing with a high number of unique participants and service providers the complexity accordingly increases. Therefore, the main objective was to focus only on a few but important aspects. The test data scenario with the definition of the service providers, service consumers and the selected simulation time period are outlined in the next part.

<u>Service Provider</u>
As described in 4.1.1 real-world web services can show cyclic performance behaviors, performance changes over time, performance differences caused from geographical network latency and general performance volatilities. All these different characteristics were considered when defining the test case scenarios.

Because the optimization of the service selection takes place within a group of services offering the same functionality, it seemed not necessary to have services representing different functionalities. Certainly, there are services showing differences in service quality depending on the service functionality or the data input. For example, the performance of a translation service strongly depends on the input size, whereas the input size of weather service request via a zip code does not. Overall, the focus was to reduce complexity by having only as many services as necessary showing those quality aspects, which are mentioned above.

For the test data scenario four services instances were selected. The first two letters of the service name indicate the service location. The services *DE1Service* and *DE2Service* are located in Germany, the service *SE1Service* in Sweden and the service *US1Service* in the United States.

The network latency caused from the different geographical location was also considered for the four service instances. Hereby the expected response time was adapted accordingly to the geographic distance. The intention was not to correctly define the network latency of each service, but moreover to show performance differences caused from different geographical located services.

<u>Service Consumer</u>

For each service instance four unique consumer were defined with 16 service consumers. Because of the mentioned complexity, all service calls take place from consumer located in Germany and identified with the discretized IP *DEConsumer*. The number of consumers and the call frequency was selected equally for each service instance to have a representative comparison not influenced on differences in data volume.

Time Period and Data Volume
The simulation takes place within a 30-day time period. Each of the service providers are requested by the consumer with a call frequency of 90 seconds. The resulting test data set contains of 460,800 unique service calls.

The characteristics of the performances response time and availability are analyzed in detail in the next subsection. In the first part of the analyses for each performance attribute, the overall performance trend is illustrated and the specifics are highlighted. The second part focuses on time specific characteristics with an analysis of weekday and daytime behaviors.

5.1.1 Response Time

Figure 5.1 shows the overall response time trend in the course of time separated for each service. The response time ranges from 220 up to 340 milliseconds within the 30-day time period.

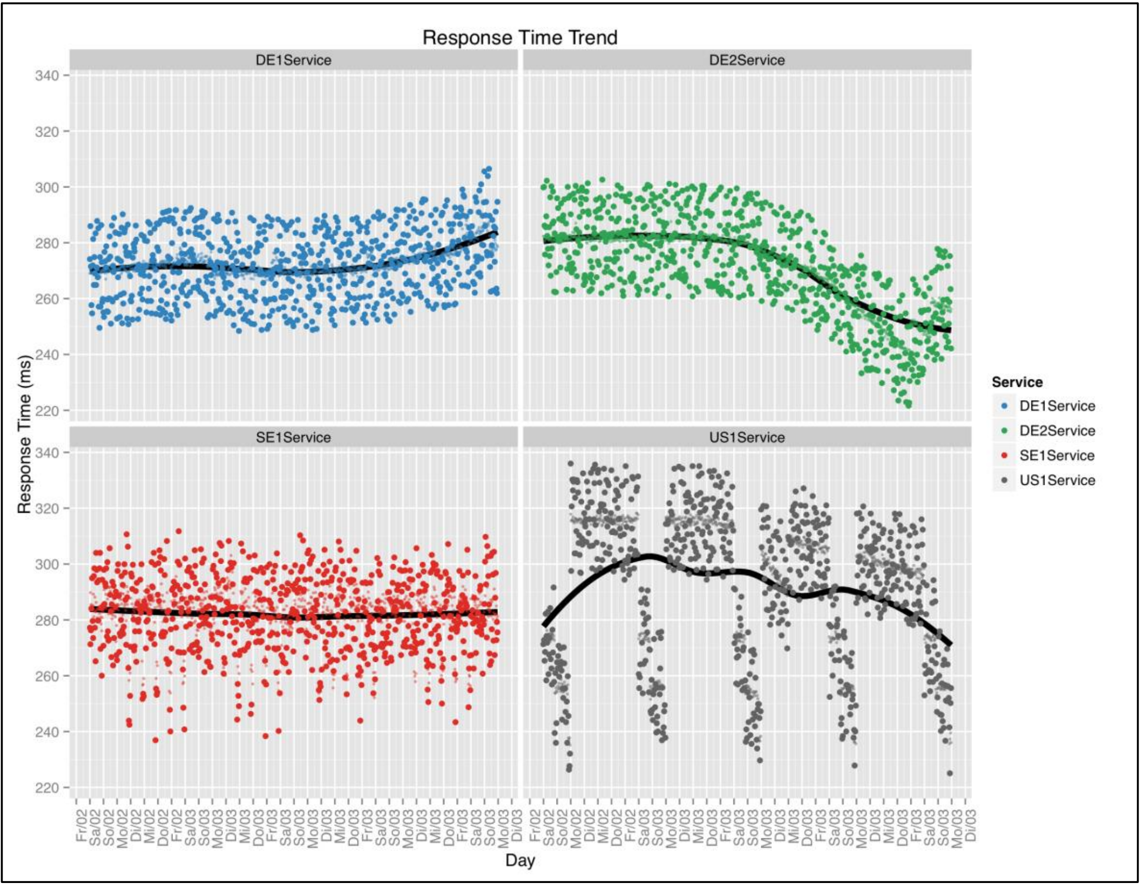


Figure 5.1: Overall response time trend separated by service instances

It has to be mentioned that because of visualization purposes, displaying more than 100K single records for a single service instance, the response time was aggregated by hour. Therefore, especially extreme outliers are exceeding the displayed range, but are not appearing on the figure. The aggregation also makes the visualization of the response time development more compact in general.

The overall performance of the service *DE1Service* is decreasing in the middle of the time course form an average of 270 to over 280 milliseconds. On the contrary, the performance of the service *DE2Service* is improving from 280 to less than 250 milliseconds. The service *SE1Service* constantly develops with a response time over 280 milliseconds. A closer look on the distribution shows that the service has a stronger volatile behavior than the other services. The service *US1Service* not only shows a higher mean response time compared to the other services, but also a significant cyclic behaviors. A detailed look on the figure reveals that this service has good performances on weekends from Saturday to Sunday. For an additional analysis, showing the overall response time trend together on one figure, see Appendix A.3.

Overall, the two German services *DE1Service* and *DE2Service* are showing the best performance with the lowest response time in the main time. As described before, the service *DE1Service* having the best performance from the beginning, but is replaced shortly in the middle of the time course. The performance of service *DE2Service* is then increasing from 275 below 250 milliseconds. The service *US1Service* generally seems to be out of competition except on weekends where the response time decreases down to 240 milliseconds.

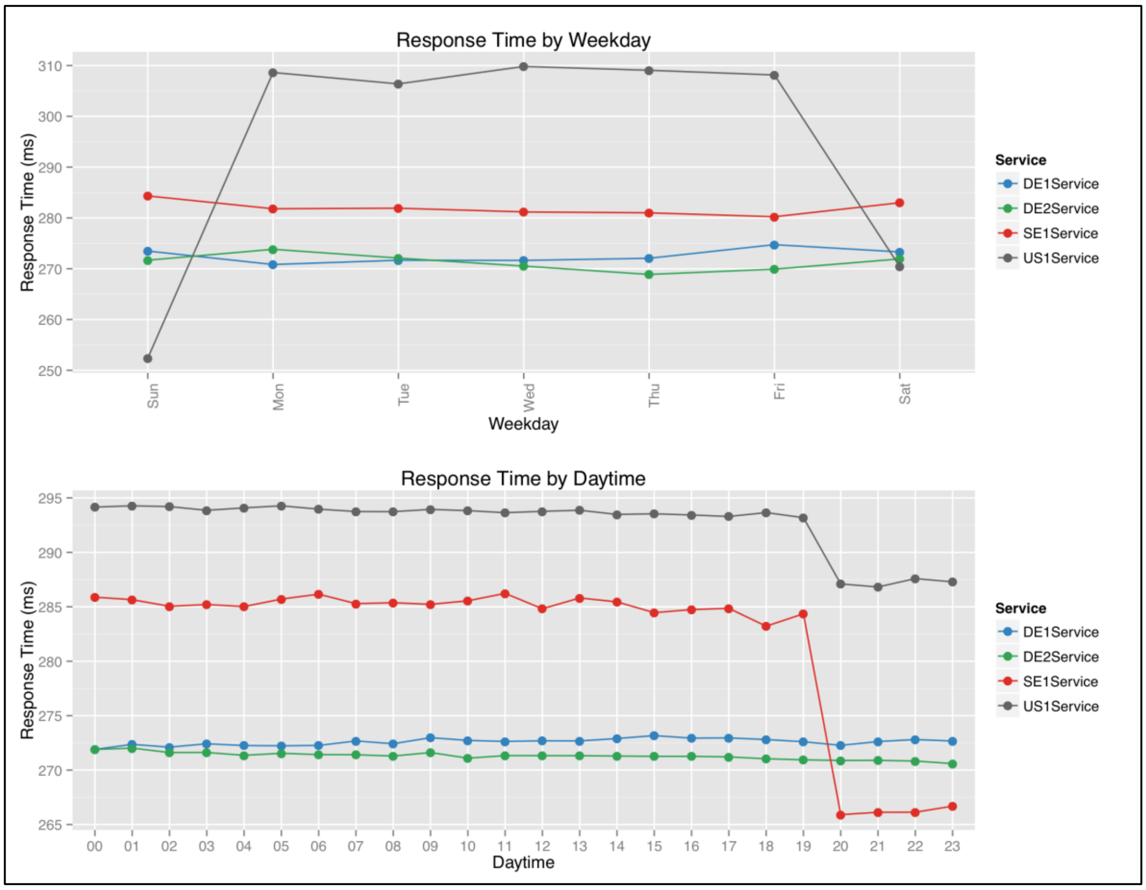


Figure 5.2: Response time aggregated by weekday and daytime

Figure 5.2 shows the response time aggregated by weekday and daytime. Because the performances can be developing very diverse in the course of time, it has to be mentioned that the aggregated data does only allow making statements about the general and overall trend at the end of the simulation time.

As already clearly seen in the response time trend, the service *US1Service* shows the best performance on Saturdays and Sundays. On the other weekdays, the services *DE1Service* and *DE2Service* compete for the best response time. Within the daytime analysis both German services show the best performance with the lowest response time values, except on evenings after 20 pm, where the service *SE1Service* performs better.

5.1.2 Availability

Figure 5.3 shows the overall trend of the quality attribute availability in the time course of the 30-day time periode. The overall availability of the services is aggregated by day and ranges between approximately 96% up to almost 99%.

