Cost Model for the SAP Gamification Platform

Author: Svenja Brunstein

MASTER THESIS

to obtain the academic degree of "Master of Science (MSc)"
in the Master Degree Program
Business Informatics

Department of Business Informatics – Data & Knowledge Engineering
Johannes Kepler University Linz
Altenberger Str. 69, 4040 Linz, Austria

Submission: June 2013
Supervisor: o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl, JKU Linz
Co-Supervisor: Dr. Stefan Berger, JKU Linz

Declaration

I hereby declare under oath that the submitted Master's thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited.
The submitted document here present is identical to the electronically submitted text document.

Dresden, June 19, 2013

Svenja Brunstein

Acknowledgment

This master's thesis was supported and supervised at SAP by Philipp Herzig from SAP Mobile Research Dresden.

Abstract

In this master's thesis, a generic platform to support gamification of applications is the object of study. This Gamification Platform was recently invented and implements a new type of architecture. The performance of the platform is unknown since no prior knowledge exists for this complex type of architecture. Conducted experiments have shown that the response time and throughput are varying extremely, which has to be analyzed and described more precisely with a cost model.

To support decision-making and be able to calculate costs of a gamification concept beforehand, performance and throughput of the Gamification Platform should be known prior to its implementation. Additionally, it has to be decided whether the applications and the platform should prefer to communicate in synchronous or asynchronous mode. The synchronous mode consumes less memory, whereas the asynchronous mode operates faster. A cost model allows the pre-calculation of performance and throughput for the generic Gamification Platform.

First, a queueing network model is built based on the architecture of the platform, which estimates the performance. Subsequently, cost factors are researched and measured in three categories: Users, rules, and infrastructure. The influence of each significant cost factor on performance is modeled with polynomials.

The accuracy of the presented cost model in synchronous mode has a Mean Magnitude of Relative Error of 0.15, the asynchronous cost model's Mean Magnitude of Relative Error is 0.115. For a real use case the Magnitude of Relative Error are 0.436 in synchronous mode and 0.148 in asynchronous mode. Errors due to exclusion of factors and simplifying assumptions will need to be overcome in future research.

Even though the Gamification Platform is generic, the presented cost model only covers the most relevant parts. However, even this cost model is already complex, and, more importantly, delivers accurate estimations of the platform's performance. Further need for research is emphasized to include more cost factors as well as their effects on each other, and to support the automation of model usage.

Contents

List of Figures

List of Tables

List of Abbreviations

AIC	Akaike Information Criterion
ANOVA	Analysis of Variance
BEP	Business Entity Provider
BIC	Bayesian Information Criterion
CART	Classification and Regression Trees
CEP	Complex Event Processor
COCOMO	Constructive Cost Model
CPU	Central Processing Unit
CV	Cross-Validation
DBMS	Database Management System
e.g.	exempli gratia (for example)
ECA	Event Condition Action
ERP	Enterprise Resource Planning
et al.	et alii (and others)
FCFS	First Come, First Served
i.e.	id est (that is)
JMS	Java Message Service
JMT	Java Modelling Tools
LAN	Local Area Network
LHS	Left-Hand Side
LJS	Lean Java Server
MB1	Message Broker I
MB2	Message Broker II
MdMRE	Median Magnitude of Relative Errors
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
OLS	Ordinary Least Squares
OSR	Optimized Set Reduction
QoS	Quality of Service
RAM	Random-Access Memory
RHS	Ride-Hand Side
RPC	Remote Procedure Call
SLA	Service-Level Agreement
TP	Transaction Processing
tpm	Transactions per Minute

TPS	Transactions Per Second
WAN	Wide Area Network
WME	Working Memory Element

List of Listings

Chapter 1

Introduction

The introduction gives an overview on the motivation for the thesis, describes the generic *Gamification Platform*, which is the basis for the cost model, defines the research questions, and presents the approach from theory and data to the cost model. Additionally, the scope is defined, the concrete and real example used within the thesis is introduced and the outline is explained.

1.1 Motivation

Gamification is defined as the use of game elements and mechanics in non-gaming applications, which are introduced to improve user experience and user commitment in the "gamified" applications [Deterding et al., 2011]. Herzig et al. [2012b] showed that gamification of *Enterprise Resource Planning* (ERP) software can improve factors such as software enjoyment and perceived ease of use. Moreover, enterprise gamification can be used to encourage desired user behavior [Thom et al., 2012] and to increase commitment with applications or services [Zichermann and Cunningham, 2011].

As many companies plan to introduce gamification in their applications in the near future, the market size of gamification is expected to reach $2.8 billion by 2016 [M2 Research, 2012]. Existing gamification solutions only implement subsets of possible features and are not as flexible as required by enterprises [Herzig et al., 2012a]. Hence, a generic *Gamification Platform* was invented and implemented lately to overcome the restrictions given by other solutions. The platform aims at introducing gamification in existing or new business applications as easily and economically as possible. Game mechanics such as rules, missions, and rewards can be defined for each application individually [Herzig et al., 2012a].

One requirement of gamification in general is that feedback has to be provided immediately based on the users' actions and interactions. However, as this platform will be used by several applications, the response time becomes critical with a higher amount of events generated by various applications and users. It has to be ensured that in any case the response time is below a determined boundary (e.g., 400ms for collaborative scenarios [Kurose and Ross, 2013]).

Using a generic Gamification Platform for several applications according to the client-server-model is a new type of gamification architecture. Currently, no scientific approach or cost model exists for this architecture to predict the run-time costs of a specific scenario within the platform. Hence, for the first version of the platform, analyses were conducted in order to find the best runtime configuration for one application. However, this is a very cost-intensive process and cannot be done for every new application. For future implementations, a reusable, universal cost model has to support and facilitate the decision prior to deployment and runtime.

1.2 Gamification Platform

In this subsection the purpose and architecture of the generic Gamification Platform are explained.

1.2.1 Purpose

The generic platform is developed to provide gamification as a service. Due to the generic implementation, many common game mechanics are made available, which can be used by applications for their specific purposes. These mechanics include, for example,

- Badges
- Leader Boards
- Level
- Missions
- Points
- and Teams.

Each application can determine its own rule set consisting of several rules to define its gamification concept. Rules make use of *events*, *conditions* and *actions* (ECA) [Paton and Díaz, 1999], or only of *conditions* and *actions* as in production rules [Davis et al., 1977].

Events are sent to the Gamification Platform to inform the platform about an action (caused by a user) in the application.

Conditions determine whether the action has to be triggered based on received event(s) or context information.

Actions define the task which the Gamification Platform will perform upon every time an event matching the conditions is detected.

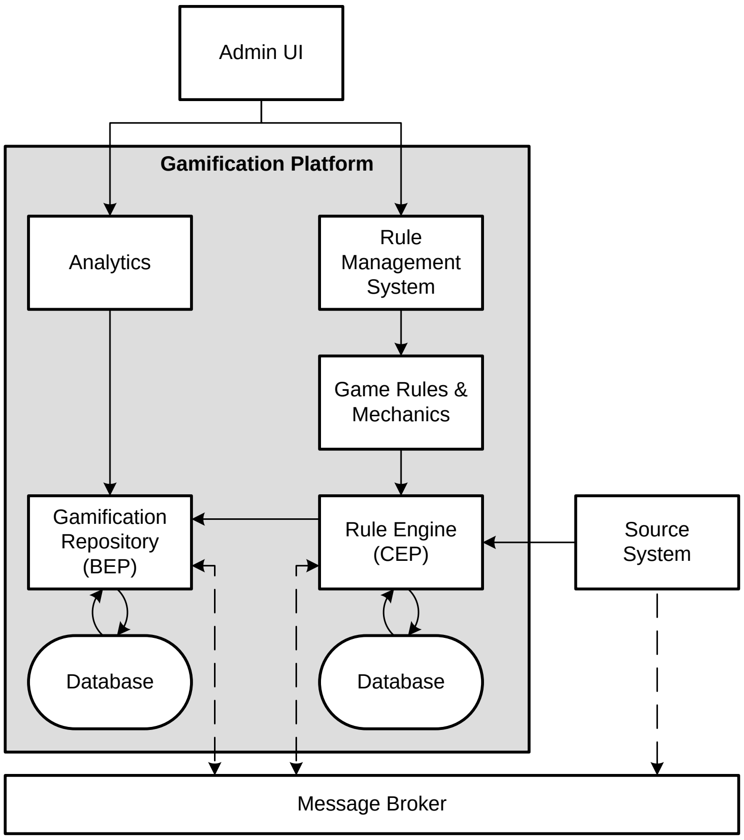


Figure 1.1: Gamification Platform Architecture [Herzig et al., 2012a]

Each application can be loosely coupled to the Gamification Platform by using a specific rule set and therewith easily introduce its own gamification concept. As the content of the rule set is completely flexible according to the requirements of the applications gamification concept, any application can introduce gamification by applying the provided gamification mechanics for its own purpose. The loosely coupled approach enables reusability and fast deployment as well as many possible fields of application.

1.2.2 Architecture

The architecture of the Gamification Platform is depicted in Fig. 1.1. The platform consists of two web applications written in Java and currently deployed on a *Lean Java Server* (LJS)[1]. On the one hand, a *Gamification Repository* encapsulates game mechanics as well as data of players and their progress. This gamification repository is also called *Business Entity Provider* (BEP). On the other hand, the *Rule Engine* contains a context-aware *Complex Event Processor* (CEP). *Drools Expert* is used

[1]LJS is an Apache Tomcat server including several additional features.

as implementation for the CEP, which, in turn, is based on the *RETE algorithm* (Sect. 2.3) for efficient execution of many rules with many objects. Context information is stored in the BEP. Both CEP and BEP persist their data to databases. *Analytics* of gamification data is performed on data of the BEP. To manage rules, a user interface (*Admin UI*) gives access to modify and create rules in a *Rule Management System* directly. These rules are then combined with game mechanics provided by the Gamification Platform (*Game Rules & Mechanics*) and made available in the CEP.

For communication between the components BEP, CEP, and source system either synchronous *Remote Procedure Calls* (RPCs) or an asynchronous message broker, e.g., *Java Message Service* (JMS), is used. The asynchronous communication is depicted using dashed lines in Fig. 1.1. For synchronous communication, on the one hand, latencies are likely to occur, because context data has to be fetched from the BEP every time it is needed. For asynchronous communication, on the other hand, context data from the BEP has to be duplicated and maintained twice to ensure consistency. Since both cases have drawbacks, a decision for one of the cases (or a combination, e.g., context updates triggered by certain events) has to be made for every particular application. In this thesis, both *modes*, synchronous and asynchronous, are researched and cost models are created for both types of communication.

CEP, BEP, and, in asynchronous mode also the Message Broker are from now on referred to as components and are analyzed more precisely in the thesis. Rule Management System, Admin UI and Analytics are not considered any further, because these do not influence the response time of the Gamification Platform and are only used to maintain and analyze content.

1.3 Research Questions

Consequently, research questions to be answered within the master's thesis can be defined as follows:

- Which cost factors in the cost categories users, rules, and infrastructure influence the performance of the Gamification Platform?
- How much is the performance of each component of the Gamification Platform being influenced by these cost factors?
- How can the response time of the Gamification Platform be predicted using a cost model, which is based on the identified cost factors?

1.4 Approach

To predict response time and throughput of the Gamification Platform based on a cost model, several steps need to be taken. Hence, Sect. 2.1 presents the general process of modeling with a particular focus on cost and performance modeling from a theoretical point of view.

The cost model is designed as a two-layer model. A queueing network model in the upper layer predicts the total response time and consists of several resources, which are defined by the architecture of the Gamification Platform. The resource parameters (e.g., service times, interarrival times) defining the queueing network more precisely have to be gained either from the lower layer models or from forecasted data. Lower layer models are polynomial models for each cost factor, and forecasted data is defined by the specific usage scenario and gamification concept.

The first step is to determine the relevant parts of the architecture, which affect the performance (Chap. 3), and to transform them into queueing network models. The theoretical background of queueing networks is explained in Sect. 2.2. As the RETE algorithm is used in the CEP component and is expected to considerably influence the response time of the Gamification Platform, the theoretical background of the RETE algorithm is summarized in Sect. 2.3.

Secondly, all cost factors influencing the response time have to be identified (Chap. 4). In Subsect. 2.1.1 it is explained how this can be done theoretically based on expert knowledge and on previous cost and performance model research. For all identified factors it has to be decided whether their influence on system performance is significant or negligible. Furthermore, some cost factors are excluded due to the restricted time frame of the thesis. For every cost factor to be considered, its significance is determined by measuring the response time at two factor levels differentiating in several orders of magnitude and investigating the change of the response time. Subsequently, cost factors which are identified to significantly influence the response time will be included in the cost model and are examined more closely.

Having identified the cost factors and components as well as the queueing network structure, the service times of the components in the queueing network need to be measured (Chap. 5). Measurements of service times are needed for the calibration of the queueing networks to specific scenarios. More precisely, service times have to be related to the cost factors, so that it is possible to predict the service times for other configurations from the polynomial models based on measured data. The statistical background for fitting and selecting polynomials is explained in Sect. 2.4.

The measurements are carried out as experiments for each cost factor, each component and each mode (synchronous and asynchronous) individually. For every experiment a polynomial model is constructed to describe the influence of the cost factor on the service time of a component. Additionally, distributions of service times and interarrival times have to be measured, as these are also necessary input values for the queueing network. The theoretical backgrounds of the test design and measurements are both explained in Subsubsect. 2.1.1.

After the polynomial models are built, the usage of the polynomial and queueing network models, and how required inputs can be gained from a use case are explained in Chap. 6. In the final step, the cost model is evaluated by simulating the real use case introduced in Sect. 1.6. Furthermore, the cost model is evaluated at abstract level with the modeling criteria defined in Subsubsect. 2.1.3.

1.5 Scope

The scope of the thesis is to provide a preliminary, slightly simplified cost model for the Gamification Platform. This cost model is not expected to provide a complete and generic solution, but rather should provide an approximate estimation of system response times. The thesis presents the approach used to create the first version of the cost model. The platform is generic and supports a large variety of possibilities through the use of the CEP. However, the cost model is restricted in its usage to simple cases, i.e., not covering all possible CEP features, but the basic and most frequently applied features.

As the average response time of complex event processing is a large area of study itself, only the fundamental features are covered in the thesis. The influence of alpha node equality checks and beta node joins are investigated. Many of the complex event features, however, are not considered within the thesis, namely temporal reasoning, and sliding windows.

Additionally, queueing networks as modeling technique do not allow to cover all types of scenarios accurately. For example, changing or context dependent probabilities cannot be modeled in queueing networks. This means that rules which become invalid or valid based on a specific event or time can only be modeled using a simplified and potentially inaccurate solution. For the same reason it is impossible to simulate loops or a specific routing of the requests/events in the system, which are necessary for rules with cyclic references.

1.6 Concrete Example

To illustrate different parts and functions of the system in the thesis a comprehensive, real example is chosen. The described application heads towards integration of gamification to ensure a higher and long-lasting commitment of the users. Colleagues are matched for joint lunches by the application. Each user may enter time slots in the future when and where he is available either for lunch or coffee meetings. The application automatically combines these different times slots and suggests meetings of several partners by sending invitations to all participants of this meeting. After a participant received the invitation, he can accept or decline the proposed meeting. Once the meeting took place, the attendance at the meeting has to be confirmed by every participant. Moreover, users can use tags to specify interests, add buddies and notes.

The Gamification Platform will be used to assign missions to the users, e.g., *add 10 tags*. After completing a mission, the user usually is rewarded with a badge, saying that he completed the mission successfully. In many cases, the user also unlocks further missions when completing one. The small examples used within the thesis are combined into a larger use case for the validation of the cost model in the end.

1.7 Outline

This master's thesis is structured as follows.

Chapter 2 presents the theory on which the thesis is based. Modeling (Sect. 2.1), queueing networks (Sect. 2.2), RETE (Sect. 2.3), and statistics used to create the polynomial models and and to evaluate cost factors (Sect. 2.4) are explained from a theoretical perspective.

In Chap. 3 the systems' components are analyzed and queueing networks representing the system are discussed.

Chapter 4 focuses on the cost factors of the cost model. Cost factors are divided into three categories: Users (Sect. 4.1), rules (Sect. 4.2), and infrastructure (Sect. 4.3). In each of the categories, several cost factors are defined, described, and tested with regard to their significance.

Experiments performed to determine the influence of the cost factors on the service times are described in Chap. 5.

Chapter 6 presents the results of the experiments, and the development of the performance and cost model from the measured data.

The validation of performance predicted by the cost model with a use case is conducted in Chap. 7.

The thesis is concluded with a summary (Chap. 8) and an outlook (Chap. 9).

Chapter 2

Theoretical Foundations

In this chapter, the underlying theory for this master's thesis is presented. Additionally, all specific terms needed for the understanding of the thesis are defined. The chapter is structured as follows. First, theory on modeling in general with focus on cost and performance modeling is provided. The next section focuses on queueing theory and queueing networks, which is a common modeling technique for performance analysis and will be used for the cost model in this thesis. An introduction to the RETE algorithm, which is used as implementation of the rule engine in the Gamification Platform and is expected to be a major influence factor on the performance of the platform, follows in the second last section. The last section presents statistical methods applied within the thesis.