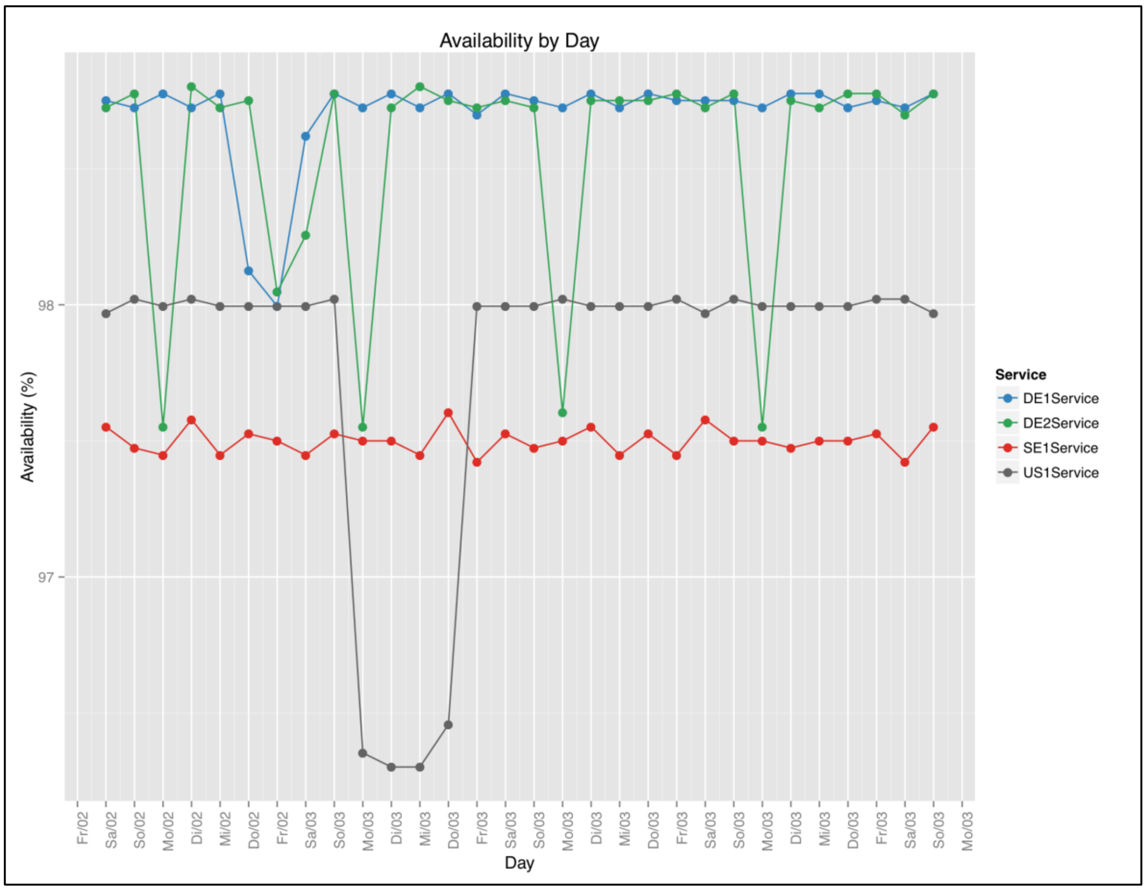


Figure 5.3: Overall availability trend

After a few days from the beginning of the time course, the availability of the service *DE1Service* is dropping down to 98% for a duration of three days. Service *DE2Service* shows weekly repeating decreases in availability. On Mondays, the availability decreases from an average of 98.7% down to 97.5%. The availability of the Swedish service *SE1Service* develops constantly in the range of approximately 97.5%. The service *US1Service* has a performance incident shortly after the first week. Here the availability decreases from 98% down to almost 96%, but recovers after the duration of four days. In total, both German services show the highest availability in the main time of the 30-day simulation period.

The data in Figure 5.4 reveals some different results regarding the service availability aggregated by weekday and daytime. Another striking result is the performance decrease of the service *DE2Service* on Monday mornings at 9 am. All other services behave relatively constant. Altogether, service *DE1Service* and *DE2Service* have the overall highest availability, whereas service *SE1Service* and *US1Service* can be ranked on third and fourth place.

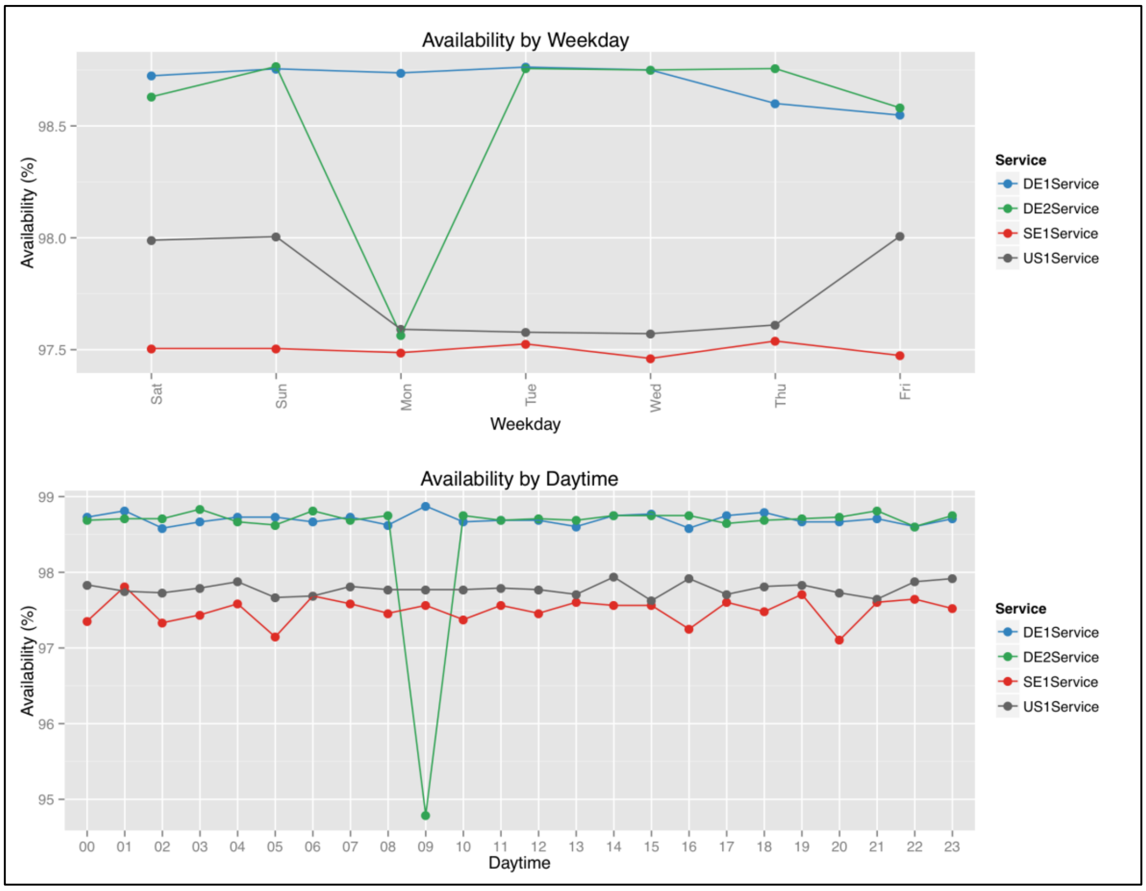


Figure 5.4: Availability aggregated by weekday and daytime

Table 5.1 finally shows an overview of the main statistics (mean and standard deviation) of the two quality attributes for the corresponding service instances. The mean values of the response time and availability attribute are generally close together. The value distribution was defined on purpose to challenge the machine learning methods. Otherwise, if the value range would be strongly distinguishing, the results might be getting too obvious.

In total, service *DE2Service* performs best regarding the overall mean response time, but *DE1Service* follows closely behind. Service *DE1Service* has the lowest deviation in response time, whereas service *SE1Service* has the highest. Service *DE1Service* is the most available service, whereas service *SE1Service* has the lowest availability as well as the most deviation in availability.

Service	Instances	RT Mean (ms)	RT Std. Dev. (ms)	AV Mean (%)	AV Std. Dev. (%)[4]
DE1Service	115.200	272.591	19.175	98.700	11.329
DE2Service	115.200	271.297	27.273	98.554	11.938
SE1Service	115.200	282.040	49.595	97.500	15.612
US1Service	115.200	292.699	46.648	97.780	14.732

Table 5.1: Statistics of the attribute response time and availability within the created test data scenario

Overall, the illustrated and described test data scenario shows a variety of different service behaviors. The scenarios were defined as they are because of illustration, interpretation and evaluation purposes. The objective was to show certain possible real-world service characteristics with focus on the development in the time course. In real-world there obviously exist other scenarios, where web services behave more volatile, have a much wider range of response time development or act more differently among each other. The availability of commercial web services would also be in the main time possibly close to 99.9%.

The main restriction of the simulation approach is that even when aiming to imitate real-world services, the evaluation results based on this data cannot be generalized. In real-world there are always unforeseen things, which cannot be considered in the overall model of the test data generation. Another restriction of the created test data scenario is the equal data volume for each service instance. In real-world an equal data basis for each service instance within a functional service group would be rather unlikely. In future work, the effects of an unequal data basis could be further analyzed (see subsection 6.3.4). The simulation tool enables to create corresponding test data scenarios having service instances consumed from with different call frequencies.

Nevertheless, the ability flexibly creating a variety of different service characteristics in a short amount of time with no restrictions in data volume, were mandatory features for this evaluation.

5.2 Machine Learning Methods

This section evaluates the machine learning methods regarding the criteria speed and accuracy. The methods are evaluated with a *10-fold cross-validation*. Hereby the initial data created within the test data scenario is randomly partitioned into ten folds. The learning and prediction is performed ten times, each time one fold is learned by the machine learning method and all other folds are tested (predicted or classified). For example fold one is trained and fold two to ten is tested, then fold two is trained and fold one, three to ten are tested and so on [7]. The 10-fold cross-validation is executed three times and the average is taken.

5.2.1 Speed and Accuracy

Table 5.2 presents the evaluation results of the selected machine learning methods for predicting the performance attribute response time. Hereby the time of training/learning (in seconds) and for a uniform accuracy method the mean

[4] Given that *available* means 100% and *non-available* 0%.

absolute error (in milliseconds) were assessed. Within the classification models, where instead of numeric values a class label is predicted, the center of the predicted value range is selected. For example having the class '150-200', the numeric value of 175 is selected.

The results show that the time of training, having more than 100 thousand instances for a single service, is very low in general. Nevertheless, there are significant performance differences between the several methods. Overall, the M5P and J48 are dismissed because of the worse time of training. The other methods **FIMT-DD**, **Hoeffding Tree** and **Naïve Bayes**, highlighted with a light gray background, are selected for the further evaluation (see next section).

Training[5] (sec)				**Mean Absolute Error (ms)**				**Method**	**Lib**
DE1	DE2	SE1	US1	DE1	DE2	SE1	US1		
0.839	0.671	0.701	0.700	6.212	8.471	12.310	10.339	**Naïve Bayes** (Batch Dis.)	Weka
0.671	0.671	0.697	0.685	12.887	11.895	12.882	11.752	**Naïve Bayes** (Inc. Dis.)	Weka
0.206	0.095	0.098	0.099	6.409	8.526	12.473	10.798	**Naïve Bayes** (Batch Dis.)	MOA
0.114	0.097	0.095	0.095	12.915	12.024	12.904	11.869	**Naïve Bayes** (Inc. Dis.)	MOA
5.959	5.822	5.890	5.867	6.118	8.411	12.313	10.294	**J48** (Batch Dis.)	Weka
3.103	3.180	3.789	3.529	12.869	11.894	12.882	11.745	**J48** (Inc. Dis.)	Weka
0.805	0.613	0.601	0.604	6.118	8.411	12.313	10.294	**Hoeffd. Tree** (Batch Dis.)	MOA
0.379	0.441	0.504	0.452	12.869	11.894	12.882	11.745	**Hoeffd. Tree** (Inc. Dis.)	MOA
0.640	0.465	0.399	0.397	67.965	70.231	90.325	91.421	**FIMT-DD** (Regression)	MOA
133.57	141.36	143.23	138.69	17.763	22.198	45.934	40.385	**M5P** (Regression)	Weka

Table 5.2: Evaluation results of the machine learning methods for the criteria speed (time of training) and accuracy (mean absolute error)

In the following part, an explanation for this selection considering several aspects is given.

Incremental VS Batch Discretization

The batch discretization, as described in subsection 4.2.2, shows an overall lower mean absolute error than the incremental discretization. This is because instead of discretizing by a static value range (incremental discretization) a best-fit discretization considering the value distribution is applied. On the other side, therefore the overall training time for the batch discretization is generally a little higher. Although the mean absolute error of the incremental discretization is twice

[5] The time of the batch discretization is additionally included.

as high as the batch discretization, it is generally low with only approximately 12 milliseconds (the average response time deviation is more than 35 milliseconds, see Table 5.1). Because the optimized service selection is implemented as a continuous process, the batch discretization is unsuitable. For the later evaluation, therefore the incremental discretization is chosen.

<u>Weka VS MOA</u>
The machine learning methods within MOA perform generally better with respect to the training time than those within Weka. The same Naïve Bayes algorithm within both libraries shows different evaluation results. This is because of the different types of learning. MOA is optimized for massive data streams, which is why the methods are based on incrementally learning. As seen from the table, the Naïve Bayes within MOA has a more than six times lower training time. The MOA library is selected for the central service broker implementation because of the generally better performance.

<u>Regression VS Classification</u>
Surprisingly, the main difference between the regression methods (FIMTD-DD, M5P) and the classification methods (Naïve Bayes, J48, Hoeffding Tree) is the generally much higher mean absolute error of the regression methods. Despite the discretization, where information gets lost, the classification methods have a very high accuracy. Noticeable is also that the mean absolute error is developing corresponding to the mean deviation of each service. Service *SE1Service* and *US1Service*, which have the highest response time deviation (see Table 5.1), at the same time have the highest mean absolute error. Despite the high mean absolute error, the FIMT-DD is selected, because it is assumed that the 10-fold-cross-validation is not the optimal evaluation method also with respect to the drift detection function.

5.3 Optimized Service Selection

Objective of this section is the overall evaluation of the central service broker with focus on the foreground-/background-model. Within the evaluation the selection quality, speed, scalability, robustness and utility prediction of the optimized service selection is assessed. The evaluation runs were not executed in real-time, but in sequential batch mode. Thereby, the created test data first is loaded into the main memory. Thereafter each single instance is chronological submitted to the service broker for further processing within the service optimization process. This evaluation approach was required for a flexible and quick repetition of the evaluation runs having multiple evaluation settings and machine learning methods.

The created web scenario with completed test data pool allows an insight into the web service characteristics at any time. The evaluation thereby follows the approach: when the service optimization for a certain call context class is triggered, the performances from the learned past service calls were predicted, and on the other hand, the future performances (as actual performances) within this call context class were directly fetched from the completed test data pool. This allows, after normalizing all performances and determining the best-fit services, the comparison of the actual and predicted best-fit service.

The requirements for running the evaluation were to have an optimal hardware configuration without restrictions neither in computation nor in main memory storage. The first development system was configured within a virtualization environment. Here the evaluation execution caused several performance issues. The whole development environment therefore was natively installed on a Linux OS (Ubuntu 12.04 LTS) notebook with an i5-3340M CPU @ 2.70GHz × 4 (64Bit) and 15.6 GiB RAM. The central service broker is executed within eclipse enabling 10GiB of RAM.

5.3.1 Selection Quality

Table 5.3 presents the final evaluation results of the service broker's selection quality. The data shows two evaluation types with respect to the strategically learning: the results for learning after each 100 instances (A) and after each 10 instances (B). The measure *TOP1 Accuracy* gives evidence on the selection quality as percentage of the correctly determined best-fit services with respect to the actual best-fit services, whereas the measure *TOP2 Accuracy* shows the percentage of the correctly determined best-fit services within the actual two best-fit services. The third measure *Mean Absolute Error* displays the mean deviation of the calculated and the actual utilities. The last measure *Recommendations* shows the number of inserted recommendations within the foreground-table.

	Naïve Bayes	**Hoeffding Tree**	**FIMT-DD**
A) Strategically Learning after each **10 Instances**			
TOP1 Accuracy (%)	59.245	60.751	70.039
TOP2 Accuracy (%)	89.397	89.839	93.863
Mean Absolute Error (Utility)	1.682	1.674	1.029
Recommendations (# records)	1554	1526	3687
B) Strategically Learning after each **100 Instances**			
TOP1 Accuracy (%)	58.634	59.837	69.287
TOP2 Accuracy (%)	90.163	90.421	93.471
Mean Absolute Error (Utility)	1.656	1.660	1.049
Recommendations (# records)	659	647	1189

Table 5.3: Evaluation results of the machine learning methods Naïve Bayes, Hoeffding Tree and FIMT-DD within the optimized web service selection

The results are quite revealing in several ways. Unlike the results of the machine learning methods from the previous section, the FIMT-DD algorithm outperforms all other algorithms with the highest *TOP1* and *TOP2* selection accuracy in both evaluation types A and B. The FIMT-DD shows an overall selection accuracy of 70%, which means that in more than 70 percent of all service optimizations the actual best-fit service could be determined. Within the *TOP2* measure, the selection accuracy for determining either the best or the second best service increases up to almost 94%. The results of the machine learning methods Naïve Bayes and Hoeffding Tree are very similar in both evaluation types with an overall *TOP1* selection accuracy of approximately 60% and a *TOP2* accuracy of approximately 90%.

Surprisingly, a higher learning frequency, in this case the difference of learning 100 and 10 instances at a time, does not have a significant effect on the selection quality. Within the ten times higher learning frequency a selection accuracy of less than one percent is gained. This shows that for finding a best-fit learning strategy a tradeoff between accuracy and learning effort has to be made. Interestingly, there are significant differences in the update behavior of the foreground-model. Within the FIMT-DD more than 3600 recommendations were inserted, whereas within the other methods less than half of this number were inserted. Furthermore, the utility mean absolute error of the FIMT-DD algorithm (about 1) is generally low in comparison to the other methods (about 1.6). The mean absolute error compared to the two evaluation types and within the machine learning methods does not show significant differences.

Figure 5.5 shows the service selection accuracy of the three algorithms FIMT-DD, Hoeffding Tee and Naïve Bayes in the course of time. The analysis comprises the evolving *TOP1* selection accuracy with the evaluation type A from Table 5.3 of the three machine learning methods with strategically learning after 10 instances (for an analysis of the *TOP1* and *TOP2* selection accuracy, see Appendix A.4). It is apparent that the FIMT-DD has the overall highest selection accuracy with an approximately ten percent higher accuracy at the end.

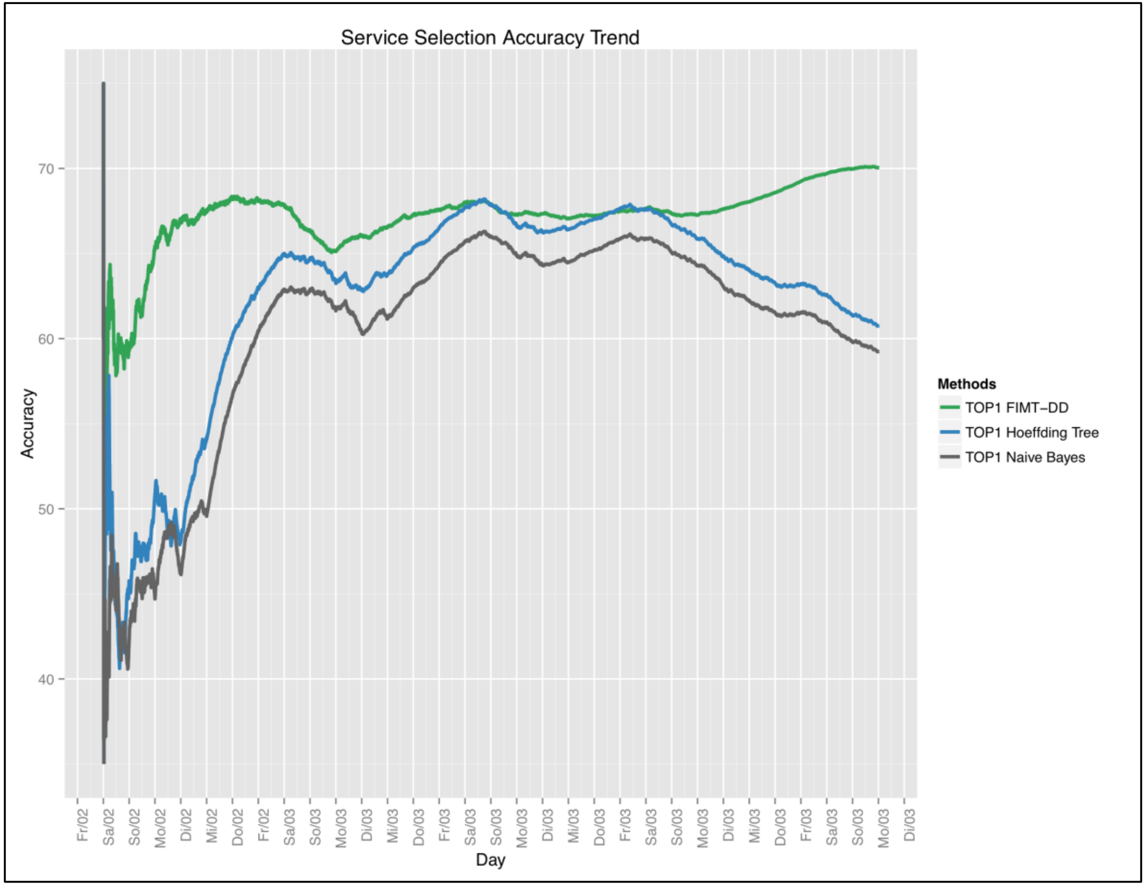


Figure 5.5: Service selection accuracy of the FIMT-DD, Hoeffding Tree and Naïve Bayes algorithm in the course of time

What is interesting in this data is the development in the beginning and at the end of the time course. During the first days, the FIMT-DD outperforms the other machine learning methods partly more than 20 percent, but the Hoeffding Tree and Naïve Bayes nearly catch up in the middle of the time course. During the last nine days, the selection accuracy of the FIMT-DD generally improves, whereas the accuracy of the other methods invades more than six percent. Analyzing the test data scenario from section 5.1 shows, that all services, but especially service *DE2Service* have performance changes in the response time at the end of the time course. The Naïve Bayes and the Hoeffding Tree are not able to handle this performance change, whereas the FIMT-DD is capable to detect and handle this performance drift. This is because both Hoeffding Tree and Naive Bayes are classifier without drift detection assuming that the distribution does not change over time.