2.1 Modeling

Similar to the estimation of costs in software engineering projects, where badly estimated projects might result in poor resource allocations [Briand and Wieczorek, 2002], a bad estimation of complexity or computational effort in information systems usually leads to various drawbacks. For example, if a web service is configured to handle peak loads well, it is oversized and wasteful when normal load occurs. On the contrary, a web service configured to handle the mean load well might not perform as expected if peak loads occur [Menasce and Almeida, 2001].

The Quality of Service (QoS) or performance delivered by a system closely correlates with the cost of the infrastructure needed to provide the service. For example, if the number of users increases, more resources (e.g., servers, communication links, storage devices) are needed to provide the same QoS to all users. Moreover, if the performance modeling and planning is done inaccurately, it might lead to unexpected unavailability of the system. To avoid such a behavior, the system's performance needs to be observed and planned in a proactive manner [Menasce and Almeida, 2001], for example, with a performance model. A performance or cost model helps in estimating the performance and cost of a system.

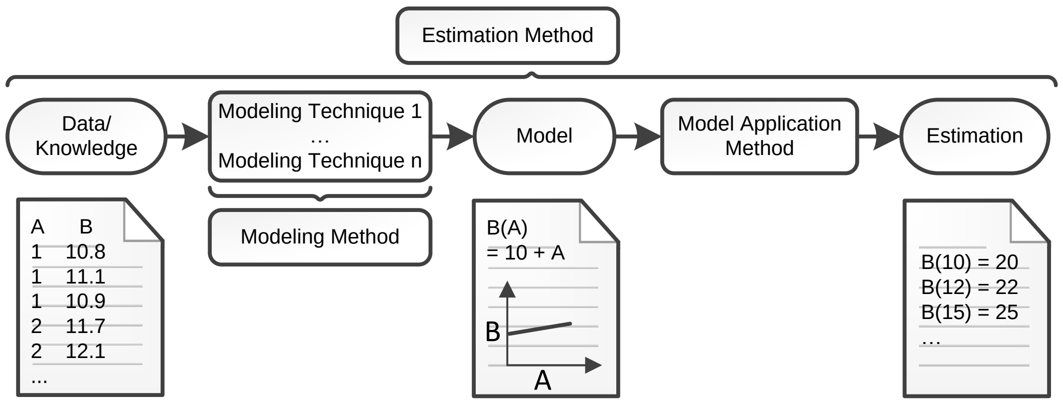


Figure 2.1: Modeling Terminology [Briand and Wieczorek, 2002]

The approach to create an estimation with a model in general is depicted in Fig. 2.1. One or more *modeling techniques* (Subsect. 2.1.2) are used on *data* (Subsect. 2.1.1) to create a *model* (Subsect. 2.1.3). Examples for modeling techniques are linear regression and ordinary least squares. As more than one modeling technique can be used, the combination of used modeling techniques is called *modeling method*. Afterwards, the *model application method* (Subsect. 2.1.4) is applied to the model in order to get an *estimation* (Subsect. 2.1.5). The whole process from data to an estimation is called *estimation method*.

2.1.1 Data/Knowledge

Before a performance or cost model can be built, it is important to comprehend the system which will be analyzed [Jain, 1991]. Therefore, cost factors need to be established, the test design has to be defined and measurements have to be taken.

Cost Factors

Cost factors and values, which shall be estimated (in Fig. 2.1: Factor A and value B), have to be determined before taking measurements. Cost factors or parameters in soft- and hardware can be divided into four categories (Table 2.1). For a comprehensive model, cost factors of each category need to be considered. Defining the possible cost factors separately before composing a holistic cost model is a bottom up approach of building a model. Intuitively appealing cost factors and their relationships have to be verified by research. Previous research has identified several distinct elements, listed in Table 2.2, as significant influence factors on performance of a system and, hence, should be considered for every cost model.

In order to find the most important cost factors besides the obvious ones known from research, expert opinions should be obtained to find the particular cost factors for

Parameter Category	Description	Examples
System Parameters	system characteristics that affect the performance	max. number of threads in database management system, max. number of connections on server
Resource Parameters	resource characteristics influencing the performance	disk seek time, CPU speed rating
Workload Intensity Parameters	load placed on the system	number of requests/s, number of clients
Workload Service Demand Parameters	service time required by each basic component at each resource	CPU time of transactions in database, transmission time between components

Table 2.1: Parameter Categories [Menasce and Almeida, 2001]

Element	Description
Client platform	Quantity and type
Server platform	Quantity, type, configuration, and function
Middleware	Type (e.g., TP monitors)
DBMS	Type
Services/applications	Main Web services and applications supported
Network connectivity	Network connectivity diagram showing all LANs, WANs, network technologies, routers, servers, load balancers, firewalls, and number of clients per LAN segment
Network protocols	List of protocols used
Usage patterns	Peak periods (e.g., hour of day, day of week, week of month, month of year)
Service-level agreements	Existing SLAs per Web service. When formal SLAs are absent, industry standards can be used
LAN management and support	LAN management support structure, size, expertise, and responsiveness to users
Procurement procedures	Elements of the procurement process, justification mechanisms, and duration of the procurement cycle

Table 2.2: Elements with Influence on Performance [Menasce and Almeida, 2001]

the analyzed system. For example, expert opinions were used to create COCOMO, which is one of the most utilized models for cost estimation of software engineering projects [Briand and Wieczorek, 2002]. Experts, due to their good insight into the system and its relevant components, are able to give valuable hints on possible cost factors. A complete list of cost factors is crucial, because if only one important factor is not considered, the results may render useless [Jain, 1991].

Type and value of estimation metrics (e.g., response time in ms, throughput in events per second) have to be defined beforehand [Jain, 1991]. Subsequently, each cost factor has to be analyzed separately before constructing a predictive cost model comprising all cost factors [Boehm, 1984]. Significant relationships between cost factors should be included in the cost model as well [Chrysler, 1978].

Test Design

Upon identification of all relevant influence factors the next step in building the cost model involves choosing the subset of cost factors to be studied as well as the estimation method: analytical modeling, simulation, or measurements in the real system. Afterwards, the workload has to be defined in terms of number of requests and probabilities for each different request type. For the purpose of including all possible real situations, a performance test should include several different test scenarios. One test should be performed under normal conditions, one under extreme conditions (stress testing) and one test should simulate peak loads with peaks several orders of magnitude larger than the average (spike test) [Menasce and Almeida, 2001].

According to Gray [1993], throughput of a system should not only be investigated in steady state, but also during ramp-up. In his definition, the state is steady when the system performs normal and as expected, and ramp-up is the phase between starting the system and reaching the steady state.

To test the influence of k factors with l levels on system's performance, a full factorial design has to be used. The number of experiments in a full factorial design, where all possible configurations are combined, is defined by Jain [1991]:

$$\prod_{i=1}^{k} l_i \tag{2.1}$$

In most cases the effort of this design is too high, e.g., 5 factors with 3 levels each would lead to $3^5 = 243$ tests! If several combinations can be excluded, as cost factors might be known to be independent from each other, a l^{k-p} fractional factorial design can be used. l, again, is the number of levels and p represents the size of the fraction of the full factorial used [Box and Hunter, 2000]. This test design significantly reduces the number of tests to be conducted, while having the drawback to remove the possibility to determine all dependencies between cost factors. After the tests have been accomplished, measured data has to be analyzed, interpreted and used to build a cost model with modeling techniques presented in Subsect. 2.1.2 [Jain, 1991].

In this thesis, data to build the polynomial models will be collected using the real system. A full factorial design cannot be accomplished since the number of relevant cost factors and factor levels is too high to conduct all tests within the time frame of the thesis. Because of this, every cost factor is researched independently from the others. Additionally, all tests are conducted under normal conditions, but the queueing network model can also simulate extreme or peak loads.

Measurements

To derive a predictive performance model, real training data for the model has to be collected. In the case of performance modeling, several samples are measured. In Fig. 2.1, three measurements of B were taken for $A = 1$ (B is measured at 10.8, 11.1, and 10.9), at least two for $A = 2$ (B is measured at 11.7 and 12.1) and so on.

As a consequence, if the performance is measured m times, m different measurements will be found which are most likely to be varying. It is impossible to find the actual mean value with a finite number of measurements, but a probabilistic statement can be defined as follows.

$$Pr[c_1 \leq \mu \leq c_2] = 1 - \alpha \tag{2.2}$$

μ is the population mean, (c_1, c_2) is the confidence interval for the mean estimation, and $(1 - \alpha)$ the confidence coefficient. This formula states that the probability of the population mean being inside the confidence interval, i.e., between c_1 and c_2, can only be ensured with a certain confidence [Jain, 1991].

With the help of the *Central Limit Theorem*, under the assumption that the observations of the sample are independent and are drawn from the same distribution, the confidence interval is defined as

$$(c_1, c_2) = (\overline{x} - z_{1-\alpha/2} \times s/\sqrt{n}, \overline{x} + z_{1-\alpha/2} \times s/\sqrt{n}) \tag{2.3}$$

In this term, $\overline{x}$ is defined as the sample mean, s is the sample standard deviation, n the sample size, and $z_{1-\alpha/2}$ the $(1 - \alpha)$-quantile of a normal distribution [Menasce and Almeida, 2001; Jain, 1991]. By using this formula, it is possible to extrapolate an estimation from a sample of measurements, because it is impossible to conduct an infinite number of measurements. The smaller the confidence interval is chosen, the higher is the precision of estimated values [Jain, 1991].

Moreover, it is possible to calculate the necessary number of measurements to be conducted to reach a confidence level of $100(1 - \alpha)\%$ with a maximum variability of $r \times \overline{x}$ [Menasce and Almeida, 2001]:

$$n \geq (\frac{100 \times z_{1-\alpha/2} \times s}{r \times \overline{x}})^2 \tag{2.4}$$

2.1.2 Modeling Techniques

To infer a cost model from data, different modeling techniques can be used. First, if a model is developed by measuring and analyzing data, a data driven modeling technique is chosen. Second, composite modeling techniques are also based on measured data, but a prediction for new cases is made based only on a specific set of old cases, in particular, on the most similar case(s).

Alternatively, it is also possible to predict the behavior with a non-model based technique, e.g., expert judgment. Jorgensen and Shepperd [2007] criticize that this method is often used in companies, but only little research has been conducted on expert judgment. As these methods do not provide a model, but rather use forecasts from experts who are estimating based on their knowledge and experience, it is not considered further in the thesis. It should only be mentioned that expert judgment or other non-model based methods can also be combined with estimations based on data.

Data Driven Modeling Techniques

Data driven models are directly derived from data by performing data analysis [Briand and Wieczorek, 2002]. When “a considerable amount of data describing this problem is available” [Solomatine et al., 2008, p. 27] and also “no considerable changes to the modeled system during the period covered by the model” [Solomatine et al., 2008, p. 27] are expected, a data driven model can be used. Data driven modeling techniques are, e.g., (Stepwise) Analysis of Variance (ANOVA), Classification and Regression Trees (CART), Ordinary Least Squares (OLS), and Polynomial Regression. With ANOVA, the most significant, independent factors can be found [Cohen, 1968]. It helps in deciding which cost factors have the highest impact on the performance. To determine the impact of unknown, independent parameters on dependent variables, OLS can be applied [Hayashi, 2000]. The data-driven technique polynomial regression (Subsect. 2.4.2) is used in Chap. 2.1.2 to determine the impact of each cost factor in the cost model from the measured data.

Data driven models can be divided further into using a parametric or non-parametric modeling method. With a parametric method the model parameters are defined a priori and distributions of the data are assumed. Non-parametric methods derive the model from data and do not make assumptions on the amount and deviation of parameters beforehand [Briand and Wieczorek, 2002; Whitley and Ball, 2002]. For example, ANOVA is a parametric modeling method, and CART is a non-parametric modeling method.

Briand et al. [1999] studied a number of modeling techniques. They conclude that the quality and adequacy of data collection is much more important for the cost model’s accuracy than the modeling technique used. In other words, the usage of different modeling techniques might lead to the same or a very similar result.

Composite Modeling Methods

Composite modeling methods are, for example, Analogy, Optimized Set Reduction (OSR), and COBRA. The latter two are especially designed for usage in software engineering cost estimation. For a project cost estimation OSR takes into account a subset of most similar projects [Briand and Wieczorek, 2002]. Mean values from this subset might be more accurate for an estimation of the project than other methods, since individual characteristics of projects are shared in similar projects. Of course, a knowledge base of finished projects has to exist beforehand, so that the OSR method may access data of similar projects. Analogy basically uses the same approach as OSR, but it is not limited to software engineering cost estimations [Skousen et al., 2002]. For these modeling methods a significant knowledge base has to exist beforehand. Since no knowledge base exists for the Gamification Platform, the composite modeling methods are not considered any further.

2.1.3 Cost Model

A cost model is the result of applying several modeling techniques on the data and knowledge. This cost model can then predict or estimate future cases. For example, in Fig. 2.1, the model consists of the simple formula $B(A) = 10 + A$. For each value of A, B can be estimated. Similarly, a model can be built for more than one factor.

To ensure that the cost model itself or cost factors are not superfluous, the estimations of the cost model should be compared to the estimations of a simple "straw man" model [Menzies et al., 2006]. In the example, a comparison of the accuracy of $B(A) = 10 + A$ and a simpler model, e.g., $B = 11.5$ should be conducted.

In this section, cost and performance models are discussed in detail. Moreover, model criteria to determine the quality of a model are presented. Finally, it is explained how a model can be validated.

Cost and Performance Models

Response time is one of the main values to be estimated with a performance model. Two different definitions for response time are depicted in Fig. 2.2. The first definition defines response time as the time between sending the request and sending the response. The second definition defines response time as the time frame started by the user sending the request and ended by the systems response being completely received by the user. In addition, *reaction time* is the time between user finishing the request and start of the execution at the system. *Think time* is defined as the time when the system completes a request for a specific user and this user starting the next request.

Throughout the thesis, the second definition of response time is used. The difference between the first and the second definition of response time is the time to transfer the response. The transfer time depends on the distance between user and system as well as on the size of the response. In the thesis, the distance between user and system is assumed to remain constant over time and is kept at a minimum. No other applications use the connection between user and system, which otherwise could also influence the transfer time. The size of the response is also fixed at this point, so that the difference between first and second definition is expected to be a small, fixed value with very little variance.

Throughput is a measure for completed requests per unit of time, which is usually measured in transactions per second (TPS) in transactions processing [Jain, 1991]. This measure will be estimated in addition to the response time.

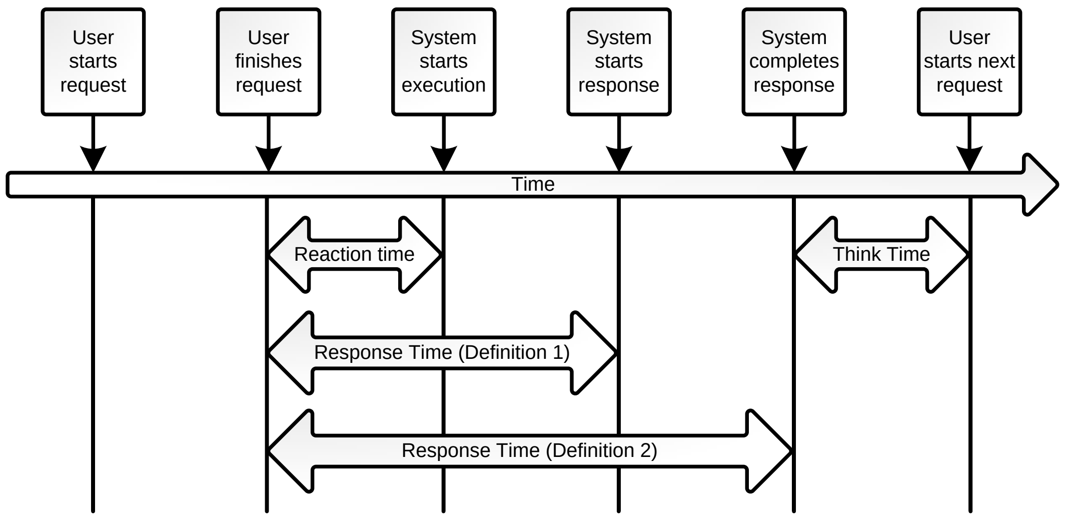


Figure 2.2: Definition of Response Time [Jain, 1991]

Model Type	**Description**
Workload Model	*"captures the resource demands and workload intensity characteristics of the load brought to the system by the different types of transactions and requests"* [Menasce and Almeida, 2001, p.176], the collection of measured values should be combined on basic component level [Menasce and Almeida, 2001].
Performance Model	estimates response times, throughput (requests/s), workload, and resource queue lengths based on the description of the system.
Cost Model	converts the performance requirements into costs for software, hardware, third-party services, and staffing and estimates overall costs.

Table 2.3: Model Types [Menasce and Almeida, 2001]

Menasce and Almeida [2001] employ three models for workload, performance and cost modeling, which are described in detail in Table 2.3. A workload model describes the load placed on a system in order to measure the performance. If the real system is available, a workload model can be built using measurements. If no data can be measured beforehand, the workload has to be modeled using literature and knowledge [Feitelson, 2002]. Performance models predict the performance of new systems by using workload models to estimate demands on the systems [Menasce and Gomaa, 2000]. Cost models are used to estimate costs, e.g., COCOMO estimates the costs of software engineering projects beforehand based on several cost factors [Briand and Wieczorek, 2002]. Alternatively, a metric can be provided to convert the performance requirements into costs.

The TCP-C benchmark, for example, "provides a dollar per tpm ($/tpm) metric, which indicates how much needs to be spent per unit of throughput measured in

Model and Estimate Criteria	**Estimation Method Criteria**	**Application Criteria**
• Quality of model and estimate • Inputs required • Completeness • Type of estimates • Calibration • Interpretability	• Assumptions • Repeatability • Complexity • Automation (Modeling) • Transparency	• Application Coverage • Generalizability • Comprehensiveness • Availability of estimates • Automation (Method Usage)

Table 2.4: Model Criteria [Briand and Wieczorek, 2002]

transactions per minute (tpm)" [Menasce and Almeida, 2001, pp. 117-118]. This benchmark is an example of how to derive a cost model directly from the performance model.

In this thesis, the performance is modeled by using a queueing network. The queueing network theory is further explained in Sect. 2.2. As the input needed for applying the queueing network model is dependent on cost factors, which are in turn modeled using polynomial models, the comprehensive model combining both is called cost model. The workload is modeled based on experience and, wherever available, real data.

Model Criteria

The quality of a model can be described and evaluated using model criteria specified by Briand and Wieczorek [2002]. Each cost model should be verified with the criteria. The cost model developed in the thesis is evaluated with these criteria in Sect. 7.2.

There are several criteria for models, which can be divided into three categories: *Model and Estimate Criteria*, *Estimation Method Criteria* and *Application Criteria* (Table 2.4). Model and estimate criteria evaluate the model and the estimations as defined in Fig. 2.1. Estimation method criteria describes the quality of the whole process (from data to the estimation). Application criteria assesses the model application method.

Quality of model and estimate is often seen as the most important criteria, as this compares the estimated values with the actual values. *Inputs required* concerns about the kinds of input and how they can be assessed. *Completeness* defines how many estimations the model is able to give, i.e., whether all needed values can be done with the same model. *Type of estimates* signify the numerical precision, i.e., whether a continuous or discrete scale is available to illustrate the uncertainty of the estimation. Sometimes models allow a *Calibration* to different environments in order to fit a very generalized model to a more specific environment. *Interpretability* describes to which extent a person who is not familiar with the model is still able to interpret it [Briand and Wieczorek, 2002].

The *Assumptions* criterion allows to state how realistic the underlying assumptions of the method are. If the same estimation method obtains the same result on various test runs, the *Repeatability* of the estimation method is high. *Complexity* is a measure to point out how simple or complicated it is to use the method, and, hence, indicates whether the method is prone to errors. *Automation of Modeling* describes how many manual steps are required to construct a model with the method and how much support is given by a tool. *Transparency* deals with the replicability of a model, that is whether the algorithms and statistics used in the method are well documented [Briand and Wieczorek, 2002].

Application Coverage states how generalized the model is and whether it is possible to reuse it in other scenarios (e.g., prediction, benchmarking, risk-assessment). *Generalizability* is similar, but focused on reusability in different development environments. *Comprehensiveness* covers the granularity of the estimation (e.g., estimations at component level or for the whole system). *Availability of Estimates* is a criterion to describe at which point the estimation can be done. A model might be used in different phases of implementation by first using rough data and later using more detailed and accurate input. The final criterion, *Automation of Method Usage*, defines the extent of tool support available for applying the model [Briand and Wieczorek, 2002]. Furthermore, according to Jain [1991], it is important to observe the measurement activities and overhead produced by making measurements, as measurements are using resources which could otherwise be allocated to the users. An overhead of up to 5% is regarded as acceptable [Menasce and Almeida, 2001].

Model Validation

The cost model should also be validated after it was built. The aim of validation is to ensure that the cost model not only fits to the existing training data, but also to new data. The two most common model validation criteria are *Mean Magnitude Relative Error* (MMRE) and *Percentage Relative Error Deviation* (PRED) [Port and Korte, 2008]. The evaluation criteria compare an actual value (e.g., measured response time) with a predicted value (e.g., response time estimated by the model). MMRE is calculated by using the mean *Magnitude of Relative Error* (MRE) [Briand et al., 1999]:

$$MRE_i = \frac{|\text{Actual Value}_i - \text{Predicted Value}_i|}{\text{Actual Value}_i} \tag{2.5}$$

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} \frac{|\text{Actual Value}_i - \text{Predicted Value}_i|}{\text{Actual Value}_i} \tag{2.6}$$

n is the number of pairs available with measured and predicted values. Another possibility is to use MdMRE, which is the median of all MRE values [Briand et al., 1999].

PRED is calculated as follows [Menzies et al., 2006]:

$$PRED(x) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1, & \text{if } \mathrm{MRE}_i \leq \frac{x}{100} \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

x is a value between 0 and 100, usually set to 25 or 30. It should be considered that a good value for MMRE and MdMRE is close to 0, while a good value for PRED is close to 1. MMRE is very sensitive to outliers, whereas PRED and MdMRE are more reliable even when outliers occur [Port and Korte, 2008; Menzies et al., 2006].

2.1.4 Model Application Method

The model application method defines in detail how to obtain an estimation from the cost model. In simple cases, the model application method means choosing values for the parameters. These values are inserted into the cost model and a result (the estimation) can be calculated.

In a more complex case, the model might need to be adapted to the environment before inserting values for the parameters. For example, COCOMO consists of three sub models, each one being valid for a certain project phase. Here, model application method means first choosing the sub model according to the project phase and then setting values for the parameters [Boehm et al., 1995]. In the example of Fig. 2.1, the model application method means choosing a value for A. This value is inserted into the formula to get the estimation.

2.1.5 Estimation

The estimation is the result of the cost model for one case (e.g., a value for each parameter). For the example of Fig. 2.1, the estimation for $A = 10$ is $B = 20$, for $A = 15$, B is expected to be 25, and so on. In most cases not only the mean or median is of interest, but also the expected distribution of the estimated value. Hence, a level of accuracy such as standard deviation or variance should be given together with the mean estimation. It is important to highlight that the estimation is inaccurate and the real mean value will be close to the estimated value, but not exactly the same [Briand and Wieczorek, 2002]. In the example, a standard deviation s, e.g., $s = 0.3$ could be delivered as well to emphasize the measurement's inaccuracies.

2.1.6 Modeling Terminology

To delimit terms used within this thesis from each other, and also from terms used in literature, the terminology is defined precisely in this subsection.

Cost Model The aim of the thesis is to provide a cost model for the Gamification Platform. This model is including several cost factors, which influence the performance of the system, and therewith generate costs in terms of response time. Since the cost model is predicting the response time and throughput of the Gamification Platform, it is also a performance model. However, within the thesis, the model is referenced as cost model with regard to the cost factors. Additionally, polynomial models and queueing network models are used as parts of the cost model. Polynomial models describe the influence of cost factors on service times of components. These components are utilized in the queueing network models, which are used to predict the total response time and throughput. The cost model is the combination of both models.

Cost Factor Cost factors are used in the cost model. They have an influence on the performance of the Gamification Platform. Several cost factors are identified for the cost model, and each cost factor is characterized in its precise influence by using polynomial models.

Model Criteria Briand and Wieczorek [2002] define criteria to qualitatively evaluate a model. These criteria are referenced as model criteria in the thesis and described in detail in Subsect. 2.1.3. Furthermore, they are applied to the cost model in Sect. 7.2.

No.	Name	Description
1	Arrival Process	Distribution of interarrival times (time between two arrivals)
2	Service Time Distribution	Distribution of service times (time one response spends at a component)
3	Number of Servers	Defines how many components of the same type are available in the queueing network
4	System Capacity	Maximum amount of requests within the system, this can be restricted by space availability or to avoid longer waiting times
5	Population Size	Number of service requesters
6	Service Discipline	Scheduling Mode (e.g., FCFS)

Table 2.5: Kendall's Notation Parameters [Kendall, 1953; Jain, 1991]

2.2 Queueing Networks

Queueing Networks are a common technique for investigating and modeling the performance of a system, e.g., response time and throughput. The performance can be examined either at a high level by looking at the system as if it was a black box, or by examining the components of the system at a more detailed level. Systems answering client requests with a finite-capacity resource can be regarded as queueing network [Kleinrock, 1975]. "Queueing theory helps in determining the time that the jobs spend in various queues in the system." [Jain, 1991, p. 507] The response time of a request is the total time a request spends inside the system [Jain, 1991].

Delays can be decomposed into service times and waiting times. Service times are times spent consuming resources (e.g., processors, disks), waiting times are times a request has to wait before it is being served at a resource [Gray, 1993]. Waiting times occur because the resource is used by other requests [Menasce and Almeida, 2001]. The total response time for a request can be calculated by combining waiting and service times of all components, which are visited by the request.

Kleinrock [1976] emphasizes the difficulties in transforming the queueing theory into practice. This is because queueing theory is based on assumptions and conditions rarely to be met in reality, and in reality often much more complex systems are used than those treated in theory. As a consequence, if simplified assumptions are used, estimations with queueing networks can lead to inaccurate results when compared to measured data.

2.2.1 Kendall's Notation

Queueing theory defines six characteristics of systems as defined in Table 2.5 and illustrated in Fig. 2.3. Distributions of arrival times and service times are denoted by the symbols defined in Table 2.6.

Symbol	**Name**
M	Exponentially/Markovian distributed
E_k	Erlang distribution with parameter
H_k	Hyperexponential distribution with parameter k
D	Deterministic (without variance)
G	General (not specified)

Table 2.6: Distribution Symbols in Kendell's Notation [Kendall, 1953; Jain, 1991]

The parameters are concatenated with a slash character, e.g., $G/G/1/\infty/\infty/FCFS$. If the notation only consists of three parameters, which is the usual case, the system capacity and population size are assumed to be infinite, and the service discipline is assumed to be *First Come, First Served* (FCFS). For example, a queue of type G/G/1 has an arbitrary interarrival time distribution, and arbitrary distributed service time, one server, an infinite system capacity and population size, and FCFS as its service discipline [Kleinrock, 1975]. A queue of type M/G/1 has a markovian interarrival time (Poisson, exponential), an arbitrary service time distribution, and 1 server [Kleinrock, 1975].

2.2.2 Illustration

Queueing networks are illustrated in a simple way. In Fig. 2.3, n clients are sending requests to a server. Every client is illustrated by a circle. Requests sent by the clients form a queue at the server, which is depicted by a striped rectangle. If the server is able to create multiple processes to execute the requests in different threads, these m threads are represented by circles. Arrows show the flow of requests through the system.

Resources can be *load-independent*, *load-dependent* or *delay resources*. Fig. 2.4 illustrates all three possibilities. The load-independent resource (Fig. 2.4a) has the same service time independently of load arriving at the queue. A load-dependent resource (Fig. 2.4b) performs differently for a high load and a small load, whereby in most cases the service rate is smaller for a higher load. A delay resource (Fig. 2.4c) has no queue, so that service times and rates are constant at different loads.

2.2.3 Network Types

Queueing networks can either be open or closed. A closed queueing network has no external input, whereas an open queueing network has an external input. On the one hand, in a closed model, the number of jobs does not change over time, because they keep circulating in the system. On the other hand, the number of jobs in an open model does vary over time [Jain, 1991].

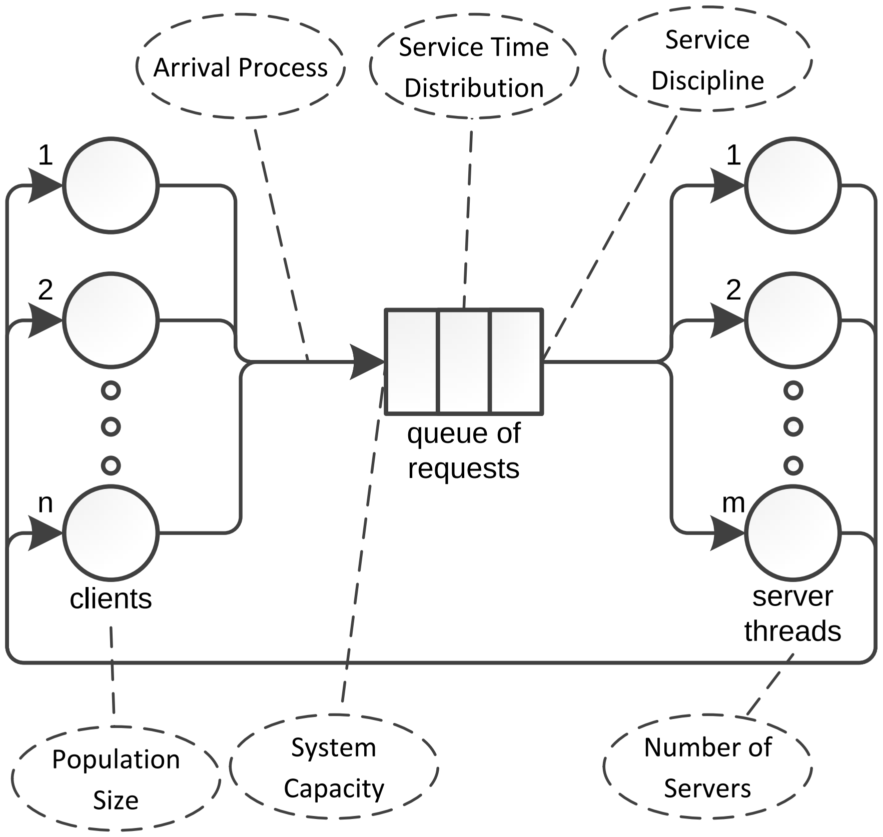
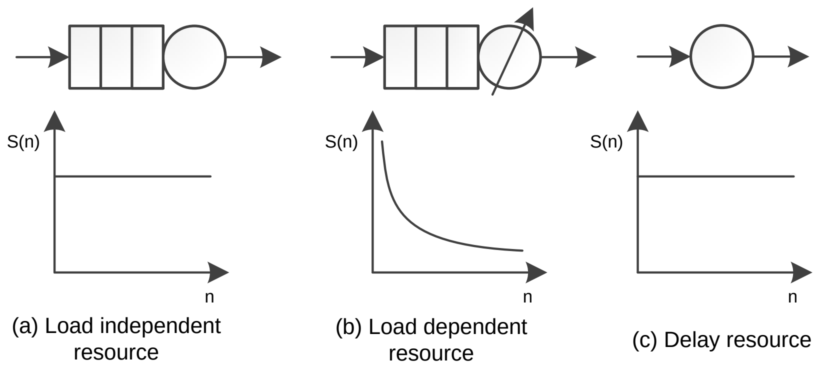


Figure 2.3: Queueing Network of a Multiple Process Server [Menasce and Almeida, 2001]



$S(n)$ = average service time for n requests; n = number of requests

Figure 2.4: Queueing Network Resource Types [Menasce and Almeida, 2001]

The scheduling of queues has an impact on the average response time as well as the distribution of the response times [Adiri, 1969]. Requests are normally scheduled in a FCFS mode at the nodes. This can lead to very different response time distributions based on the service demands as requests are treated equally and have to wait in the queues for the same amount of time independently of their expected service demand. In fact, there are several other scheduling algorithms that can be used in a queueing network, which are listed in [Kleinrock, 1976].

2.2.4 Queueing Network Simulation

The response time calculation is supported by *Java Modeling Tools* (JMT). JMT consists of six programs helping in simulating queueing networks. In this thesis the graphical editor and simulation tool *JSIMgraph* is used. With JSIMgraph, a graphical representation of the queueing network model can be created, and after configuring all system parameters such as service time distributions, classes and interarrival times, this model can be simulated. Of course, an analytical evaluation of a queueing network is also possible, but with a growing complexity of the model, a simulation is more efficient. Each simulation calculates a defined set of performance indexes, such as response times or throughput. These performance indexes are presented together with a calculated confidence interval [Bertoli et al., 2009].

Additionally, a what-if analysis can provide helpful information for different scenarios such as varying interarrival times or service times. This functionality can be used to detect the point at which the system is overloaded [Bertoli et al., 2009]. To simulate a queueing network and predict response time/throughput using *JSIMgraph*, the following inputs are necessary:

- Structure (resources/components, queues, connections)
- Classes (of requests), each one having
  - a mean interarrival time
  - an interarrival time distribution
- Probabilities for routing of the request classes
- Resources/Components, each one having
  - a mean service time
  - a service time distribution

2.3 RETE

This section is about the RETE algorithm introduced by Charles L. Forgy in the 1980s [Forgy, 1979, 1982]. The RETE algorithm is designed to quicken the times for evaluating many patterns for many objects [Forgy, 1982]. The algorithm trades off time against memory usage to evaluate conditions of rules [Stuckenschmidt and Broekstra, 2005; Albert and Régnier, 1991]. In this section, a general explanation of the RETE algorithm and its advantages is given at the beginning. Subsequently, cost and performance estimation of the algorithm are discussed.