Furthermore, the FIMT-DD has a more constant and straightened accuracy development (range of 60 and 70 percent) compared to the other methods, which are generally more varying (range of 40 and 65 percent). Surprisingly, the accuracy development of the Hoeffding Tree and Naïve Bayes is almost identical and only distinguishing constantly in two percent. Although the Hoeffding Tree is an incremental decision tree, the learning method in the leaves is based on the Naïve Bayes and therefore shows a similar accuracy.

Overall, the FIMT-DD performs better than all other machine learning methods with respect to the selection accuracy. In 70% of all service optimizations the actual best-fit service, in 24% the second best-fit service and in 6% the third or fourth best-fit service could be determined. The results are outstanding considering the wide variety of service behaviors with daily and weekly cyclic characteristics as well as performance changes and strong deviating services. The most striking result is the accuracy development at the end of the time course, where the FIMT-DD not only shows a perfect handling of the performance drift, but moreover improving the accuracy.

5.3.2 Speed and Scalability

The new implementation of the service broker reveals an excellent performance concerning the overall processing time. The incremental high-performance machine learning methods and their native Java implementation generally reason this. Table 5.4 shows the evaluation results with the *Processing per Instance* showing the time needed to process a single instance in milliseconds. Similar to the table presented in the subsection above, the two evaluation types with strategically learning after each 100 instances (A) and after each 10 instances (B) are shown.

As seen in the table, a single instance is processed in less than three milliseconds by an overall of 460 thousand records including the strategically learning, the service optimization as well as the update of the best-fit services within the foreground-model. It has to be mentioned that there is an additional overhead caused by evaluation activities such as measuring and monitoring the broker processing. The overhead is estimated to be less than 20% of the processing time.

	Naïve Bayes	**Hoeffding Tree**	**FIMT-DD**
A) Strategically Learning after each **10 Instances**			
Processing per Instance (ms)	22.997	23.080	23.129
B) Strategically Learning after each **100 Instances**			
Processing per Instance (ms)	2.602	2.614	2.621

Table 5.4: Time for processing a single instance in milliseconds of the machine learning methods Naïve Bayes, Hoeffding Tree and FIMT-DD

It is apparent from Table 5.4 that the processing time of these methods does not significantly distinguish from each other. All three methods are implemented within the high-performance MOA library. In general, the Naïve Bayes method has a slightly better performance than the methods Hoeffding Tree and FIMT-DD. When comparing the number of possible processed instances per hour the Naïve Bayes would be able to process approximately 900 instances more than the FIMT-DD. What is interesting in this data is that the time of processing is not increasing linear when increasing the frequency of the strategically learning. In this case, the processing time is only increasing by a factor of 8.8 when increasing the learning frequency by a factor of 10.

As described in 4.3.1, the Weka library does not fully support incremental loading for PostgreSQL, which is the reason why batch loading is applied in the new implementation. Concerning the fact that real incremental loading does have a higher overhead, the mentioned results therefore have to be questioned in general. The overhead results in fetching every single record via database instead of loading all records into the fast main memory. Since the execution time of a SELECT query, fetching a single feedback record, ranges below ten milliseconds in average, the overhead is relatively small. There also exists no network latency, having the database and central broker on the same machine. Therefore, the overall time for processing a single instance with incrementally loading is estimated in a range below 35 milliseconds under the same conditions with strategically learning of 10 instances at a time.

Comparing those results with the ticket persisting rate of the feedback interface reveals that the data incoming rate is much lower than the possible processing rate of the service optimization. Within the transmission of 460 thousand feedback tickets, the time for processing and persisting a single instance takes approximately 215 milliseconds.

In summary, the new implementation is capable of real-time processing when strategically learning after each 10 instances. The processing of a single instance within the optimized service selection takes less than 25 milliseconds, whereas the rate of processing and persisting the submitted feedback tickets takes more than 215 milliseconds. That implies that within the current implementation state the feedback service interface is the bottleneck. The time for requesting a best-fit web service from consumer side, does not present any performance issue. Because the best-fit services are pre-determined and listed on side of the foreground-model, the best-fit service selection only consists of the call context pre-processing and a single SQL query. Thereby the overall time constantly ranges between less than ten milliseconds.

5.3.3 Robustness

In this subsection the robustness of the central service broker, with respect to the defined criteria in subsection 2.2.4 is evaluated. A criterion for the robustness was how a continuously changing environment with new service candidates and new attributes is handled. The integration and native Java implementation of Weka and MOA enables a high degree of automation. New service candidates can be simply handled when having an object-oriented environment by just creating a new service object with corresponding prediction model. The handling of new attributes on the other hand, cannot be handled fully automatically, because there is always an adaption necessary within the pre-processing. The label attributes, which the learning models predict, also have to be declared within the service optimization.

A further defined evaluation criterion was, if the machine learning is able to handle noisy data such as attributes with missing values and outliers. Within the current approach for creating test data, missing values (for the attribute response time) only occur when having non-available service calls. For predicting the response time, those records are not considered and filtered out. In real-world there could occur other failures and therefore other types of missing values. The question is, whether the framework or the learning methods should handle the missing values. Within the framework, this could be easy accomplished by just filtering out the corresponding instances. However, this involves that useful information might get lost. Considering missing values within the machine learning gives an optimal utilization of the existing data, but on the contrary, the complexity of the machine learning increase and the prediction might get more prone to error. Before such architecture decision is made, comprehensive real world data should be collected. After knowing the scope and types of missing value attributes, the general feasibility of learning with missing values and their effects could be further analyzed.

The detection of outliers is managed indirectly by the current broker implementation. Within the normalization process, after the performances are predicted, the thresholds for the normalization are accordingly set with respect to an outlier detection. In contrary to this approach outliers also could be handled directly before the learning takes place. The Weka library provides several outlier detection functions, but all for batch processing mode. It showed that handling outlier within a continuous process is not trivial, because the overall value distribution is not known and can change in the course of time. This advanced challenge has to be analyzed separately, because of the high complexity.

5.3.4 Utility Prediction

This section analyzes the utility prediction with respect to striking service characteristics and service performance changes created in the test data scenario. For the analyses, the FIMT-DD algorithm, as overall best performing machine learning method, is chosen with strategically learning of 10 instances at a time.

<u>Overall Utility Trend FIMT-DD</u>
Figure 5.6 shows the development of the actual and the predicted utilities of the FIMT-DD algorithm in the course of time with the defined utility function '*0.5*responsetime+0.5*availability*'. As seen in the figure the utility trend, generally reflects the service characteristics created within the test data scenario. The services *DE1Service* and *DE2Service* showing the best results with the overall highest utilities, whereas the other services *SE1Service* and *US1Service* compete on the third and fourth place.

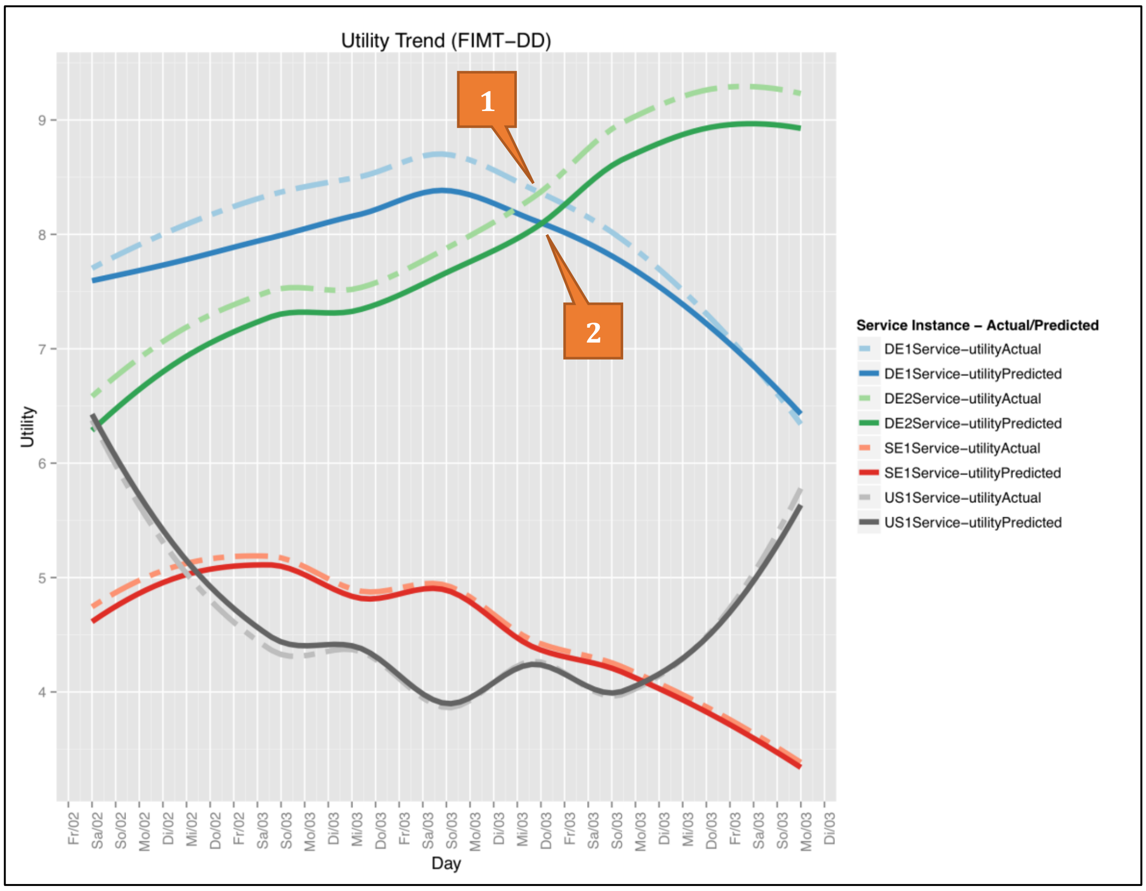


Figure 5.6: Overall utility trend of the FIMT-DD algorithm with actual (dashed lines) and predicted (solid lines) utilities and corresponding service instances

What is interesting is that the FIMT-DD method has an outstanding prediction accuracy. All utility changes are recognized and are nearly evolve identical to the actual utilities. The added notes, as seen in the figure, show the point in time when the best-fit service changes. Here service *DE1Service* leads until half time, where shortly after service *DE2Service* catches up. Noticeable is that the event of the actual (seen as note number 1) and predicted (seen as note number 2) service change takes place at the same time. Even though the predicted utilities of these two services have an average downward deviation of almost 0.5 points, the predicted utilities behave and develop the same way as the actual utilities.

Another interesting fact is that the predicted utilities of the services *SE1Service* and *US1Service*, which are generally more volatile than the two German services, do not show a strong deviation from the actual utilities. This might be affected from the normalization process, where the lower performance values are generally smoothened. For another analysis comparing the utility trend of the FIMT-DD with different utility functions, see Appendix A.5.

<u>Overall Utility Trend Hoeffding Tree and Naïve Bayes</u>

In comparison to the before described analyses Figure 5.7 shows the corresponding utility trends from the machine learning methods Hoeffding Tree and Naïve Bayes. The general deviation from the actual and the predicted utilities is striking when comparing these results with those achieved with the FIMT-DD. The utility of service *DE1Service* and *DE2Service* are evolving almost identically. Furthermore, the performance decrease of service *DE1Service* is not recognized both within the Naïve Bayes and within the Hoeffding Tree. Moreover, the utility increases at the end of the time course. Both methods show almost an identically utility development. As already outlined in subsection 5.3.1, this is because the Hoeffding Tree learning method in the leaves is based on the Naïve Bayes.