2.3.1 RETE Terminology

A production system uses several *productions* or *rules*, each consisting of an if-then statement. The if-part is called *LHS* (left-hand side) or *condition*, the then-part is called *RHS* (right-hand side) or *action*. A *working memory element* (WME) is an object in the working memory which has several attribute-value pairs [Forgy, 1982], like an event in terms of ECA [Paton and Díaz, 1999]. In Drools, which is based on the RETE algorithm and used as implementation of the CEP in the Gamification Platform, WMEs are called *facts* [The JBoss Drools Team, 2013].

The LHS of a rule can be further broken down into a sequence of *patterns*, where every pattern is a partial description of a working memory element [Forgy, 1982]. The patterns are represented by nodes in a tree. The algorithm is efficient for many rules and many objects, because nodes are shared between rules and do not need to be maintained twice [Doorenbos, 1995].

RETE uses two disjoint memories, *production memory* and *working memory*. *Production memory* holds the rules in a tree-structured sorting network, which is compiled according to the patterns of the rules. *Working memory* holds a list of objects satisfying the pattern of the node for each node of the tree at run time. As a result, it is unnecessary to continuously iterate over the objects to check whether they fit the patterns. This information is stored between the cycles, so that large amounts of objects can be handled efficiently. The lists are updated any time a WME is inserted, updated or deleted from the working memory [Forgy, 1982]. Conditions are mainly based on the content of the working memory, whereas actions normally change the working memory by adding, changing, or deleting facts [Forgy, 1979, 1982].

Tokens are descriptions of working memory changes. Each token consists of a *tag*, + or –, and a list of data elements. A + tag is used for adding a WME, a – tag for a deletion of a WME from the working memory. Hence, an update is a sequence of – and + tag [Forgy, 1982].

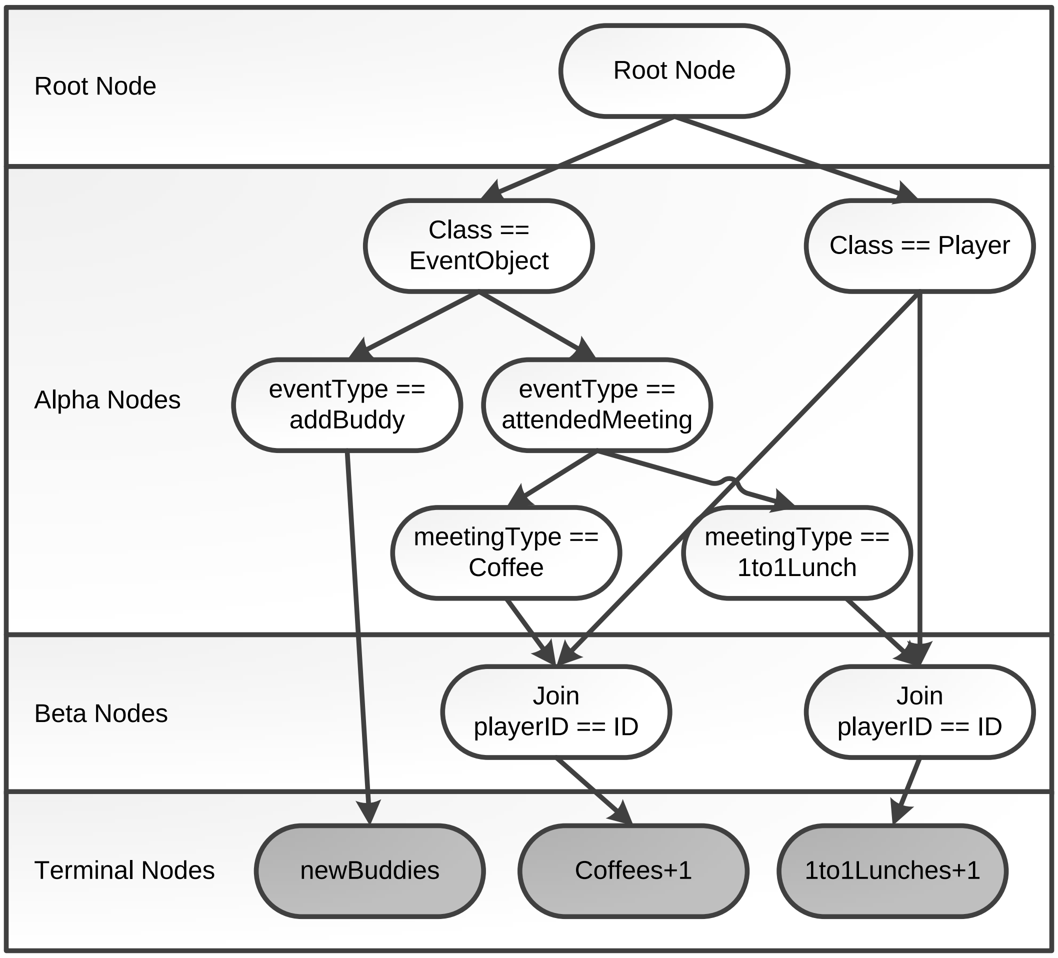


Figure 2.5: RETE Tree Example

2.3.2 RETE Tree

A RETE tree can be divided into a *root node*, several *alpha nodes*, *beta nodes* and a *terminal node* for each rule. One example is given in Fig. 2.5. The root node receives tokens and passes copies of it to all of its successors [Forgy, 1982].

Alpha Nodes

Alpha nodes only consider intra-element features, so they involve only one input and evaluate only one working memory element [Forgy, 1982]. The first alpha nodes usually check for the class of the object. In the other alpha nodes, literal conditions are evaluated in a linear sequence defined by the patterns of a rule [The JBoss Drools Team, 2013; Forgy, 1982]. In each alpha node, one feature of the WME is tested. The most bottom nodes of the alpha nodes are called *alpha memory* [Barachini, 1994].

Facts stored in the alpha memory are propagated to its successors, which are either beta or terminal nodes.

In the example of Fig. 2.5, each inserted fact is being duplicated and sent to the two alpha nodes connected to the root node. Its left successor propagates only facts of class *EventObject*, whereas its right successor propagates only facts of class *Player*. For instance, if the fact is of class *Player*, it is sent to the right inputs of the two beta nodes. A fact of type *EventObject* is sent to the two successor alpha nodes of the *EventObject* class check node. These next nodes check for the event type. If the event type is *addBuddy*, the fact is sent to the terminal node *newBuddies*. If the event type is *attendedMeeting*, the fact is sent to the next two successors of this node. These nodes, in turn, check for the meeting type. If the fact fulfills one of the conditions, it is propagated to the left input of the corresponding beta node.

Beta Nodes

In beta nodes, the inter-element features are tested. Each beta node has two inputs, whereby each input maintains a list of facts. The two lists are called *left* and *right memory*. It is also possible that a beta node serves as input for another beta node [Forgy, 1982].

Facts stored in the alpha memory are used in the input nodes for two beta nodes in the example in Fig. 2.5. For the join node on the left, facts of type *EventObject*, event type *attendedMeeting* and meeting type *Coffee* are inserted as a list into the left input. At the right input of this beta node, all facts of type *Player* are used. Each fact from the left input is joined with each fact in the right input. The facts are combined on *playerID* of the *EventObject* and *ID* of the *Player*, and this new combined fact is propagated to the terminal node *Coffees+1*. The join node on the right behaves similarly, the difference being that on the left input facts have to be of meeting type *1to1Lunch*, and resulting facts from the join are copied and sent to the *1to1Lunches+1* terminal node.

Terminal Nodes

The terminal nodes are the leaf nodes in the RETE tree. Each rule is represented by one terminal node, and the pattern belonging to the rule leads the way to the terminal node [Forgy, 1982]. Every path in the tree has to end in a terminal node. If an object reaches a terminal node, it is inserted into the *conflict set* [Gupta, 1984]. The *conflict set* is a collection of ordered pairs of a rule and lists of facts which match the LHS of this rule [Forgy, 1982, 1979]. If an object is inserted into the *conflict set*, the RHS of the corresponding rule is executed [Barachini et al., 1992].

In the example of Fig. 2.5, three terminal nodes exist (*newBuddies*, *Coffees+1*, and *1to1Lunches+1*), with every node representing a rule. Each fact arriving at the *newBuddies* terminal node initiates the execution of the action part of the *newBuddies* rule, e.g., which gives the player one point. The same is true for the other terminal nodes, whose inputs are beta nodes.

```
1 rule "Coffees+1"
2   when
3     $p : Player($playerid : uid)
4     $evt : EventObject($playerid==playerid, type=='attendedMeeting', data['
      meetingType']=='Coffee') from entry-point eventstream
5   then
6     updateAPI.givePoints($playerid, 'Coffees', 1, 'Attended a Coffee Meeting');
7 end
```

Listing 2.1: Coffees+1 Rule

Complexity Measure	**Best Case**	**Worst Case**
Effect of working memory size on number of tokens	O(1)	O(W^C)
Effect of production memory size on number of nodes	O(P)	O(P)
Effect of production memory size on number of tokens	O(1)	O(P)
Effect of working memory size on time for one firing	O(1)	O(W^{2C-1})
Effect of production memory size on time for one firing	O($\log_2 P$)	O(P)

Table 2.7: Space and Time Complexity of RETE Algorithm [Forgy, 1982]

Rules

The rule *Coffees+1* from the example in Fig. 2.5 is defined in List. 2.1 as it would be implemented in Drools. In this rule, a player (line 3) is joined with the EventObject (line 4) based on the *playerid*. Additionally, the type of the EventObject is limited to *attendedMeeting*, and the *meetingType* to *Coffee*. The RHS is then giving the player a point, but this precise action is not illustrated in the RETE tree.

2.3.3 Response Times and Cost Model for RETE

Forgy [1982] defined time and space complexity in the RETE algorithm for best and worst case in big O notation as summarized in Table 2.7. C is the number of patterns in a rule, P the number of rules in production memory, and W is the number of elements in working memory. The worst case for one firing depends on both production and working memory, in total

$$O(W^{2C-1}) + O(P) = O(W^{2C-1} + P) \tag{2.8}$$

RETE's complexity is in the worst case linear to the number of rules, and polynomial to the number of facts; in the best case the complexity is constant. In practice, RETE's run-time complexity lies somewhere between best and worst case, and is highly dependent on rule characteristics and fact distribution [Albert and Fages, 1988]. The most time consuming step in the RETE algorithm is the join step in the beta nodes, which, according to Gupta et al. [1986], constitutes about 90% of the time. As a consequence, the time span during which facts move through the alpha nodes is negligible. Hashing techniques can reduce complexity of search in local memory, whereby joins are performed on the hashed facts [Albert, 1989].

Albert [1989] defines the average cost of the RETE algorithm based on generating function theory. Although the scientific relevance remains unclear, Albert reaches interesting conclusions about the average costs of the RETE algorithm. Since the formulas and approach presented in his paper are too general and complex for the thesis' purpose, they will not be taken into consideration any further. For example, all results presented in the paper are based on the assumption that each path is traversed with the same probability. In paragraph 5.2 this assumption is reversed by introducing a probability for each path, which should be used to modify all previous results. As this is not described closely enough, the reader is left uncertain on how exactly the constant should be applied to the previous results. As a result, the average cost of the RETE algorithm for alpha nodes and join beta nodes has to be researched and calculated without being able to use previously generated knowledge.

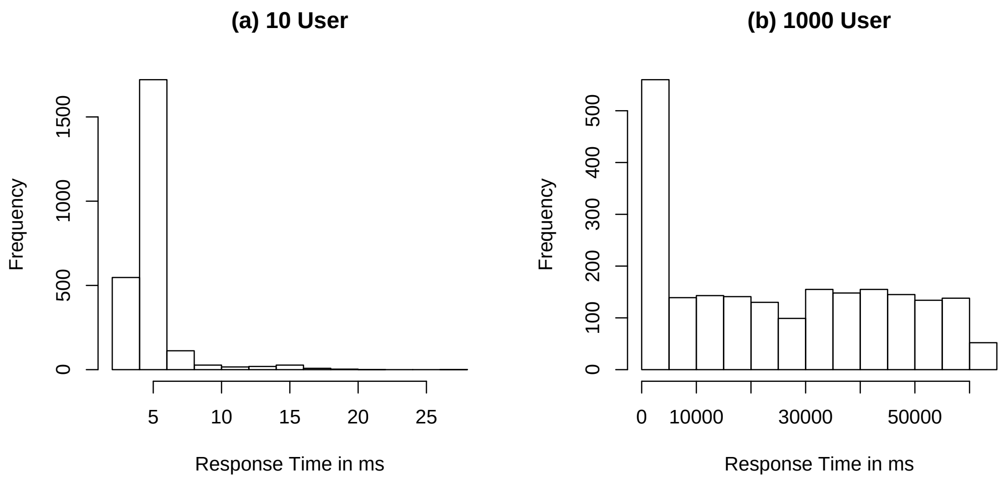


Figure 2.6: Distribution of Response Times for Different Number of Users

2.4 Statistics

In this section, statistical methods used within the thesis are explained. The *Mann-Whitney U test* is used in Chap. 4 to determine which cost factors significantly affect the response time and throughput. Samples will be drawn for two distinct levels of each cost factor, and sample medians and distributions will be compared to detect differences caused by the cost factor. *Polynomial Regression* allows to determine the influence of each cost factor in Chap. 6. In order to avoid overfitting, the polynomial models are evaluated with *Information Criteria* to select the most parsimonious model.

2.4.1 Mann-Whitney U Test

The *Mann-Whitney U Test* compares the means of two independent samples. The null hypothesis for this test is that both samples belong to the same population [Kirk, 2007]. The alternative hypothesis states that the samples belong to two different populations. In this test the distribution of the values can be arbitrary. It is similar to a *t-test*, whereas for the *t-test* the data has to follow a normal distribution [Black, 2009].

As response times in the Gamification Platform are not normally distributed, the Mann-Whitney U test has to be used. For example, for 10 users (Fig. 2.6a), the distribution is highly right-skewed; for 1,000 users (Fig. 2.6b) the right-skewed distribution is still indicated. However, the distribution and mean value change significantly, which is confirmed by using the Mann-Whitney U test in Subsect. 4.1.1.

Each Mann-Whitney U test concludes with a value for p. “The p-value defines the smallest value of alpha for which the null hypothesis cannot be rejected.” [Black, 2009, p. 308] In the U test the samples are compared using an α of 1% (two-sided, i.e., 0,5% on each side). So, if the p-value is below 0.01, the hypothesis has to be rejected and both samples are assumed to belong to different populations.

2.4.2 Polynomial Regression

The relationship of a single independent variable on a dependent variable can be described with a polynomial of order k. It is an expression of the form

$$y = c_0 + c_1x + c_2x^2 + \cdots + c_kx^k = \sum_{i=0}^{k} c_ix^i \quad (k \geq 0) \tag{2.9}$$

having c_i as its k coefficients. In general, the curve of a kth degree polynomial fitting best to the data is considered as the kth model [Kleinbaum, 2007]. In R^1, the best fit is calculated using a least squares approach [The R Foundation for Statistical Computing, 2013b].

To determine the quality of the polynomial models, the coefficient of determination, R^2, is given [Rawlings et al., 1998]. It is calculated as follows [The R Foundation for Statistical Computing, 2013c]:

$$R^2 = 1 - \frac{\sum R_i^2}{\sum y_i^2} \tag{2.10}$$

with R_i as residuals, and y_i as y-values.

Since for n measured sample values, a perfect curve with sum of all residuals being zero can be found by using a nth degree polynomial, the best model is chosen by using three information criteria.

2.4.3 Information Criteria

To select the best polynomial model while disallowing overfitted models, information criteria are used. The most common two, Akaike Information Criterion and Bayesian Information Criterion, are used. Additionally, the error rates of cross-validations are compared as a third criterion for selecting the best non-overfitted model.

For the purpose of making the polynomial model selection process as transparent as possible, the results of all three model criteria are used. Typically, the model with the smallest amount of parameters suggested by one of the three criteria will be chosen in order to keep the cost model as simple as possible.

[1]“R is a language and environment for statistical computing and graphics”[The R Foundation for Statistical Computing, 2013a], which is used to perform polynomial regression and other statistical tasks in this thesis.

Akaike Information Criterion

The polynomial model selection process is supported by the Akaike Information Criterion (AIC). This criterion “provides a mathematical formulation of the principle of parsimony in the field of model construction” [Akaike, 1974, p. 722] AIC is defined as

$$AIC = -2log(L_i) + 2p_i \tag{2.11}$$

with L as maximum likelihood for the model i, and p as independently adjusted parameters in model i [Wagenmakers and Farrell, 2004; Akaike, 1974]. The aim is to minimize AIC for a model, which is then selected as the best model. The larger a model is, the more it is penalized [Faraway, 2002]. At a specific point the penalization of adding another parameter is higher than the explanation gain caused by the parameter. The polynomial model with the lowest AIC is selected eventually and, thus, avoids overfitting [Wagenmakers and Farrell, 2004].

Bayesian Information Criterion

The Bayesian Information Criterion (BIC) is used similarly to AIC. BIC is defined as

$$BIC = -2log(L_i) + p_i log(n) \tag{2.12}$$

with, again, L as maximum likelihood for the model i, p as independently adjusted parameters in model i and n as number of observation used for the likelihood calculation [Wagenmakers and Farrell, 2004; Kenneth P. Burnham, 2004]. BIC is seen as being more consistent as AIC, and that BIC penalizes parameters more than AIC. As a result, BIC will often prefer a model with less parameters compared to AIC [Wagenmakers and Farrell, 2004].

Cross-Validation

Polynomial model selection is also supported by *cross-validation*. A k-fold cross-validation splits the data into k subsets of the same size. The polynomial model is built based on data from $k-1$ subsets and validated on the remaining data set. This is performed k times, while each of the k subsets is used once as validation set [Kohavi, 1995].

In order to avoid overfitting, cross validation is used as a third criterion to determine the best model. In this case, a 5-fold cross-validation is chosen, since $k = 5$ is the standard value in the R function *cvFit* [Alfons, 2013]. The prediction error of two models using different numbers of parameters are compared, and the model with the smaller prediction error is preferred.

Chapter 3

System Analysis

In this chapter, the systems' components are analyzed and queueing networks representing the system are discussed. The structures of the queueing networks are derived directly from the architecture of the Gamification Platform. These queuing networks are used to estimate the performance.

3.1 Components

The Gamification Platform comprises several components, which are influencing the performance. Those components are depicted in Fig. 3.1 for the synchronous mode. A source system with several users, which is the application accessing the Gamification Platform, sends events to the rule engine (CEP). A gamification repository (BEP) handles context information such as the number of points for each player. The *queryAPI* reads data from the BEP, while the *updateAPI* updates data in the BEP.

In asynchronous mode (Fig. 3.2), a message broker handles communication between the components. In order to avoid delays and race conditions, a duplicate of the

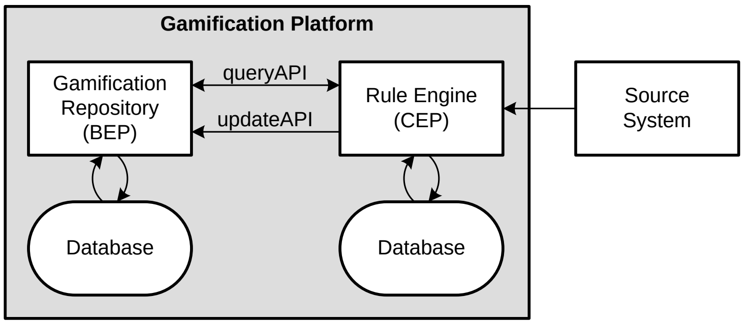


Figure 3.1: Gamification Platform in Synchronous Mode

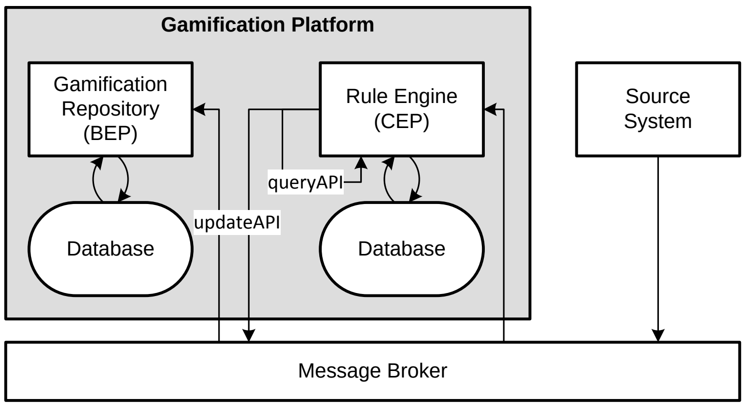


Figure 3.2: Gamification Platform in Asynchronous Mode

context is held in the CEP. The *updateAPI* updates the internal context, to keep the data in the CEP up to date, and also updates the BEP via message broker. The *queryAPI* accesses the context directly stored in the CEP, if available. Additionally, if the BEP is changed externally, the CEP synchronizes with the BEP in order to achieve a consistent state. But, for the purpose of the thesis, such external changes are not considered. In both modes, player data used in the BEP and, e.g., rule knowledge used in the CEP, are persisted in databases.

3.2 Queueing Networks

This part gives an overview about the queueing networks for the synchronous as well as for the asynchronous mode of the Gamification Platform. Both queueing networks are designed as open networks (Subsect. 2.2.3), because the number of requests in the Gamification Platform changes over time and is not fixed. Additionally, all resources are either depicted as load independent or delay resources as defined in Subsect. 2.2.2.

Even though all components are modeled as being load independent, they are depending on several cost factors, including the load (represented by the number of events per second). The service time not only increases if more requests need to be handled, but also if the rules are more complex or other cost factors of the cost model vary. That is why load is regarded as one of the factors, instead of using load dependent resources. Hence, all cost factors are modeled equally by changing the service times of load independent resources manually. Furthermore, with this approach it is possible to use service time distributions other than constants for the resources, as the service times in reality are varying based on exponential distributions.

The notation for both queueing networks as described in Table 2.5 is defined as $M/M/1/\infty/\infty/FCFS$. Interarrival and service times are exponentially distributed as it will be shown based on the measured data in Sect. 5.4 and Subsect. 5.5.1. The system capacity and population size are assumed to be infinite for simplification, and the service discipline is set to FCFS.

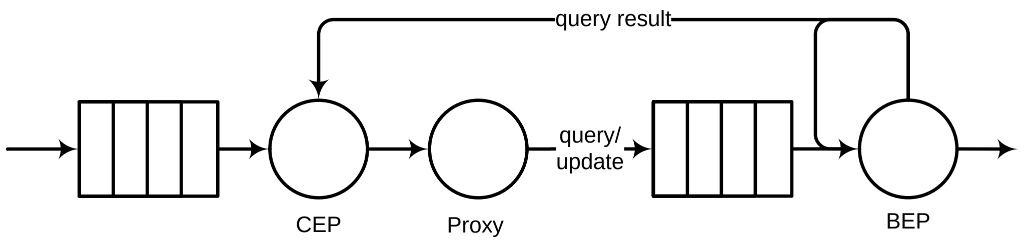


Figure 3.3: Queueing Network of Gamification Platform in Synchronous Mode

3.2.1 Synchronous Mode

The queueing network for the synchronous mode is depicted in Fig. 3.3. It includes both CEP and BEP as components. Every component employs its own queue to manage waiting requests. Depending on the rules, queryAPI and updateAPI can be called for each event. Additionally, a proxy is implemented, which in the synchronous mode is only a component delaying the requests and forwarding each request to the BEP.

A simple rule might only trigger an update of the BEP. For such a rule, an event (request in queueing network terminology) enters the system, passes the queue of the CEP component and is being served at the CEP. Afterwards, an update is sent to the BEP through the proxy. The request first passes the queue and is then being served at the BEP. Finally, the request leaves the system. If rules change the context of the scenario, the updated object returns to the CEP.

A more complex rule might use both updateAPI and queryAPI. If the rule contains several calls to the queryAPI, the BEP is visited multiple times to collect context information for evaluating the condition. A request enters the system and passes queue and CEP. Subsequently, for every queryAPI call, the request is sent to the BEP (through proxy and BEP queue), and the result is directly returned to the CEP, from where the next queryAPI call is started. Additionally, a BEP query can trigger another BEP query, for example, the function *hasPlayerMission* uses the function *getMissionsForPlayer* to receive all missions of the player and loop through them determining whether the mission is assigned to the player. Queries and updates to the BEP are served at the same component and, hence, waiting in the same queue.

3.2.2 Asynchronous Mode

The queueing network for the asynchronous mode is depicted in Fig. 3.4. Again, CEP, BEP, and proxy are included as components in the queueing network. Additionally, two message broker components and the CEP context component are integrated. JMS, which is used as implementation of the message broker, is offering both *Point-to-Point* and *Publish-Subscribe Model*. The former works with queues for requests, whereas the latter works with topics. Messages can be published to a topic and

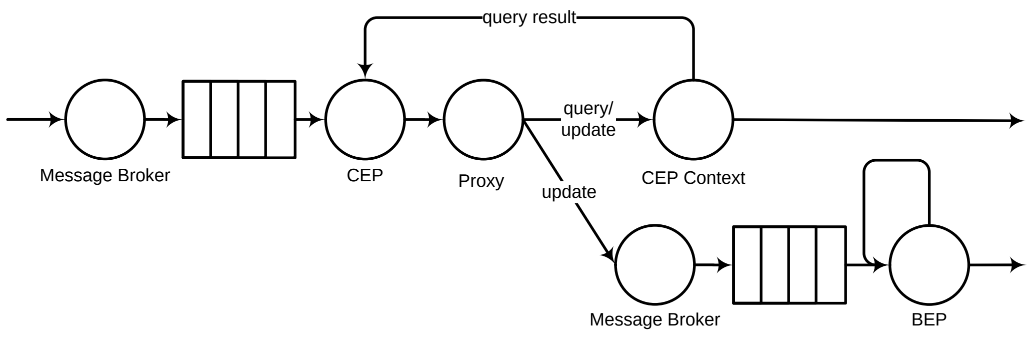


Figure 3.4: Queueing Network of Gamification Platform in Asynchronous Mode

consumed by subscribers of that specific topic [Sun Microsystems, Inc, 2012; Apache Software Foundation, 2013]. Within the Gamification Platform, the message broker distributes messages to all interested receivers using the Publish-Subscribe Model.

If a request is sent asynchronously to the system, the message broker forwards the request without delay to the queue of the CEP. Depending on the rules and whether context information is needed for the request, it uses the queryAPI. As the context is duplicated in the CEP, the query is processed at the CEP Context component. The CEP Context component is addressed via the proxy, which processes every request immediately as it is a delay resource without a queue. After the result is sent back to the CEP, the next queryAPI call is started. When all context retrievals are completed, the BEP and context might be updated through the updateAPI. The proxy distributes the calls to CEP Context and BEP (through the message broker) simultaneously to update the duplicated context. After this update is finished, the request leaves the system. Again, if the application includes rules based on the context, the updated object is returned to the CEP.

3.2.3 Measuring Points

Measuring points are inserted before and after each component as shown for the synchronous mode in Fig. 3.5 and for the asynchronous mode in Fig. 3.6. The measuring points are also described in Table 3.1. Using the measuring points, the service times of each component can be calculated as defined in Table 3.2. CEP and BEP service times are subdivided into update time and query time, since cost factors might influence the service times of accessing or updating data only. Based on the measured service times as well as their distributions polynomial models can be built which allows the calculation of service times before implementation. The calculated values can be inserted into a simulation tool for queueing networks to estimate the total response times and find bottlenecks in the system.

In the next chapter, the service times are analyzed for each cost factor in order to determine which cost factors influence the service times. Subsequently, service times are researched for each of these cost factors more closely using the measuring

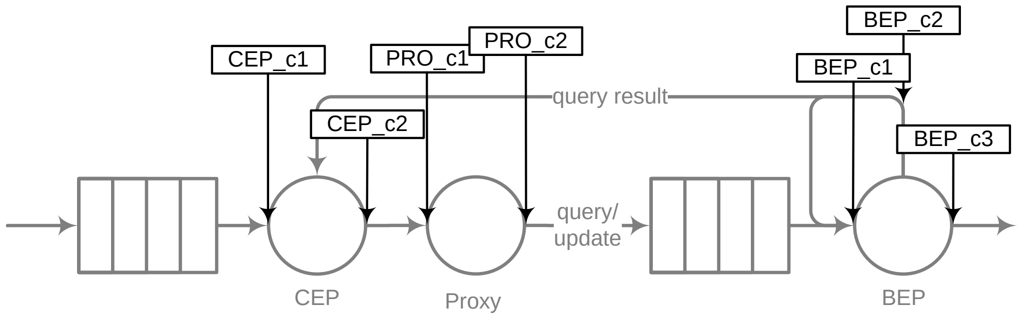


Figure 3.5: Measuring Points in Queueing Network, Synchronous Mode

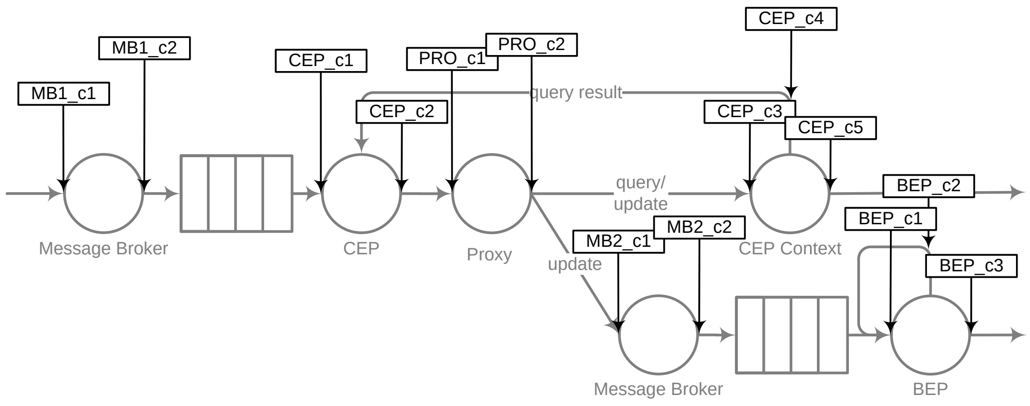


Figure 3.6: Measuring Points in Queueing Network, Asynchronous Mode

points and components introduced here. It is then possible to determine the service times of each component beforehand based on the cost factor level demanded by an application.

Measuring Point	**Description**
BEP_c1	entering BEP component
BEP_c2	request with query result leaving BEP component
BEP_c3	leaving BEP component
CEP_c1	entering CEP component
CEP_c2	leaving CEP component
CEP_c3	entering CEP context component
CEP_c4	request with query result leaving CEP context component
CEP_c5	leaving CEP context component
MB1_c1	entering message broker 1
MB1_c2	leaving message broker 1
MB2_c1	entering message broker 2
MB2_c2	leaving message broker 2
PRO_c1	entering proxy
PRO_c2	leaving proxy

Table 3.1: Measuring Points

Calculation	**Service Time of Component**
BEP_c2 - BEP_c1	BEP query
BEP_c3 - BEP_c1	BEP update
CEP_c2 - CEP_c1	CEP
CEP_c4 - CEP_c3	CEP query
CEP_c5 - CEP_c3	CEP update
MB1_c2 - MB1_c1	message broker 1
MB2_c2 - MB2_c1	message broker 2
PRO_c2 - PRO_c1	proxy

Table 3.2: Service Times Calculation for Components Based on Measuring Points

Chapter 4

Cost Factors

This chapter focuses on the factors of the cost model. The cost factors are divided into three categories: Users, rules, and infrastructure. In each of the categories, several cost factors are defined and described. A few cost factors are assumed to influence the performance, but are excluded from the cost model nevertheless. In each such case, the reasons for excluding the cost factor are explained.

The influence of each cost factor on response time is tested at two different orders of magnitude of its intensity, for a time span varying from factor to factor to create sufficient amounts of events. With those two samples a *Mann-Whitney U Test* (Sect. 2.4.1) is conducted to decide whether the cost factor has a significant influence. If the cost factor has a statistically significant impact on response time or throughput, it will be included in the test series to further investigate the exact effect.

For the first tests, which are performed to immediately identify and exclude irrelevant cost factors, the Gamification Platform is operated in synchronous mode. As the Gamification Platform in synchronous mode performs slower than in asynchronous mode, changes of total response times can be found by considering only the synchronous mode. If no difference between the two samples in synchronous mode can be found, the cost factor is discarded because it will have turned out as insignificant for the performance of the Gamification Platform.

Even though Menasce and Almeida [2001] listed several factors with influence on performance (Table 2.2), most of their very general factors are not considered in the cost model. In the first version, the aim is to evaluate the influence of cost factors defined by the architecture and applications using the Gamification Platform, whereas the more general cost factors can be further investigated in future work.