The general discretization process can reason the inaccuracy of those both methods, where information gets lost. On the other hand, the Naïve Bayes and Hoeffding Tree also do not have a drift detection, which is why at the end of this scenario the utilities are evolving with a strong deviation from the actual utility trend.

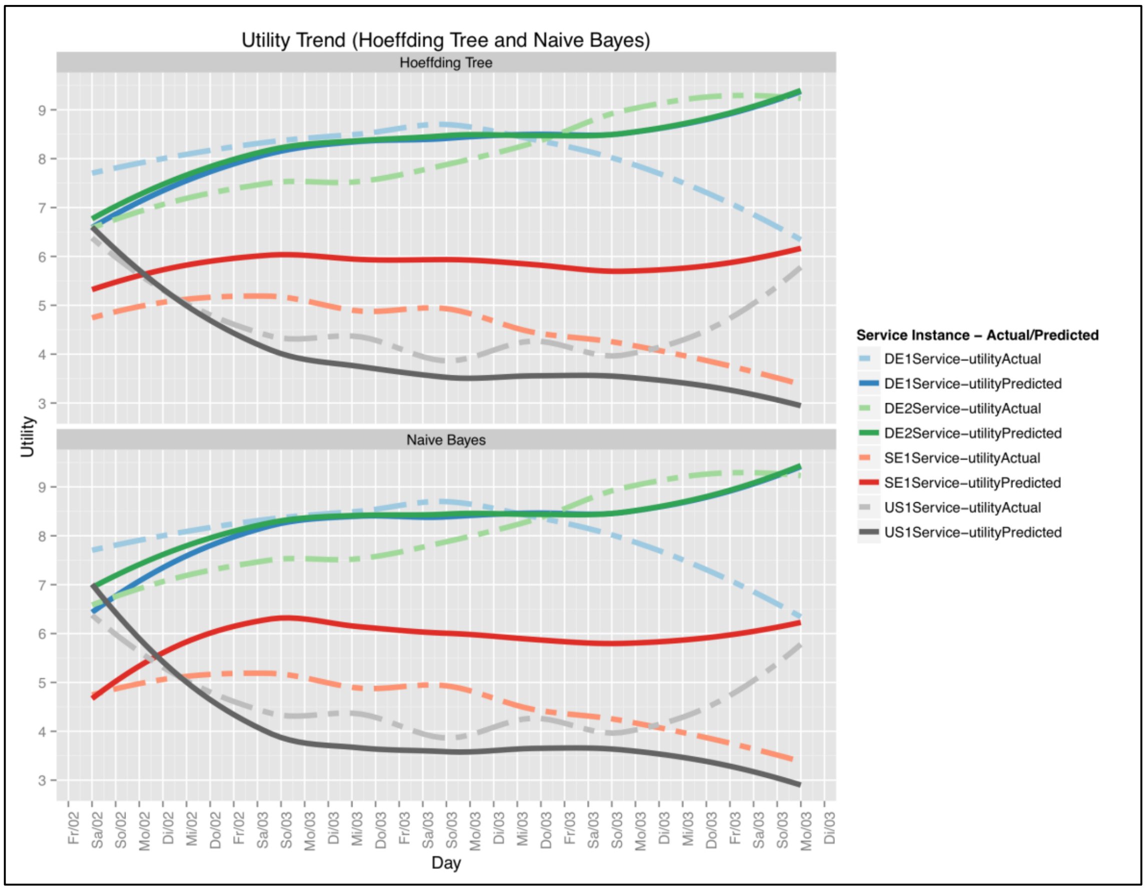


Figure 5.7: Overall utility trend of the Hoeffding Tree and Naïve Bayes

Workweek VS Weekend Trend

Within the test data scenario, the service *US1Service* showed noticeably good response time performances during weekends. As seen in Figure 5.8, those different performance characteristics are also reflected when comparing the utilities of weekends and workweeks. Despite the overall time course illustration, for the weekend only the utilities for Saturday and Sunday were selected, and for the workweek the utilities from Monday till Friday.

On the first two weekends, the service *US1Service* competes for the first place, whereas later in the course of time this service is ranked on second and third place. Contrary to that, the predicted utility trend does show different results. Here the service *US1Service* first is ranked on the second place, and later on the third place. In this case, the FIMT-DD did not recognize the best-fit service within the first two weekends. On the other hand, the utility prediction within the workweek has again an almost identical development as the actual utility trend.

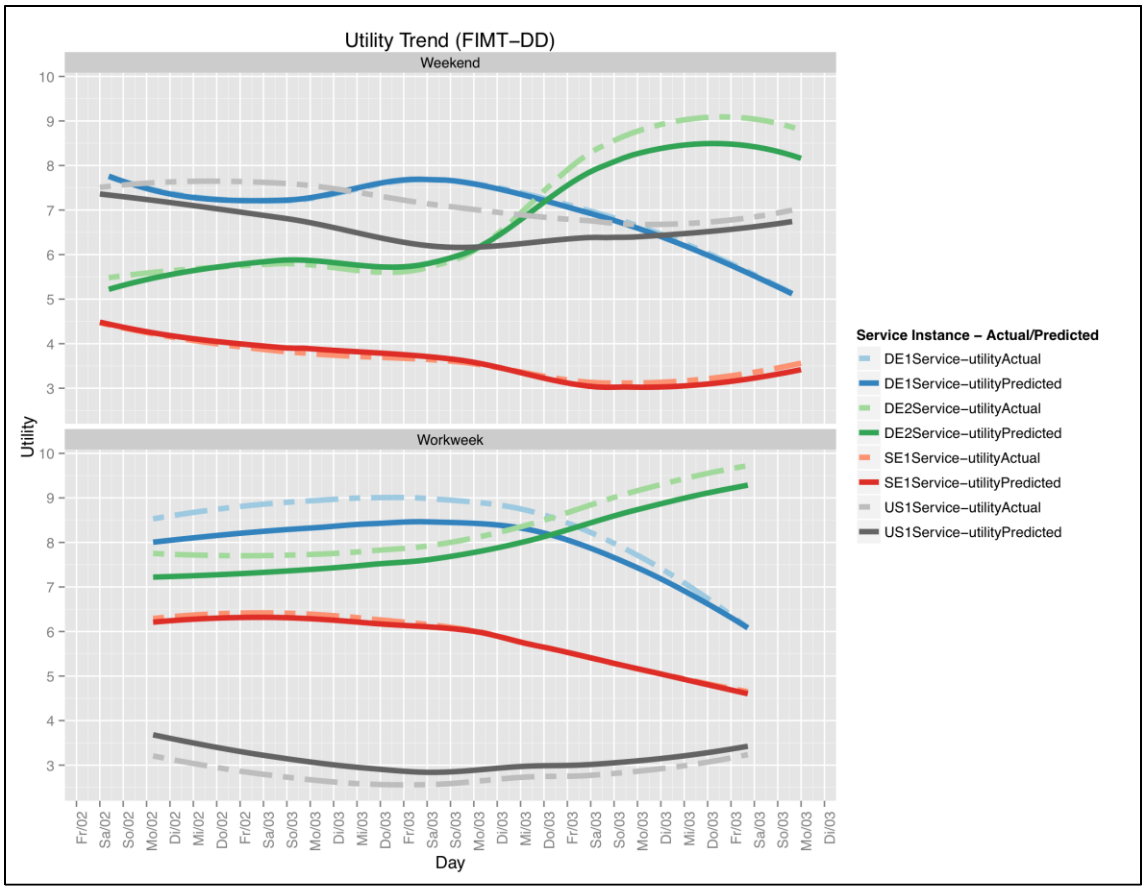


Figure 5.8: Weekend and workweek utility trend of the FIMT-DD

Monday Morning

The test data scenario showed that the service *DE2Service* has availability issues on Monday mornings. The analysis of the utility trend reflects this performance downfall. The figure below shows the utility development of the services on Monday at nine am with different utility functions (within the figure top there is a higher preference for availability, and within the figure below, the preferences are balanced). As seen in the figure service *DE2Service* is ranked on the fourth place. The overall prediction accuracy is very high. Furthermore, the effect of having different utility functions is also clearly shown. In the figure above with the higher preference for availability, the service has in general a lower average utility and distinguishing from the other services by more positioning apart. The utility trend evolves linear because of the less utility data (filtered by Monday 9 am).

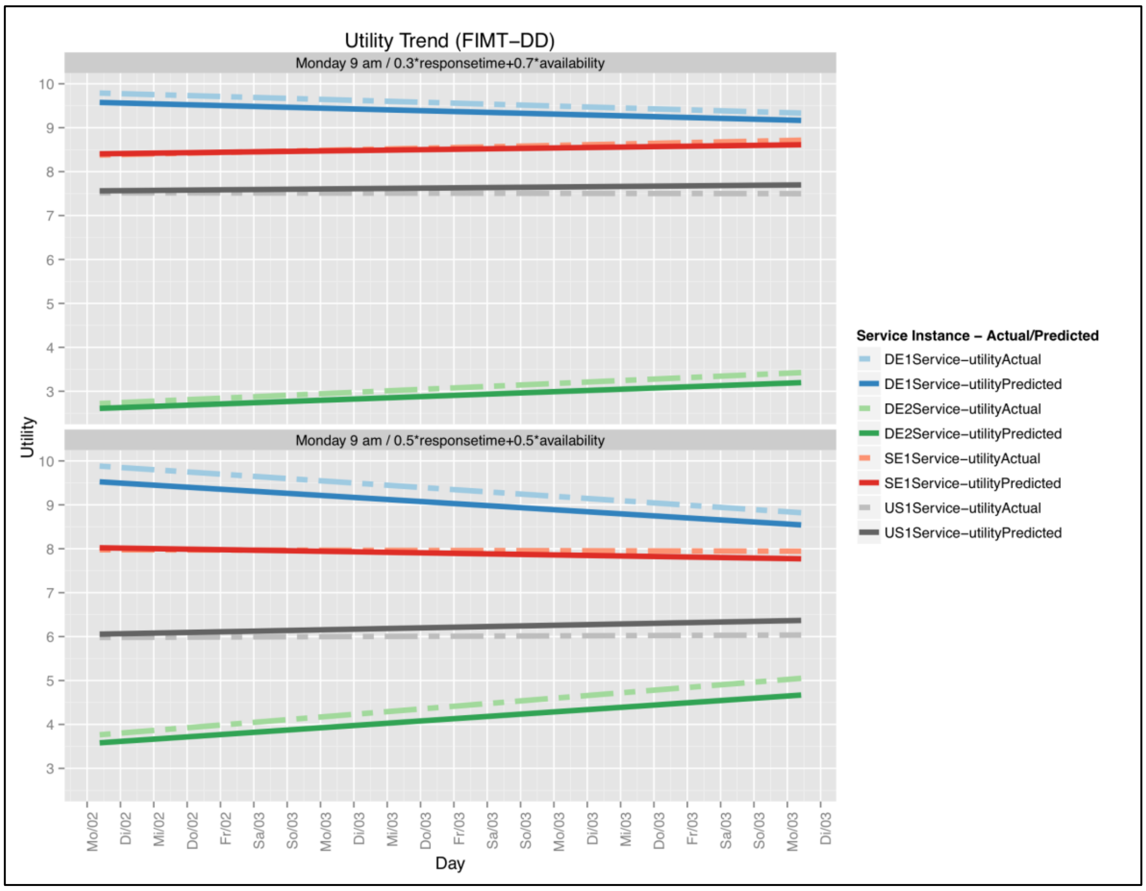


Figure 5.9: Utility trend of Monday mornings (9 am) with different quality preferences

6 Conclusion

This final chapter first concludes with an overall summary of this thesis work and thereafter compares the initially defined goal criteria with the thesis achievement. Finally, further challenges given from the implemented optimized service selection, which could be addressed within future work, are presented.

6.1 Retrospect

This thesis focuses mainly on the design, the implementation and the evaluation of machine learning methods for an optimized web service selection. It is founded on the SOC (Service Oriented Computing) research work of Kirchner, et al. assuming that there will be a future service market with a selection of functionally equal services. It was outlined that the resulting selection problem requires a distinction between non-functional service characteristics. Consulting non-functional service promises of providers (defined as Service Level Agreements) is not optimal because of several mentioned reasons. The research therefore proposed the development of distributed knowledge based on a central broker framework monitoring and automatically measuring the service quality during each service consumption. The broker aims at optimizing the service selection relying on the past service performances and the defined quality preferences of a unique consumer.

The challenges and restrictions of the current broker machine learning implementation were derived and corresponding evaluation criteria were defined. It was outlined that the performance and scalability of the broker's machine learning plays an important role and that the current classical batch decision tree learning is not appropriate with respect to a continuous and time-critical optimization process. Therefore, high-performance machine learning methods, which can handle high data volume and libraries with a high automation and integration degree were searched and pre-selected. A first implementation within a proof of concept showed that the alternative machine learning methods have constantly good performance and the selected library can be used and integrated with a manageable effort.

The conceptual design involved the optimization of the service selection with focus on a continuous machine learning process. The core idea aimed at separating the real-time request for a best-fit service selection from the time-consuming machine learning. This lead to the design of the so-called foreground-/background-model. Any performance issues within the critical path from the service request to the determination and recommendation of a best-fit service are thereby eliminated. The foreground-model basically holds all current best-fit services for certain consumer call context classes (call location and call time) and the consumer's service quality preferences (utility function) in an easy accessible lookup table. The learning, prediction and service optimization on basis of the past measured service calls takes place within the background-model. The asynchronous service optimization process continuously determines the best-fit services for each call context class. The introduction of a threshold mechanism allows adapting the learning frequency and defining a corresponding machine learning strategy.

Another core focus was the creation of appropriate test data with corresponding data characteristics and data volume for the overall evaluation. The development of a data generation tool allows creating sophisticated test data scenarios by simulating service consumptions with different call contexts and defined service quality characteristics (e.g., response time, availability). The implementation of the optimized web service selection consists of different activities for predicting the service performances, normalizing them (make them comparable) and calculating the consumers utilities with respect to their unique service quality preferences. Furthermore, a threshold mechanism for a strategically learning and continuously updating of the current best-fit services was implemented. It showed that there are new challenges within a continual learning process such as the discretization or the detection of outliers. For the latter an optimized normalization approach with an outlier detection was introduced.

For the overall evaluation, a sophisticated test data scenario was created with typically real-world service characteristics such as different volatile service performances, cyclic performance behaviors and performance changes in the course of time. The implementation of the machine learning showed a high degree of automation and an overall outstanding performance. Within the evaluation setting the real service quality (obtained from the completed test data pool as future performances) and the resulting actual best-fit service were compared with the predicted service quality and the recommended best-fit service of the machine learning process. The results revealed that the incremental regression tree *FIMT-DD* has the highest best-fit service selection accuracy. In 70% of all service optimizations, the central broker could determine the best-fit service and in 94% of all cases, the actual two best-fit services. Furthermore, the evaluation showed that the problem of a class label change in time course is omnipresent within the central broker setting. The *FIMT-DD* is able to detect and handle such performance changes with the integrated drift detection. The evaluation also showed that depending on the learning strategy the time of processing strongly varies. Overall, the time for processing a single instance is in a range of 23 milliseconds when optimizing the service selection after 10 instances at a time.

6.2 Achievement

This section serves as final overview comparing the thesis achievement with the initially defined success criteria.

High-Performance Service Determination
<u>Assessment</u> duration for requesting and receiving the broker's best-fit service selection (in milliseconds)
<u>Achievement</u> The best-fit services for certain call context classes are continuously pre-determined in the background and listed in the foreground lookup table. This approach significantly improves the time-critical request of a best-fit service. The service determination thereby consists of a single SQL query taking in average less than ten milliseconds.
High Service Selection Accuracy
<u>Assessment</u> percentage of correctly determined actual best-fit services (in percent)
<u>Achievement</u> In 70% of all service optimizations the actual best-fit service and in 94% of all service optimizations the actual two best-fit services could be determined within the FIMT-DD machine learning method and corresponding test data scenario.
Continuous Machine Learning Process
<u>Assessment</u> capability of the machine learning continuously processing and learning from past measured service calls

<u>Achievement</u> The conceptual design of the optimized web service selection is based as continuous machine learning process and the MOA library enables incrementally learning.
Performance Change Adaption
<u>Assessment</u> capability of the machine learning detecting performance changes
<u>Achievement</u> The FIMT-DD with integrated drift detection is capable recognizing performance drifts and accordingly adapting to the performance change.

Table 6.1: Achievement of the initially defined goal criteria

6.3 Future work

This work showed the capabilities and features of the continuous machine learning approach for an optimized web service selection. Despite the overall advantages and performance benefits of the presented implementation, there are further optimization approaches and advanced challenges, which could be worked on within future research.

6.3.1 Strategically Learning

The current implementation covers time and count thresholds for controlling the learning and optimization frequency. Within the current state, the threshold definition is not yet adapted for a best-fit learning strategy. The optimization depends on the underlying data in general and the scope of the call context classes as well as the number of submitted feedback tickets in specific. In future work learning strategies with respect to different scenarios could be defined and implemented. A scenario could include different parameters such as the frequency of submitted feedback tickets or the occurrence of certain call context classes, etc.

6.3.2 Production Deployment

The current implementation leaves space for optimization regarding the usage within a production environment. The restriction for incremental loading is not solved so far. Current development trends of the Weka library show that much effort is spend within the subject, so that it is a matter of time until incremental loading for PostgreSQL is fully supported. Although the central broker implementation runs as a continuous process and has generally a high degree of automation, the detection of new service instances is not handled so far. For handling new service instances the central broker framework have to be extended. In addition, the automatic service binding within the broker framework would have be adapted for a productive deployment.

6.3.3 'Concept Drift' Problem

This problem describes the situation when the values of the label attributes are significantly changing with respect to the course of time [7]. The machine learning algorithm FIMT-DD with integrated drift detection function could be further analyzed focusing on the algorithms parameter optimization with respect to the specifics of the SOC setting.

When having other machine learning methods without integrated drift detection, own strategies and methods for handling the distortion of data could be developed. Different learning models with adaptive time frames could be defined for applying a selective distortion of the data base. For example, learning models with time frames of two weeks, four weeks and an overall model. All models could be considered for calculating the utility but with a different weighting, focusing more on either current or past events.

6.3.4 'Lack of Feedback' Problem

The lack of feedback is a core challenge within the central service broker framework. A prediction is only as good as the data input itself and referring to the central broker framework the corresponding feedback tickets. The concept of measuring past service consumptions lead to the effect that there are no periodical calls in general. Therefore, a performance monitoring is only given when services are consumed. Especially when services are consumed more often, the number of submitted feedback tickets is accordingly higher compared to services with a low consumption. Within the introduced simulation tool, test data scenarios could be created having services, which are consumed variously often and the overall machine learning on an unsteady data basis could be further analyzed.

6.3.5 'Navigation Device' Problem

The 'Navigation Device' problematic is a metaphor for a traffic jam when the navigation system redirects everyone from the highway. Instead of solving the problematic situation, the traffic jam is shifted somewhere else besides the highway. Such behavior can also take place within the SOC setting, when a service showing the overall best performance and therefore is highly recommended. Thereby the number of service consumption is increasing, which in turn can lead to a decrease in performance. The general approach for taking countermeasures could be researched and corresponding strategies could be defined, implemented and evaluated.

7 List of References

[1] Jens Kirchner, Jesper Andersson, Andreas Heberle, Welf Löwe, “Service Level Achivements - Distributed Knowledge for Optimal Service Selection,”

[2] Pete Chapman (NCR), Julian Clinton (SPSS), Randy Kerber (NCR), Thomas Khabaza (SPSS), Thomas Reinartz (DaimlerChrysler), Colin Shearer (SPSS), Rüdiger Wirth (DaimlerChrysler), *CRISP-DM 1.0: Step-by-step data mining guide*, 2000.

[3] Michael Anderson, Susan Leigh Anderson, “Machine Ethics: Creating an Ethical Intelligent Agent,” *AI Magazine*, no. Volume 28 Number 4, pp. 15–26, 2007.

[4] I. H. Witten, E. Frank, and M. A. Hall, *Data mining: Practical machine learning tools and techniques, third edition,* 3rd ed. Burlington, Mass: Morgan Kaufmann Publishers, 2011.

[5] Markus Geisler, “A Machine Learning Component for an Optimized Context-Aware Web Service Selection based on Decision Trees for a Future Service Market,” 2013.

[6] J. Han, M. Kamber, and J. Pei, *Data mining: Concepts and techniques, third edition,* 3rd ed. Waltham, Mass: Morgan Kaufmann Publishers, 2012.

[7] J. Han and M. Kamber, *Data mining: Concepts and techniques,* 2nd ed. Amsterdam, Boston, San Francisco, CA: Elsevier; Morgan Kaufmann, 2006.

[8] S. B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” 2007.

[9] KDnuggets.com, *Data Mining, Analytics, Big Data, Data, Science.* Available: http://www.kdnuggets.com/.

[10] ____, *Libraries and Development Kits for Data Mining.* Available: http://www.kdnuggets.com/software/libraries.html (2014, Apr. 28).

[11] Albert Biffet, *Big Data Mining Tools.* Available: http://albertbiffet.com/big-data-mining-tools/ (2014, Apr. 28).

[12] KDnuggets.com, *Analytics, Data Mining, Data Science software/tools used in the past 12 months for real project.* Available: http://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-software-used.html.

[13] RapidMiner.com, *Predictive Analytics, Data Mining, Self-service Open Source, RapidMiner.* Available: http://rapidminer.com/.

[14] ____, *More Options for our Community Users.* Available: http://rapidminer.com/options-community-users/.

[15] R Project, *The R Project for Statistical Computing.* Available: http://www.r-project.org/.

[16] Markus Helbig, Simon Urbanek, Ian Fellows, *JGR - Java GUI for R.* Available: http://cran.r-project.org/web/packages/JGR/.

[17] The University of Waikato, *Weka 3: Data Mining Software in Java.* Available: http://www.cs.waikato.ac.nz/ml/weka/.

[18] Massive On-line Analysis (MOA), *MOA Overview.* Available: http://moa.cms.waikato.ac.nz/overview/ (2014, Apr. 28).

[19] Apache Software Foundation, *The Official Mahout FAQ.* Available: http://mahout.apache.org/general/faq.html (2014, Apr. 28).