To present the results, mean, median and maximum response times (in ms) are listed in a table for each cost factor. The p-value is gained from the *Mann-Whitney U Test* by comparing both samples. The null hypothesis in this test declares that both samples are taken from the same population. If the null hypothesis is rejected, the samples do not belong to the same population with a certain error probability. The latter means that the cost factor modifies the response time.

```
rule "newBuddy"
  when
    $addBuddy : EventObject(type=='addBuddy', $playerid:playerid) from entry-
     point eventstream
  then
    updateAPI.givePoints($playerid, 'Buddies', 1, 'added 1 buddy');
    retract($addBuddy);
    update(queryAPI.getPlayer($playerid)); //only in synchronous mode
end

rule "tenBuddies"
  when
    p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'I Have Got Buds!') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Buddies').getAmount() >= 10)
  then
    updateAPI.completeMission($playerid, 'I Have Got Buds!');
    update($p); //only in synchronous mode
```

Listing 4.1: Simple Rule Example

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
10 User	5.13	4.93	26.77	$< 2.2 * 10^{-16}$
1000 User	25096.28	23120.73	61885.84	

Table 4.1: Response Time Metrics for Cost Factor User

4.1 User-Related Cost Factors

In this section, the cost factors concerning users are described, which include number of users, events per second per user and event type distribution. These cost factors are *workload intensity parameters* as defined in Table 2.1 and change the load placed on the system. To test these cost factors, two rules are used: *newBuddy* and *tenBuddies* (List. 4.1). In the *newBuddy* rule, an event of type *addBuddy* causes an action. In this action, the player is rewarded 1 *Buddies* point. The *tenBuddies* rule completes the *I Have Got Buds!* mission once the player has collected 10 buddies.

4.1.1 Number of Users

It is assumed that each user performs a distinct number of events per second on average. Therefore, the number of users is expected to influence the response time and throughput, as more users generate a higher amount of events per second.

In the test case, every user creates 0.5 events/s. The number of users is varied from 10 to 1,000, the test is performed for a time span of 500s or 5s to create 2500 Events in both cases. Test results are shown in Table 4.1. As the p-value is $2.2 * 10^{-16} \ll 0.01$,

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
0.5 Events/s	5.43	5.05	90.68	$< 3.211 * 10^{-16}$
50 Events/s	26139.11	22098.96	72320.28	

Table 4.2: Response Time Metrics for Cost Factor Events/s

the null hypothesis has to be rejected, which means that the number of users has a significant influence on the response time.

4.1.2 Number of Events/s per User

The more events a user or application sends to the Gamification Platform, the higher the system's workload. Hence, the event rate per user is expected to influence the response time and throughput as well.

In the test case, 10 users are simulated. The number of events/s created by every user is varied from 0.5 to 50, the test is performed for a time span of 600s or 6s to create the same amount of events. Test results are shown in Table 4.2. As the p-value is $2.2 * 10^{-16} \ll 0.01$, the number of events per second has a significant influence on the response time. It is expected that the cost factor number of users and events per second are convertible or that they could be summarized into one factor.

4.1.3 Event Type Distribution

It is asuumed that events of different event types, sent in different frequencies, influence the response time. A rule with a join node is used to test this assumption (List. 4.2). The events, which are joined on the *playerid*, are inserted in different frequencies to measure the influence of the event type distribution. For the first case, 1 event of type *addTag* is inserted at the beginning, and all other events are of type *addBuddy*. In the second case, events of type *addBuddy* and *addTag* are inserted in turns. The event distribution is shown in Fig 4.1. Fig. 4.1a shows the unbalanced event types and Fig. 4.1b the balanced event types.

The test simulates 10 users. The number of events/s created by every user is 0.5, the test is performed for a time span of 100 seconds. Test results are shown in Table 4.3. As the p-value is $2.2 * 10^{-16} \ll 0.01$, the null hypothesis stating that both samples belong to the same population has to be rejected, which means that the event type distribution has a significant influence on response time.

4.2 Rule-Related Cost Factors

In this section, the cost factors concerning rules are described. In this category, the cost factors number of rules, number of alpha and beta nodes, node types, number

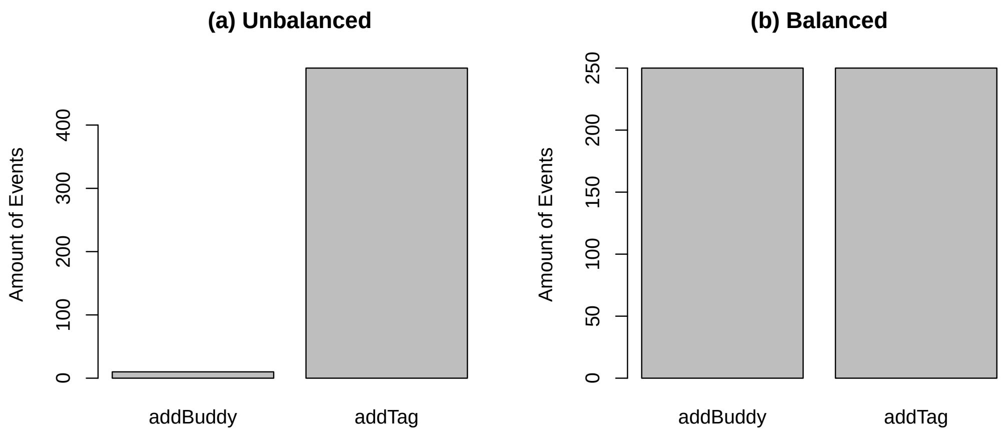


Figure 4.1: Event Type Variation

```
rule "AddedBuddyAndAddedTag"
  when
    $addBuddy : EventObject(type=='addBuddy', $playerid:playerid) from entry-
    point eventstream
    $addTag : EventObject(type=='addTag', playerid==$playerid) from entry-point
    eventstream
  then
    updateAPI.givePoints($playerid, 'Experience', 1, 'TestReason');
end
```

Listing 4.2: Join Rule Example

Cost Factor	**Mean (ms)**	**Median (ms)**	**Max. (ms)**	**p-Value**
Unbalanced	7.86	6.88	27.45	$< 2.2 * 10^{-16}$
Balanced	667.82	93.92	21940.93	

Table 4.3: Response Time Metrics for Cost Factor Event Type Distribution

of abstractions, working memory growth, independent rule streams, updateAPI and queryAPI calls, size of tables, and structure of tables are analyzed. These cost factors are workload service demand parameters as defined in Table 2.1 and influence the service times of the components.

```
rule "AddedBuddy1"
  when
    $addBuddy : EventObject(type=='addBuddy1', $playerid:playerid) from entry-
    point eventstream
  then
    updateAPI.givePoints($playerid, 'Experience', 1, 'TestReason');
end
```

Listing 4.3: Alpha Nodes Rule Example

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
10 Alpha Nodes	297.26	11.93	24399.0	$< 2.2 * 10^{-16}$
300 Alpha Nodes	22145.96	17.18	237846.8	

Table 4.4: Response Time Metrics for Cost Factor Number of Alpha Nodes

4.2.1 Number of Rules

The number of rules, according to Yu and Jajodia [2007], does not significantly influence the performance. However, the rule set determines the number of alpha nodes and the number of beta nodes. Therefore, not the rules themselves are measured as a cost factor, but the RETE tree and its alpha and beta nodes defined by the rule set. The number of alpha and beta nodes can be derived from the rules, e.g., by using the graphical RETE tree view in Drools, which displays the corresponding RETE tree for each specific rule set.

4.2.2 Number of Alpha Nodes

It is expected that the number of alpha nodes in the RETE tree has an influence on the response time. Several rules are used to test this assumption. In the first case, 10 rules with distinct LHS are used. In the second case, 300 rules with distinct LHS are used. Each rule is very similar to the rule in List. 4.3, and rules only differ in the type of events which trigger this rule. The associated RETE tree is depicted in Fig. 4.2. For example, the first rule is triggered for events of type *addBuddy1*, the second one for events of type *addBuddy2* and so on. This leads to 10 or 300 different alpha nodes. Events are sent to the system with the same probability for each of the existing rules.

The test simulates 250 users. The number of events/s created by every user is 0.5, the test is performed for a time span of 300 seconds. Test results are shown in Table 4.4. As the p-value is $2.2 * 10^{-16} \ll 0.01$, the null hypothesis stating that both samples belong to the same population has to be rejected, which means that the number of alpha nodes has a significant influence on response time.

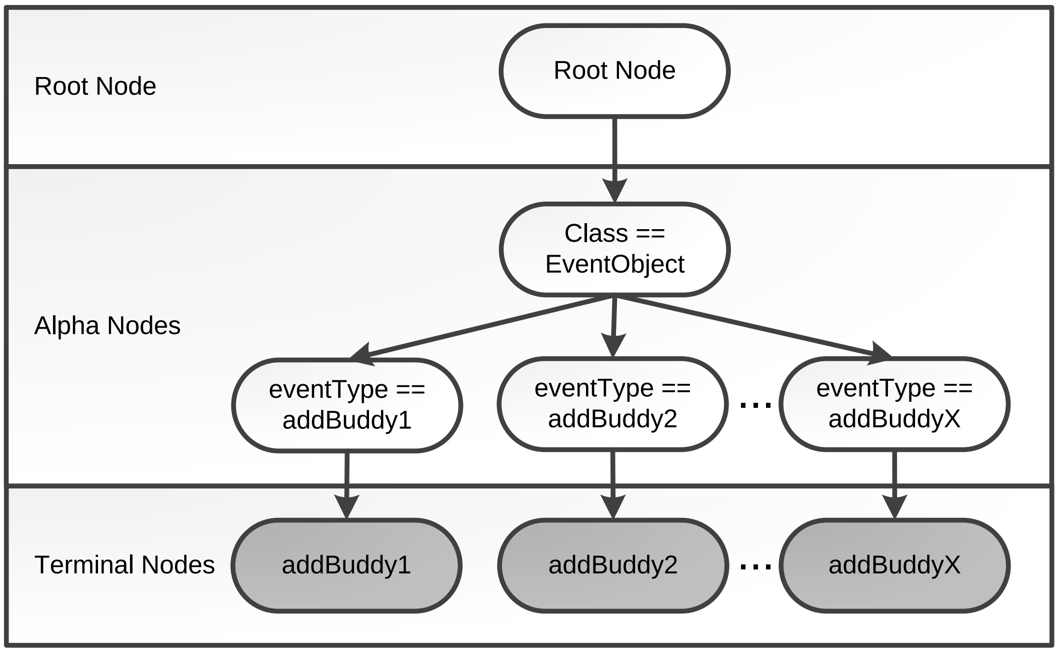


Figure 4.2: RETE Tree for Testing Cost Factor Number of Alpha Nodes

4.2.3 Number of Beta Nodes

The number of beta nodes is, similarly to the number of alpha nodes, expected to influence the response time. Join nodes are considered as one example for beta nodes, whereas other beta nodes, e.g., not nodes, are excluded. Different rules are used to test the influence of beta nodes on the service times. In the first case, 10 rules with distinct LHS are used. In the second case, 300 rules with distinct LHS are used.

Each rule is very similar to the rule in List. 4.4 and rules only differ in the type of events which trigger this rule. The corresponding RETE tree is depicted in Fig. 4.3. For example, the first rule is triggered for events of type *addBuddy1* in combination with *addTag1*, the second one for event combinations of type *addBuddy2* and *addTag2* and so on. This leads to 10 or 300 different beta nodes. Events are sent to the system with the same probability for every rule, but every time in pairs of one *addBuddyX* and one *addTagX* event, for each of the existing rules. It is important to note that both events are retracted after the rule is fired to avoid increasing of working memory elements.

The test simulates 250 users. The number of events/s created by every user is 1, and the test is performed for a time span of 300 seconds. Test results are shown in Table 4.5. As the p-value is $2.2 * 10^{-16} \ll 0.01$, the null hypothesis stating that both samples belong to the same population has to be rejected, which means that the number of beta nodes has a significant influence on response time.

```
rule "AddedBuddyAndAddedTag1"
  when
    $addBuddy : EventObject(type=='addBuddy1', $playerid:playerid) from entry-
    point eventstream
    $addTag: EventObject(type=='addTag1', playerid==$playerid) from entry-point
    eventstream
  then
    updateAPI.givePoints($playerid, 'Experience', 1, 'TestReason');
    retract($addBuddy);
    retract($addTag);
end
```

Listing 4.4: Beta Nodes Rule Example

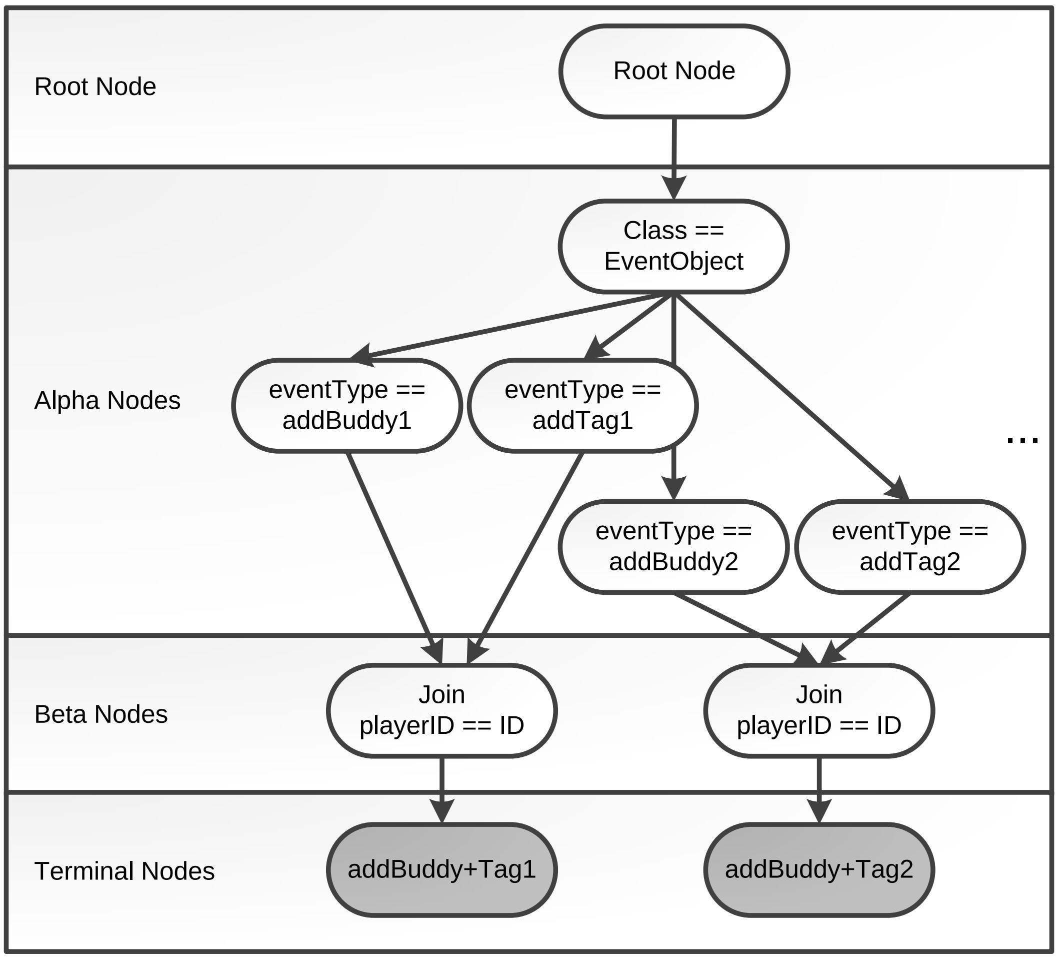


Figure 4.3: RETE Tree for Testing Cost Factor Number of Beta Nodes

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
10 Beta Nodes	37.54	11.94	3231.25	$< 2.2 * 10^{-16}$
300 Beta Nodes	202.28	13.39	38702.79	

Table 4.5: Response Time Metrics for Cost Factor Number of Beta Nodes

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
20 abstractions	7.17	6.51	400.69	$< 2.2 * 10^{-16}$
300 abstractions	13.20	7.08	634.00	

Table 4.6: Response Time Metrics for Cost Factor Abstractions

4.2.4 Node Types

Equality tests in alpha nodes and join nodes as beta nodes are only some of the node types possible. The rule engine allows, for example, many other node types, such as "greater than", "less than", "not", "between" or other complex event processing operators. All these operators influence the response time to a very different extent, according to their complexity. Hence, the other node types should be researched as well. Due to the limited time available for the thesis, however, only the two most common operators are considered, namely join nodes as beta nodes and equality as alpha nodes.

4.2.5 Number of Abstractions

Rules can create new events and insert them into the working memory. After a new event is inserted, the RETE tree is evaluated again. An event created by a rule is called abstraction, and the event is called abstract event. The response time is expected to depend on the number of abstractions, as each evaluation is demanding time.

To test the influence, several rules are used to test a certain amount of abstractions as shown in List. 4.5. Every rule creates an abstract event, which is processed by the next rule. Only the last rule gives one experience point to the player. For example, if 20 abstractions are tested, the first rule is reacting to an event *addBuddy1* and creates an event *addBuddy2* if triggered. The next rule reacts to *addBuddy2* events and creates an *addBuddy3* event every time. The last rule reacts to the last abstract event (in case of 20 abstractions: *addBuddy21*) and rewards the player with one point.

The test is performed for 1 user, 300 seconds, and an event rate of 0.5 event/s. The test results are listed in Table 4.6. As the p-value is $2.2 * 10^{-16} \ll 0.01$, the

```
rule "AddedBuddy1"
  when
    $addBuddy : EventObject(type=='addBuddy1', $playerid:playerid) from entry-point eventstream
  then
    EventObject obj = new EventObject();
    obj.setType("addBuddy2");
    obj.setPlayerid($playerid);
    retract($addBuddy);
    entryPoints["eventstream"].insert(obj);
end

rule "AddedBuddyX"
  when
    $addBuddy : EventObject(type=='addBuddyX', $playerid:playerid) from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Experience', 1, 'TestReason');
    retract($addBuddy);
end
```

Listing 4.5: Abstractions Rule Example

null hypothesis stating that both samples belong to the same population has to be rejected, which means that the number of abstractions has a significant influence on response time.

4.2.6 Working Memory Growth

The working memory, as defined by Forgy [1982] and explained in Subsect. 2.3.3, has a significant influence on the performance of RETE and on the Gamification Platform. If all events are retracted after being used in join nodes, the system's slowdown will remain manageable, but if events are not retracted and the WM grows constantly, every event is processed slower than the previous events. Retracted events avoid slowing down the evaluation of beta nodes on a long-term basis, but if events are not retracted upon their use in join nodes, the possible join matches are increasing and so is the processing time. To demonstrate the influence of the working memory size on response time, one rule with exactly one beta node is used, similar to the rule in List. 4.4. For this test, however, the retractions of events in line 9 and 10 were removed to ensure a constantly growing working memory.

The test is performed for 1 user, 300 seconds and an event rate of 1 event/s. The test results are listed in Table 4.7. The mean, median and maximum values are computed for the first 20 events and compared to the first 2000 events. As the p-value is $2.2 * 10^{-16} \ll 0.01$, the null hypothesis stating that both samples belong to the same population has to be rejected, which means that the working memory growth has a significant influence on response time, as previously stated by Forgy [1982].

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
20 events	64.64	61.20	124.18	$< 2.2 * 10^{-16}$
2000 events	433.94	310.27	1466.90	

Table 4.7: Response Time Metrics for Cost Factor Working Memory Growth

```
rule "newBuddy"
  when
    $addBuddy : EventObject(type=='addBuddy', $playerid:playerid) from entry-
    point eventstream
  then
    for (int i = 0; i < 10; i++) {
      updateAPI.givePoints($playerid, 'Experience', 1, 'TestReason');
    }
end
```

Listing 4.6: Rule with 10 UpdateAPI Calls

This cost factor is highly dependent on the events previously sent to the system. Queueing networks are stateless and independent from requests prior to the current one. This means that the working memory growth cannot be included as a cost factor in the cost model, which is based on a queueing network. In order to take the working memory size into consideration as a cost factor, a different modeling technique would have to be used.

4.2.7 Independent Rule Streams

It is possible to send events to different rule streams, so that rules can use events only from specific rule streams. This enables a more efficient event handling, as distinct branches in the RETE tree can be used for each rule stream. This feature of Drools is currently sparsely used, so it is not evaluated any further.

4.2.8 UpdateAPI calls in RHS

To test the influence of updateAPI calls, the user is rewarded with 10 *Experience* points in 10 successive updates for each buddy added (List. 4.6). This case is compared to the the previous one, where only one point was given to the user, and therefore only used one single updateAPI call. The test is performed for 100 users, 100 seconds, 0.5 events/s and 1 or 10 updateAPI calls. Test results are shown in Table 4.8. As the p-value is $2.2 * 10^{-16} \ll 0.01$, the null hypothesis stating that both samples belong to the same population has to be rejected, which means that the number of updateAPI calls has a significant influence on the response time.

Cost Factor	**Mean (ms)**	**Median (ms)**	**Max. (ms)**	**p-Value**
1 update	9.57	5.32	350.99	$< 2.2 * 10^{-16}$
10 updates	$2.37 * 10^5$	$1.95 * 10^5$	$6.72 * 10^5$	

Table 4.8: Response Time Metrics for Cost Factor UpdateAPI Calls

```
rule "newBuddy"
  when
    $addBuddy : EventObject(type=='addBuddy', $playerid:playerid) from entry-
    point eventstream
    eval(queryAPI.hasPlayerMission($playerid, 'Mission 1'))
    eval(queryAPI.hasPlayerMission($playerid, 'Mission 2'))
    eval(queryAPI.hasPlayerMission($playerid, 'Mission 3'))
     ⋮
    eval(queryAPI.hasPlayerMission($playerid, 'Mission 10'))
  then
    updateAPI.givePoints($playerid, 'Experience', 1, 'TestReason');
    retract($addBuddy);
end
```

Listing 4.7: Rule with 10 QueryAPI Calls

Cost Factor	**Mean (ms)**	**Median (ms)**	**Max. (ms)**	**p-Value**
1 query	9.18	6.98	253.08	$< 2.2 * 10^{-16}$
10 queries	70528.34	68989.55	158938.81	

Table 4.9: Response Time Metrics for Cost Factor QueryAPI Calls

4.2.9 QueryAPI calls in LHS

It is assumed that the player only gets *Buddy* points if he has 10 different missions (*Mission 1-10)*. The rule in List. 4.7 calls the queryAPI to check if the the mission is assigned to the user each time a new *addBuddy* event is inserted. The test is performed for 100 users, 100 seconds, 0.5 events/s and 1 or 10 queryAPI calls. Test results are shown in Table 4.9. As the p-value is $2.2*10^{-16} \ll 0.01$, the null hypothesis stating that both samples belong to the same population has to be rejected, which means that the number of queryAPI calls has a significant influence on the response time.

Cost Factor	**Mean (ms)**	**Median (ms)**	**Max. (ms)**	**p-Value**
1.000 players	29.35	25.47	47.94	0.6882
100.000 players	29.64	27.78	49.31	

Table 4.10: Response Time Metrics for Cost Factor Table Sizes (Table Player)

4.2.10 Size of Tables

It is assumed that the size of the tables storing the context information influences the access time to one database entry. The following tables are used by *updateAPI* and *queryAPI*:

- Player
- PlayerPoints
- Badge2Player
- Mission2Player
- Mission
- Badge
- Point

For the first version of the cost model and due to the restricted time frame of the thesis, only Player and Mission table sizes will be investigated in their influence on the response time.

Player

The test is performed for 1 user, 300 seconds, 0.5 events/s and 1.000 or 100.000 players in the database. Test results are shown in Table 4.10. As the p-value is $0.6882 > 0.01$, the null hypothesis stating that both samples belong to the same population cannot be rejected, which means that the table size of the Player table may not significantly influence the response time.

Mission

To test the influence of the table size of table Mission, a test is performed involving 1 user for 300 seconds, 0.5 events/s, and 1.000 or 100.000 missions in the database. The test results are shown in Table 4.11. The p-value is $0.2241 > 0.01$, which means that the null hypothesis stating that both samples belong to the same population cannot be rejected, which means that the table size of the Mission table may not significantly influence the response time.

Without measuring the other table sizes due to the restricted time frame, it can be said that at least the measured tables do not significantly change the response time

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
1.000 missions	28.66	25.42	48.36	0.2241
100.000 missions	29.52	25.75	47.49	

Table 4.11: Response Time Metrics for Cost Factor Table Sizes (Table Mission)

of the Gamification Platform. Hence, this cost factor is excluded in the cost model. In further research it will either have to be proven that other table sizes do not have an influence as well, or this cost factor needs to be included in a future version of the cost model.

4.2.11 Structure of Tables

The structure of tables has an influence on access time of the data in the tables. Indexes optimize access to the tables. The table structure is automatically handled by the persistence framework (EclipseLink JPA) and database (SAP MaxDB), and is, hence, not evaluated.

4.3 Infrastructure-Related Cost Factors

In this section, the cost factors concerning infrastructure are described. Database (parameters), queue scheduling algorithm, connection pools, transmission packet sizes, and random-access memory (RAM) of the server are included in this analysis. As defined in Table 2.1, these cost factors are classified as system parameters which affect the performance.

4.3.1 Database

The Gamification Platform uses SAP MaxDB as its default database management system (DBMS), but the DBMS can be easily replaced with another solution, e.g., SAP HANA. Since evaluating the differences, advantages, or disadvantages of different DSMS is beyond the scope of this thesis, this cost factor is not evaluated any further.

In addition, SAP MaxDB can be configured and tweaked in many different ways [SAP AG, 2013]. For example, the maximum number of user tasks, which defines the maximum number of locks for database objects, can lead to delays if set to a small value, or the cache memory size can speed up requests which are performed frequently. Many more parameters can be varied which might influence the response time of the database. Investigating all different parameters cannot be achieved in the restricted time frame, so that the database configuration is also not evaluated any

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
25 Threads	8.63	4.98	394.49	0.0001986
2500 Threads	8.29	5.15	275.70	

Table 4.12: Response Time Metrics for Cost Factor Maximum Number of Threads

further. Further research needs to identify the relevant parameters and determine the optimal database configuration for usage in the Gamification Platform.

4.3.2 Queue Scheduling Algorithm

As indicated by Adiri [1969], the scheduling algorithm of queues has an impact on the response time distribution and average response time. In the Gamification Platform based on a Tomcat server, the FCFS scheduling algorithm is chosen, and differences to other scheduling algorithms will not be evaluated [The Apache Software Foundation, 2013].

4.3.3 Connection Pools

Connection pools manage the communication between the components in the application. Each request needs a connection to the Gamification Platform. These connections are pooled, so that only a maximum of parallel connections can be handled efficiently. The maximum number of threads as well as the accept count are expected to influence the response time.

Maximum Number of Threads

The maximum number of request processing threads to be created on the server is expected to influence the response time. When all connections are used in the synchronous case, a new request needs to wait until one of the requests is finished. To test this cost factor a maximum thread count is varied from 25 to 2500. The test is performed for 100 users, 100 seconds, and 0.5 events/s. Test results are shown in Table 4.12. As the p-value is $0.0001986 < 0.01$, the null hypothesis stating that both samples belong to the same population has to be rejected, which means that the maximum number of threads has a significant influence on the response time. From the details in Table 4.12, especially from the small changes for mean and median, it can be concluded that this influence in comparison to other cost factors is very little. However, the measured differences indicate that a threshold exists for this cost factor, which needs to be researched in future work. For the first version of the cost model this cost factor is, hence, excluded.

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Value
Value 10	8.44	5.17	219.96	0.02877
Value 1,000	8.66	5.14	403.92	

Table 4.13: Response Time Metrics for Cost Factor Accept Count

Cost Factor	Mean (ms)	Median (ms)	Max. (ms)	p-Values
8GB	8.12	5.18	206.68	
1GB	8.25	5.19	125.25	0.7492
256MB	8.43	5.18	248.27	0.03381
64MB	287.88	39.29	11753.73	$< 2.2 * 10^{-16}$

Table 4.14: Response Time Metrics for Cost Factor RAM Size

Accept Count

The accept count defines the maximum queue length of requests waiting for connections. If a request has to wait for a connection thread, it is queued. If the queue is already full, the request is rejected; otherwise it waits until a connection is provided.

The test is performed for 100 users, 100 seconds, 0.5 events/s, and a maximum accept count of 10 resp. 1000. Test results are shown in Table 4.13. As the p-value is $0.02877 > 0.01$, the accept count does not significantly influence the response time.

4.3.4 Transmission Packet Sizes

The transmission packet size, describing which protocols are used and how transferred data is encapsulated, is expected to affect the response time. Since the current version of the Gamification Platform uses lightweight RPC calls in JSON objects to transmit data, and minimized transmission times are expected in a non-distributed scenario, this cost factor is not evaluated any further.

4.3.5 RAM of Server

The maximum RAM size of the server is expected to influence the performance. This hypothesis is checked by varying the RAM of the server between 8GB, 1024MB, 256MB, and 64MB. The test is performed for 100 users, 100 seconds, and 0.5 events/s. Test results are shown in Table 4.14. The p-values illustrate that only between 256MB to 64MB a significant change in the distribution can be found.

Fig. 4.4a-c show more precisely that the response time between 256 MB and 8GB of RAM is consistent and exponentially distributed. For 64MB (Fig. 4.4d) RAM, the

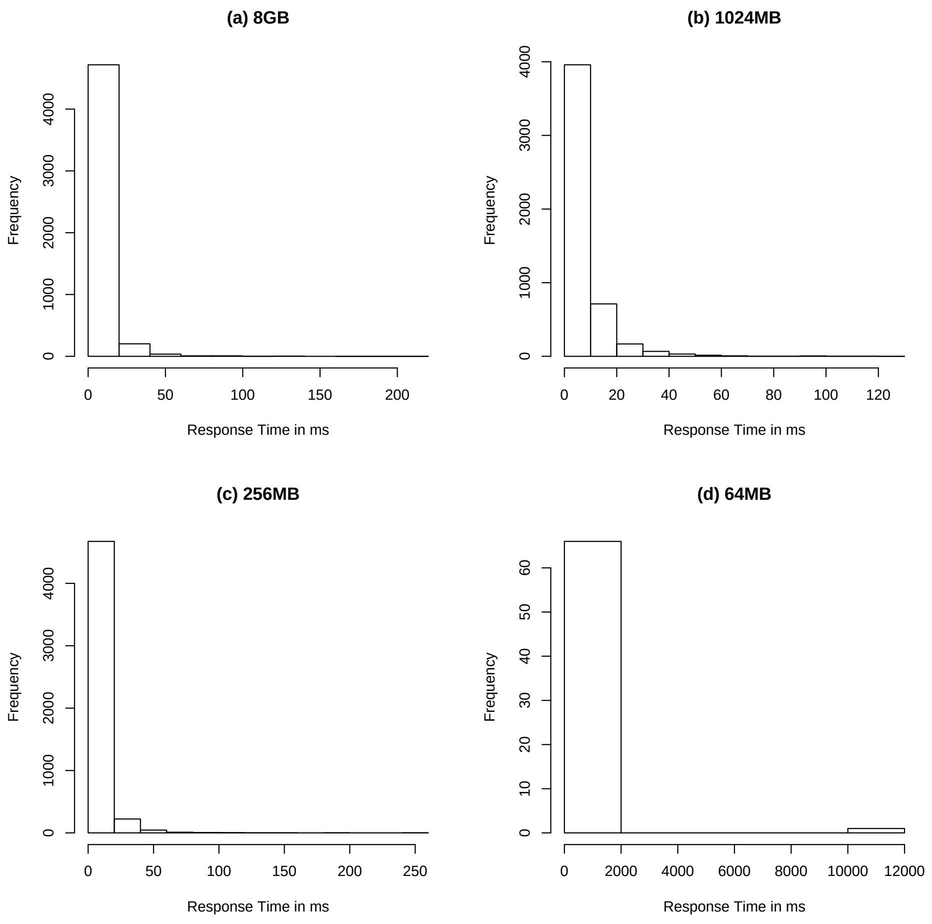


Figure 4.4: Distribution of Response Times for Various RAM Sizes

response time distribution changes, and, more importantly, the total frequency of events has dropped dramatically from 5.000 to around 70. Fig. 4.5 further investigates this effect, showing that for 64MB RAM the number of points is not as high as expected from the other cases. This is a consequence of the server running out of space, which leads to incorrect behavior. As such a behavior has to be avoided by all means, an appropriate RAM size must be ensured. However, besides this impact on the server's behavior, the RAM size does not seem to influence the response time and will, hence, be excluded from the cost model.

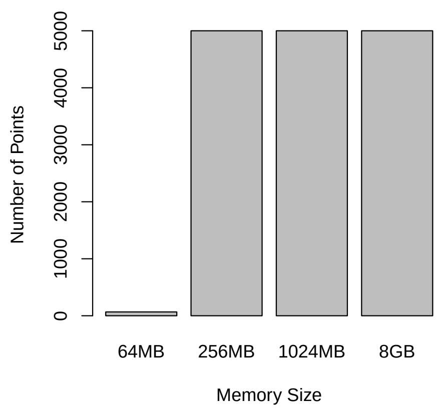


Figure 4.5: Number of Points for Various RAM Sizes

4.4 Statistically Significant Cost Factors

In this chapter, the cost factors

- number of users,
- number of events/s per user,
- event type distribution,
- number of alpha nodes,
- number of beta nodes,
- number of abstractions,
- number of updateAPI calls,
- and number of queryAPI calls

have been proven to have a statistically significant influence on the response time. On the contrary, the cost factors

- node types,
- working memory growth,
- independent rule streams,
- structure of tables,
- and all infrastructure-related cost factors

were discussed to be excluded due to statistical insignificance or because they cannot be varied easily or modeled with the chosen modeling technique.

In the next chapter, the influence of each of the statistically significant cost factors is researched and measured for each component of the queueing network models presented in Chap. 3.

Chapter 5

Experiments

This chapter describes the experiments which are conducted to determine how strong each cost factor correlates with the service times of the components in the queueing networks presented in Sect. 3.2. First, our experiment modeling technique is presented. Subsequently, the experiment design is described, and experiment data as well as experiment workload are presented. Finally, the results of the experiments are discussed in depth for each cost factor in both synchronous and asynchronous mode.