[20] Eclipse, *The Eclipse Foundation open source community website.: An amazing open source community of Tools, Projects and Collaborative Working Groups.* Available: https://www.eclipse.org.

[21] Catherine Catherine, "Simulation and Measurement of Non-Functional Properties of Web Services in a Service Market to Improve User Value," 2013.
[22] Lazim Abdullah, C.W. Rabiatul Adawiyah, "Simple Additive Weighting Methods of Multi criteria Decision Making and Applications: A Decade Review," *International Journal of Information Processing and Management (IJIPM)*, pp. 39–49, 2014.
[23] Albert Biffet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, Thomas Seidl, *MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering.*
[24] W3C, *SOAP Specifications.* Available: http://www.w3.org/TR/soap/.
[25] University of WAIKATO, *Trunk weka-dev-3.7.11: weka.filters.DiscretizeFilter.* Available: https://svn.cms.waikato.ac.nz/svn/weka.
[26] IP Location, *How to find geolocation of an IP Address?* Available: http://www.iplocation.net/.
[27] Albert Biffet, Geoff Holmes, Richard Kirkby, Bernhard Pfahringer, *Data Stream Mining: A Practical Approach.*
[28] Remco R. Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald, David Scuse, *WEKA Manual for Version 3-7-11*: The University of Waikato, 2014.
[29] Christophe Salperwyck, *MOA Extension.* Available: http://chercheurs.lille.inria.fr/salperwy/index.php?id=moa-extension.
[30] Elena Ikonomovska, João Gama, Sašo Džeroski, *Learning model trees from evolving data streams.*
[31] Ross J. Quinlan, *Learning with Continuous Classes.*

A. Appendix

This section lists some additional information about the previous test data statistics, the defined simulation parameters for the current test data scenario and graphical analyses from the evaluation chapter.

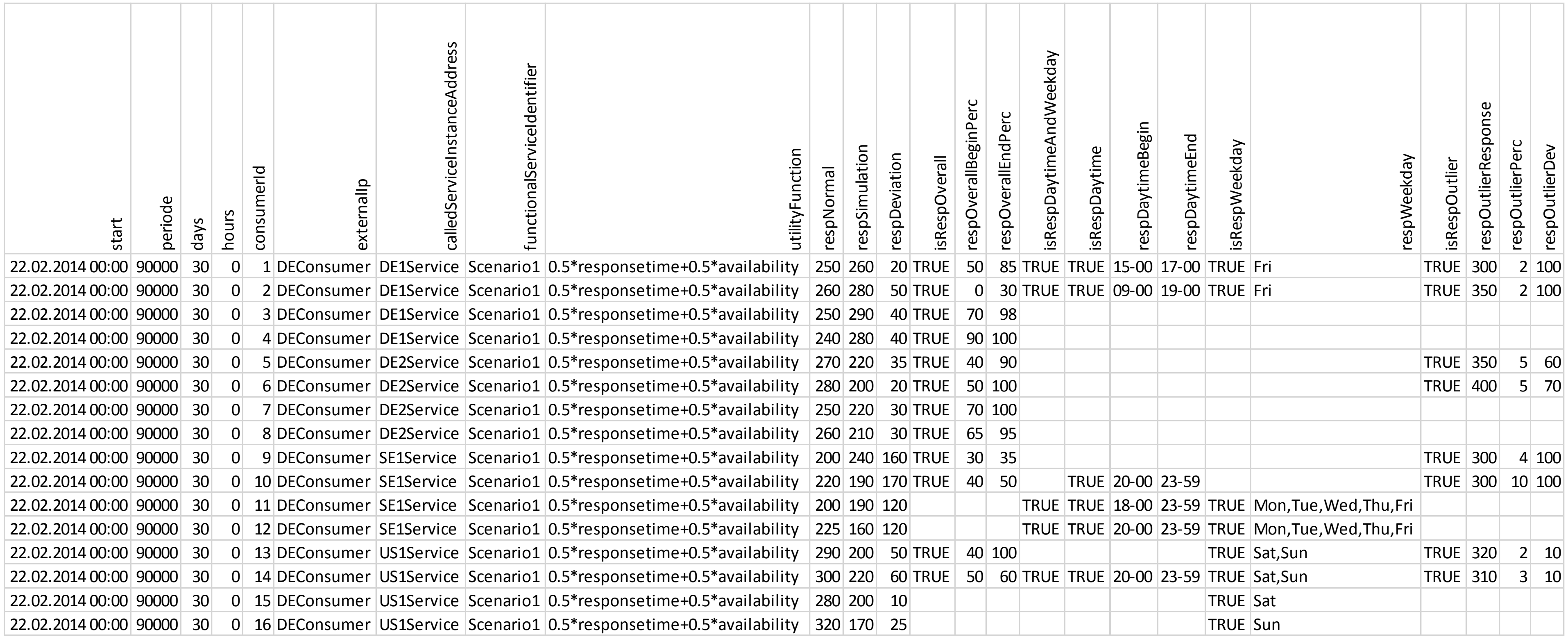
A.1Statistics of the Previous Test Data

The table below shows the service candidates with corresponding record count, response time, message size and availability. The abbreviations of the candidates are as follows: FD = File Digest, SA = Salutation, SQ = Stock Quote, WE = Weather

Service Candidate	**Records**	**Response time in ms**		**Message size in ms**		**Availability in %**
		Mean	**Deviation**	**Mean**	**Deviation**	
FD	4228	4304	3644	452510	203261	99,976
FD HSKA	2779	4650	3490	453795	200597	99,964
SA AU	13452	704	1751	305	0,9	91,711
SA DE	6856	240	921	305	0	84,072
SA UK	10042	494	1468	305	0,9	90,380
SQ LUS	6588	384	1486	154	2	98,437
SQ RestFul	6769	1528	5517	5	0	84,429
SQ RF	3440	1480	3570	5	0	83,192
SQ SMW	4052	1003	1004	4838	44	99,975
SQ YUK	8776	859	2862	12	0	99,932
WE DT	10004	636	1954	267	0	99,690
WE NDFD	6842	1096	1714	877	0	97,252
WE DE	10038	801	5782	22	0	99,930
WE WSF	8876	883	893	5	0	98,637
WE Yahoo	6645	364	515	2324	21	99,007

A.2 Simulation Parameters of the Created Test Data Scenario

The figure below shows the simulations parameters of the non-functional property response time.



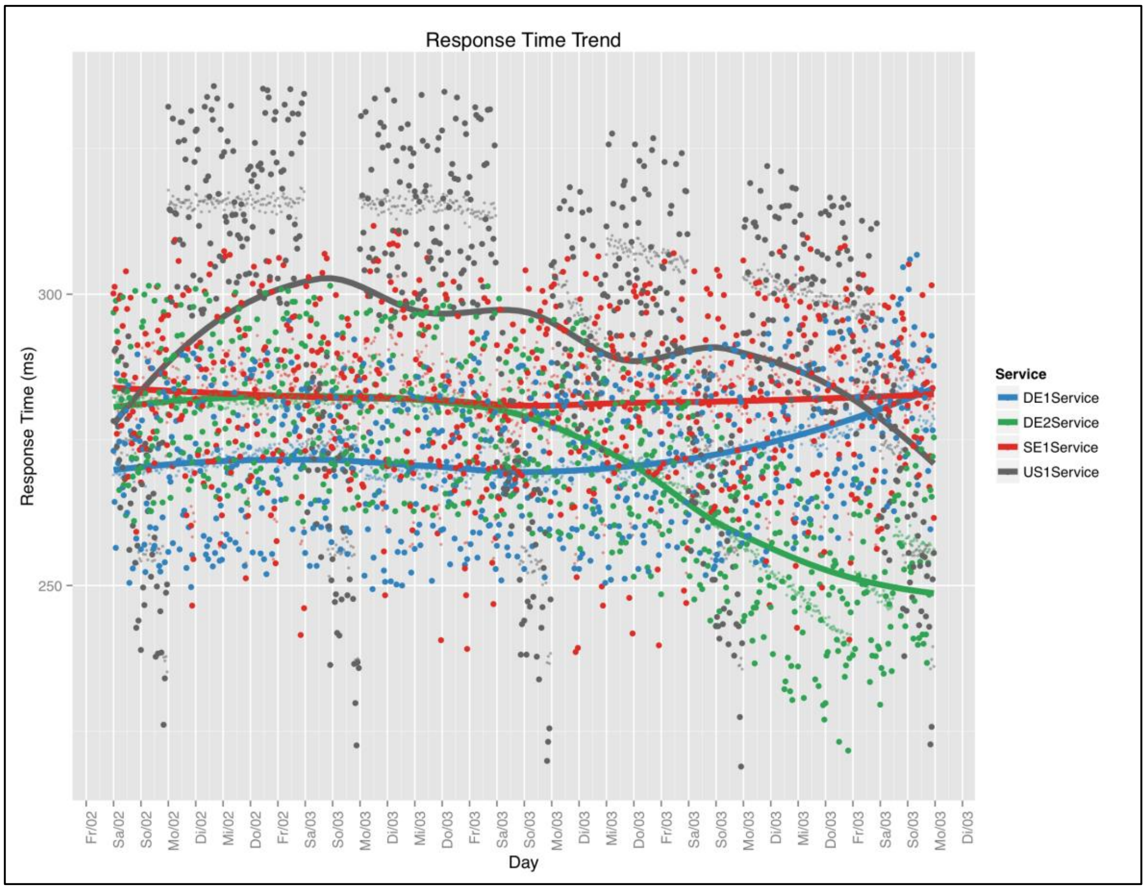
start	periode	days	hours	consumerId	externalIp	calledServiceInstanceAddress	functionalServiceIdentifier	utilityFunction	respNormal	respSimulation	respDeviation	isRespOverall	respOverallBeginPerc	respOverallEndPerc	isRespDaytimeAndWeekday	isRespDaytime	respDaytimeBegin	respDaytimeEnd	isRespWeekday	respWeekday	isRespOutlier	respOutlierResponse	respOutlierPerc	respOutlierDev
22.02.2014 00:00	90000	30	0	1	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability	250	260	20	TRUE	50	85	TRUE	TRUE	15-00	17-00	TRUE	Fri	TRUE	300	2	100
22.02.2014 00:00	90000	30	0	2	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability	260	280	50	TRUE	0	30	TRUE	TRUE	09-00	19-00	TRUE	Fri	TRUE	350	2	100
22.02.2014 00:00	90000	30	0	3	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability	250	290	40	TRUE	70	98										
22.02.2014 00:00	90000	30	0	4	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability	240	280	40	TRUE	90	100										
22.02.2014 00:00	90000	30	0	5	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability	270	220	35	TRUE	40	90							TRUE	350	5	60
22.02.2014 00:00	90000	30	0	6	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability	280	200	20	TRUE	50	100							TRUE	400	5	70
22.02.2014 00:00	90000	30	0	7	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability	250	220	30	TRUE	70	100										
22.02.2014 00:00	90000	30	0	8	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability	260	210	30	TRUE	65	95										
22.02.2014 00:00	90000	30	0	9	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability	200	240	160	TRUE	30	35							TRUE	300	4	100
22.02.2014 00:00	90000	30	0	10	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability	220	190	170	TRUE	40	50		TRUE	20-00	23-59			TRUE	300	10	100
22.02.2014 00:00	90000	30	0	11	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability	200	190	120				TRUE	TRUE	18-00	23-59	TRUE	Mon,Tue,Wed,Thu,Fri				
22.02.2014 00:00	90000	30	0	12	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability	225	160	120				TRUE	TRUE	20-00	23-59	TRUE	Mon,Tue,Wed,Thu,Fri				
22.02.2014 00:00	90000	30	0	13	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability	290	200	50	TRUE	40	100					TRUE	Sat,Sun	TRUE	320	2	10
22.02.2014 00:00	90000	30	0	14	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability	300	220	60	TRUE	50	60	TRUE	TRUE	20-00	23-59	TRUE	Sat,Sun	TRUE	310	3	10
22.02.2014 00:00	90000	30	0	15	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability	280	200	10								TRUE	Sat				
22.02.2014 00:00	90000	30	0	16	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability	320	170	25								TRUE	Sun				

The figure below shows the simulations parameters of the non-functional property availability.

start	periode	days	hours	consumerId	externalIp	calledServiceInstanceAddress	functionalServiceIdentifier	utilityFunction	isNonavailOverall	nonavailOverallPerc	nonavailOverallBeginPerc	nonavailOverallEndPerc	isNonavailDaytimeAndWeekday	isNonavailDaytime	nonavailDaytimeBegin	nonavailDaytimeEnd	isNonavailWeekday	nonavailWeekday	isNonavailOutlier	nonavailOutlierPerc
22.02.2014 00:00	90000	30	0	1	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability	TRUE	3	17	24							TRUE	3
22.02.2014 00:00	90000	30	0	2	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability	TRUE										TRUE	2
22.02.2014 00:00	90000	30	0	3	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability												
22.02.2014 00:00	90000	30	0	4	DEConsumer	DE1Service	Scenario1	0.5*responsetime+0.5*availability												
22.02.2014 00:00	90000	30	0	5	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability	TRUE	4	21	25							TRUE	1
22.02.2014 00:00	90000	30	0	6	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability											TRUE	4
22.02.2014 00:00	90000	30	0	7	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability					TRUE	TRUE	09-01	09-30	TRUE	Mon		
22.02.2014 00:00	90000	30	0	8	DEConsumer	DE2Service	Scenario1	0.5*responsetime+0.5*availability					TRUE	TRUE	09-10	09-46	TRUE	Mon		
22.02.2014 00:00	90000	30	0	9	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability											TRUE	3
22.02.2014 00:00	90000	30	0	10	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability											TRUE	4
22.02.2014 00:00	90000	30	0	11	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability											TRUE	1
22.02.2014 00:00	90000	30	0	12	DEConsumer	SE1Service	Scenario1	0.5*responsetime+0.5*availability											TRUE	2
22.02.2014 00:00	90000	30	0	13	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability	TRUE	7	30	43							TRUE	4
22.02.2014 00:00	90000	30	0	14	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability											TRUE	4
22.02.2014 00:00	90000	30	0	15	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability												
22.02.2014 00:00	90000	30	0	16	DEConsumer	US1Service	Scenario1	0.5*responsetime+0.5*availability												

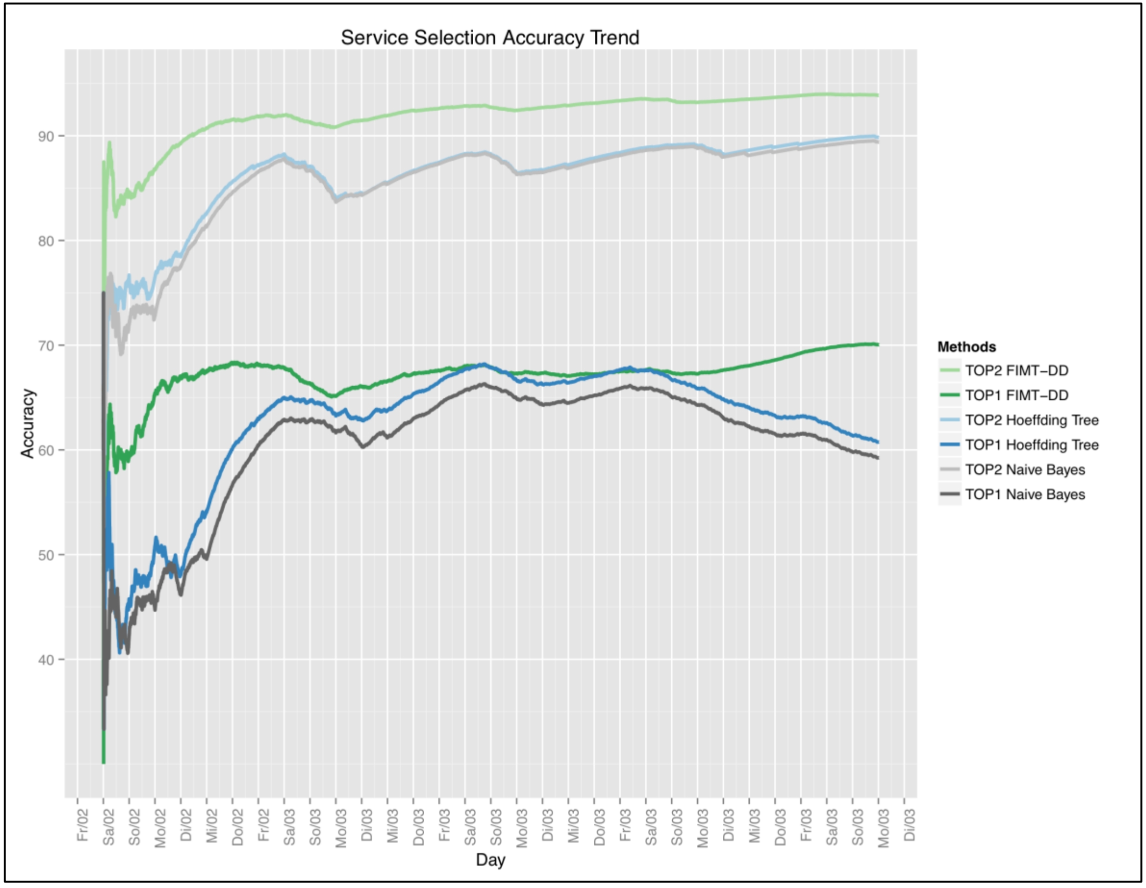
A.3 Overall Trend of the Attribute Response Time

The figure below shows the response time of the services *DE1Service*, *DE2Service*, *SE1Service* and *US1Service* in the course of time.



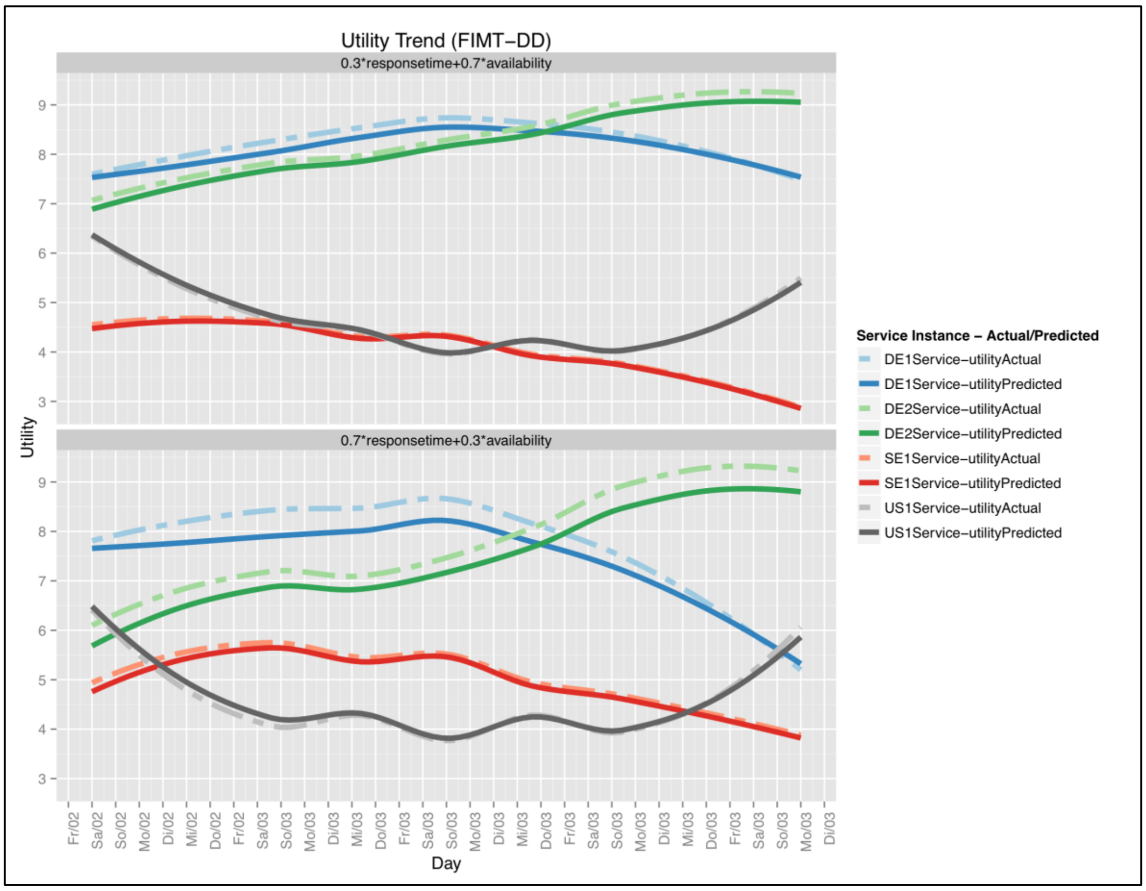
A.4Overall Trend of the Service Selection Accuracy (TOP1 and TOP2)

The figure below shows the accuracy of the service selection for the machine learning methods *FIMT-DD*, *Hoeffding Tree* and *Naïve Bayes*. The *TOP1* measure is the percentage of correctly determined best-fit services to the actual best-fit service, whereas the *TOP2* selection shows the percentage of the correctly determined best-fit services within the actual two best-fit services.



A.5 Overall Trend of the Actual and Predicted Utilities of the FIMT-DD algorithm with Different Utility Functions

The figure below shows the overall utility trend of two different utility functions with either a higher preference for the attribute response time (figure below) or availability (figure above).


Utility Trend (FIMT–DD)
0.3*responsetime+0.7*availability
0.7*responsetime+0.3*availability
Utility
Day
Fr/02
Sa/02
So/02
Mo/02
Di/02
Mi/02
Do/02
Fr/02
Sa/03
So/03
Mo/03
Di/03
Mi/03
Do/03
Fr/03
Sa/03
So/03
Mo/03
Di/03
Mi/03
Do/03
Fr/03
Sa/03
So/03
Mo/03
Di/03
Mi/03
Do/03
Fr/03
Sa/03
So/03
Mo/03
Di/03
Service Instance – Actual/Predicted
DE1Service–utilityActual
DE1Service–utilityPredicted
DE2Service–utilityActual
DE2Service–utilityPredicted
SE1Service–utilityActual
SE1Service–utilityPredicted
US1Service–utilityActual
US1Service–utilityPredicted