5.1 Experiment Modeling Technique

The experiment results contain multiple values for service times of each component, and for certain levels of each cost factor. In order to determine the influence of a particular cost factor on the service time, the values are modeled in relation to the factor level for each component. These value-pairs can be placed in a coordinate system as shown in Fig. 5.1, where the cost factor *users* is correlated to the means and medians of service time in the *proxy* component. In this case, service times are shown for the component *proxy* in synchronous mode.

In the next step a model to represent the values has to be found, which can also predict unknown values. For this purpose, polynomial regression (Subsect. 2.4.2) is used in combination with the information criteria presented in Subsect. 2.4.3. The best fitted polynomial for the example in Fig. 5.1 is the second degree polynomial, which was chosen by using AIC.

The best fitted polynomial is determined for every component and every cost factor in both modes (synchronous and asynchronous). As defined in Eq. 2.9, the coefficients c_i of the polynomial are listed for each component in a table. Additionally, the information criterion, which was used to choose the most sparse polynomial, is included in the table. An example is given in Table 5.1.

In the first column, the component is listed. In this case, the first row is used for the *CEP* component, and the second row gives service time coefficients for the *proxy*

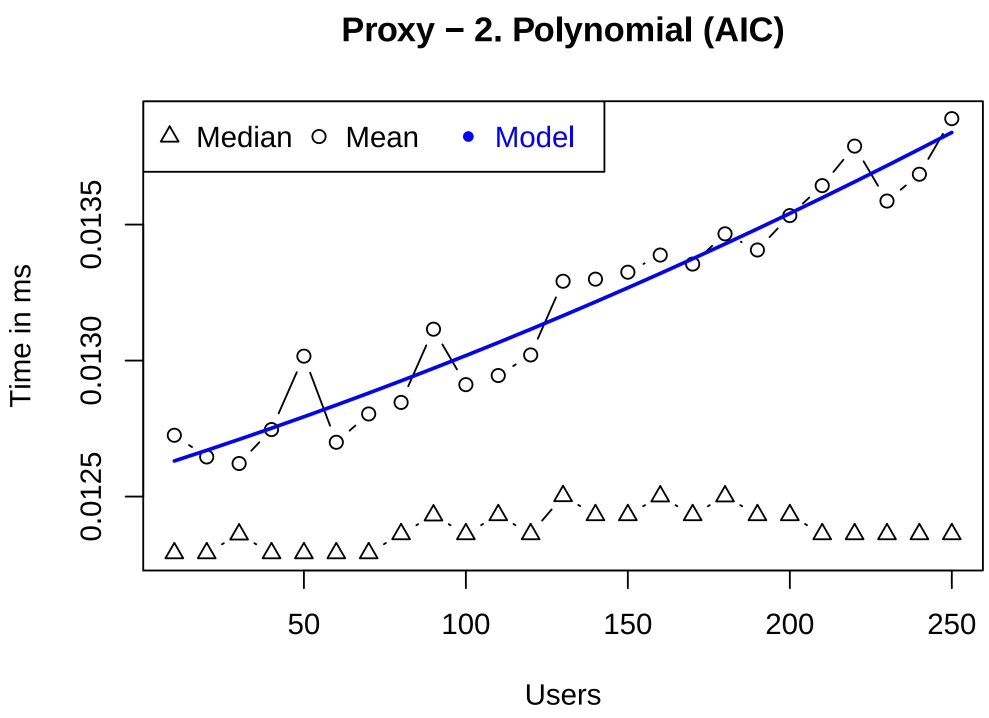


Figure 5.1: Example of Service Times per Cost Factor Level

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$
CEP	All	0.95	0.2***	0.01*	-
Proxy	CV	0.75	$2.8 * 10^{-2}$**	$7.1 * 10^{-5}$.	$2.5 * 10^{-7}$

Table 5.1: Service Time Polynomials for an Exemplary Cost Factor

component. *MB1* and *MB2* are abbreviations for *message broker 1* and *2*, used later in this chapter.

In the second column, the information criterion used to select the polynomial is listed. This can either be *AIC*, *BIC*, or *CV* (cross-validation) as defined in Subsect. 2.4.3. Moreover, combinations like *AIC/BIC* are used if two criteria prefer the same polynomial. If all three criteria prefer the same polynomial, this is noted as *All*.

In the third column, R^2 as the coefficient of determination and model quality is given. An R^2 close to 1 stands for a good model fit, whereas an R^2 close to 0 represents a poor model fit, indicating that the parameters chosen to model the values do not provide an accurate estimation and that other influencing parameters were excluded in the polynomial model.

The other columns list the coefficients $c_0...c_i$ of the polynomial. The values of the polynomials as well as the significance levels are listed. These significance levels are calculated using the t-statistic to determine the corresponding two-sided p-value [The R Foundation for Statistical Computing, 2013c]. The significance levels are defined as follows:

- ***: $p < 0.1\%$
- **: $p < 1\%$
- *: $p < 5\%$
- .: $p < 10\%$
- otherwise: $p \leq 100\%$

Based on this information, the service time s for cost factor level x at a component can be calculated using the polynomial, e.g., for the *CEP* component of the example:

$$s_{CEP}(x) = 0.2 + 0.01x$$

For the *proxy* component of the example, the service time can be calculated with the formula

$$s_{Proxy}(x) = 2.8 * 10^{-2} + 7.1 * 10^{-5} * x + 2.5 * 10^{-7} * x^2.$$

All values are given in ms.

With the formulas, the service times can be calculated for every specific case. If the cost factor level (e.g., the amount of users) is, for example, forecasted to be 100, the service times are

$$s_{CEP}(100) = 0.2 + 0.01 * 100 = 1.2$$

and

$$s_{Proxy}(100) = 2.8 * 10^{-2} + 7.1 * 10^{-5} * 100 + 2.5 * 10^{-7} * 100^2 = 0.0376.$$

The service times can be used to simulate the queueing network as described further in the model application method in Sect. 6.3. Supplementary graphical representations of all polynomials discussed in this chapter are to be found in Appendix A.

5.2 Experiment Design

With the experiment, each cost factor's influence on service time is checked for all components in the queueing networks individually. It is assumed that cost factors of one category (users, rules, or infrastructure) are predominantly independent from cost factors in another category. Additionally, since the time frame of the thesis is restricted, a full factorial design combining all cost factors and all levels of factors in a cross product is unmanageable. For only 5 cost factors with each having 10 levels the number of experiments would be 10^5 based on Eq. 2.1. For cost factors, which are assumed to influence each other, the relationship will have to be investigated in further research.

Wrong or inaccurate measurements caused by *Garbage Collection*, *Dead Code Elimination* and *Dynamic Optimization* as mentioned by Boyer [2008a] while taking the measurements are tried to be kept at minimum. This is achieved by measuring enough data to detect outliers, which could be caused by JVM behavior. Additionally, suggestions to avoid typical pitfalls are applied as discussed in Boyer [2008a,b]. For example, we ignore the first 10 seconds of each experiment, as this is considered to be the warm-up period. Furthermore, a test with 200 users is run for 300 seconds before starting measurements to initialize all classes and allow optimizations to take place. Each experiment is run on a machine with two 6-core Intel Xeon L5640 processors in hyper-threading mode and 8GB of RAM.

5.3 Experiment Data

The experiment environment is set up with two rules (List. 5.1) to measure the service times. The first rule, *newBuddy* (line 1-10), reacts to events of type *addBuddy*. If such an event occurs, the *updateAPI* is called to add one *Buddies* point to the users' points. The event is then retracted and an update is sent to the player object in synchronous mode. This update starts a reevaluation of all rules using the player as input. In asynchronous mode, the update is not required as the players' data is updated internally in the CEP and, thus, automatically triggers a reevaluation of all rules depending on the player. The second rule, *tenBuddies* (line 12-20), reacts to every update of a player. If the mission *I Have Got Buds!* is assigned to the player and he has more than 10 *Buddies* points, the mission is completed. Afterwards, an update is sent to the player object in synchronous mode, for the same reasons as in the *newBuddy* rule explained above.

To trigger the rules, events of type *addBuddy* are sent to the system. Each event will trigger the rule *newBuddy* to run once. The player is updated in line 9 to trigger an evaluation of the second rule (*tenBuddies*) thereafter. If both the expected mission is assigned to the player and he has enough points, the mission is completed and with the update of the player, the LHS of the rule is evaluated once again.

To test the number of alpha nodes, rules similar to the rule in List. 4.3 are used, where each alpha node is represented by a separate rule. In this experiment, events of each event type (*addBuddy1*, *addBuddy2*, ...) are sent to the system by each user with the same probability. The influence of the number of beta nodes is experimentally tested with rules similar to the rule in List. 4.4, where, again, each beta node is created by a distinct rule. Events are sent in pairs to the system in this experiment, e.g., one *addBuddy1* and one *addTag1* event. Rules similar to the rule in List. 4.5 are deployed to test the number of abstractions. Each user only creates *addBuddy1* events in this experiment.

```
rule "newBuddy"
  when
    $addBuddy : EventObject(type=='addBuddy', $playerid:playerid) from entry-
     point eventstream
  then
    updateAPI.givePoints($playerid, 'Buddies', 1, 'added 1 buddy');
    retract($addBuddy);
    update(queryAPI.getPlayer($playerid)); //only in synchronous mode
end

rule "tenBuddies"
  when
    p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'I Have Got Buds!') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Buddies').getAmount() >= 10)
  then
    updateAPI.completeMission($playerid, 'I Have Got Buds!');
    update($p); //only in synchronous mode
end
```

Listing 5.1: CEP Rules

5.4 Workload

The workload is determined by the number of users. Each user sends m events per second to the system. In total, the number of events can be described as

$$n \text{ user} \times m \frac{\text{event}}{\text{second} \times \text{user}} \times 300 \text{ seconds} = 300 \times n \times m \text{ events}$$

with n as the number of users. $n = 250$ and $m = 0.5$ are chosen as default values. Accordingly, the default number of events is 37,500 for a duration of 300 seconds. Therefore, the interarrival rate is approximately exponentially distributed with sample mean at 8ms as shown in Fig. 5.2.

Of course, it is hard to imagine that every user creates an event every 2 seconds in a real-world application. These numbers are consciously exaggerated to, for example, simulate the behavior for multiple applications operating on the platform with each having distinct users and rule sets. Additionally, the event rates can be converted into other units. For the example above the event rate can be converted as follows:

$$125 \text{ Events/s} = 450,000 \text{ Events/h} = 10,800,000 \text{ Events/day}$$

If not only 250 users are registered, but rather 250,000, every user would send on average 1.8 events per hour. Since the threading of so many user objects would create a significant overhead in the experiments, the high amount of events is created by only a few users, but the results may be extrapolated to other amounts of users and event rates.

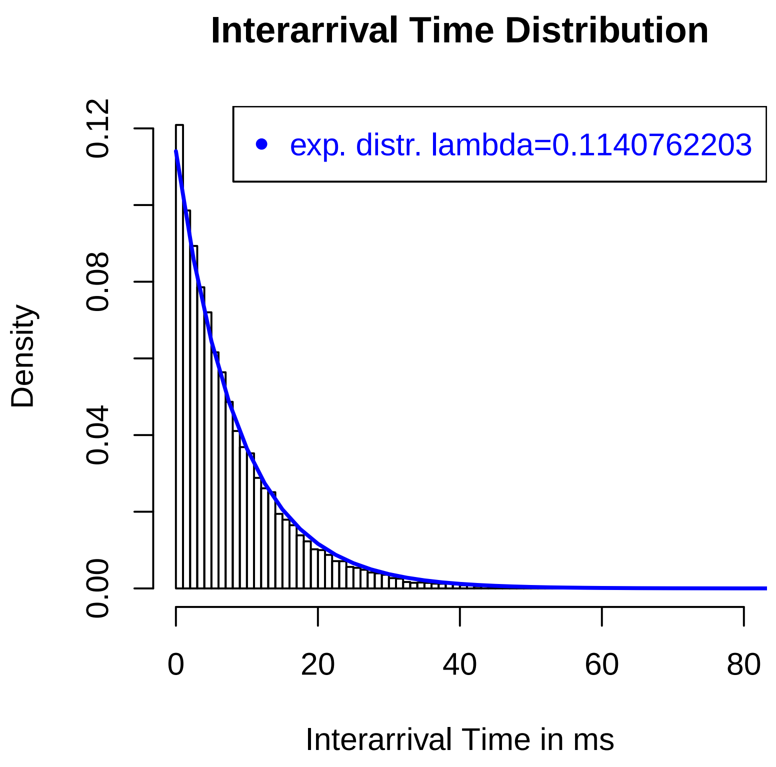


Figure 5.2: Interarrival Time Distribution for 250 User and 0.5 Events/s/User

5.5 Experiment Results

This section presents the results of the experiment series. At first, the distributions of the service times are investigated. Next, the cost factors identified in Chap. 4 are measured in both synchronous and asynchronous mode. In synchronous mode, the influence of all cost factors on service times of the components CEP, proxy, and BEP (query and update) as introduced in Subsect. 3.2.1 are presented. In asynchronous mode, the influence of all cost factors on the service times of message broker, CEP, proxy, CEP context (query and update), and BEP (update) as introduced in Subsect. 3.2.2 are considered.

5.5.1 Distributions

The distributions of the service times in the components are analyzed first. Graphical examples showing the distributions of service times for all involved components are presented in Fig. 5.3 for synchronous mode and in Fig. 5.4 for asynchronous mode.

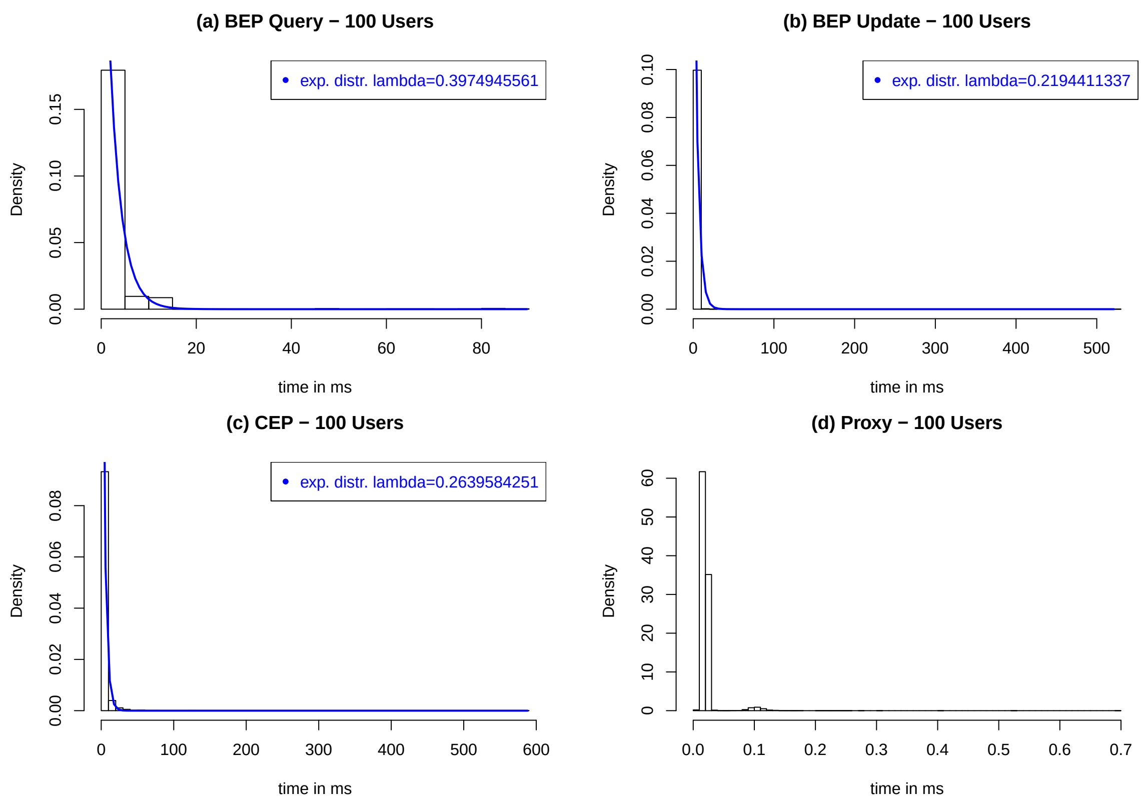


Figure 5.3: Distribution of Service Times in Components in Synchronous Mode

Synchronous Mode

In synchronous mode, all distributions tend to consist of many small values and a few larger values. The larger the value, the smaller its probability. As a consequence, the distributions can be described with an exponential distribution, which is defined as

$$f(x) = \lambda e^{-\lambda x}.$$

The mean value of the measured service times is then used as the mean value of the exponential distribution, whereby

$$\lambda = 1/\text{mean}(\text{service times})$$

can be calculated.

As shown in Fig. 5.3, the service times of the components *BEP query* (a), *BEP update* (b) and *CEP* (c) are best modeled with exponential distributions. However, for the *proxy* component (d) the service time is best represented by a constant value defined by the observed mean, for two reasons. First, the distribution is too inaccurate. Second, this component's absolute values are very low compared to the other

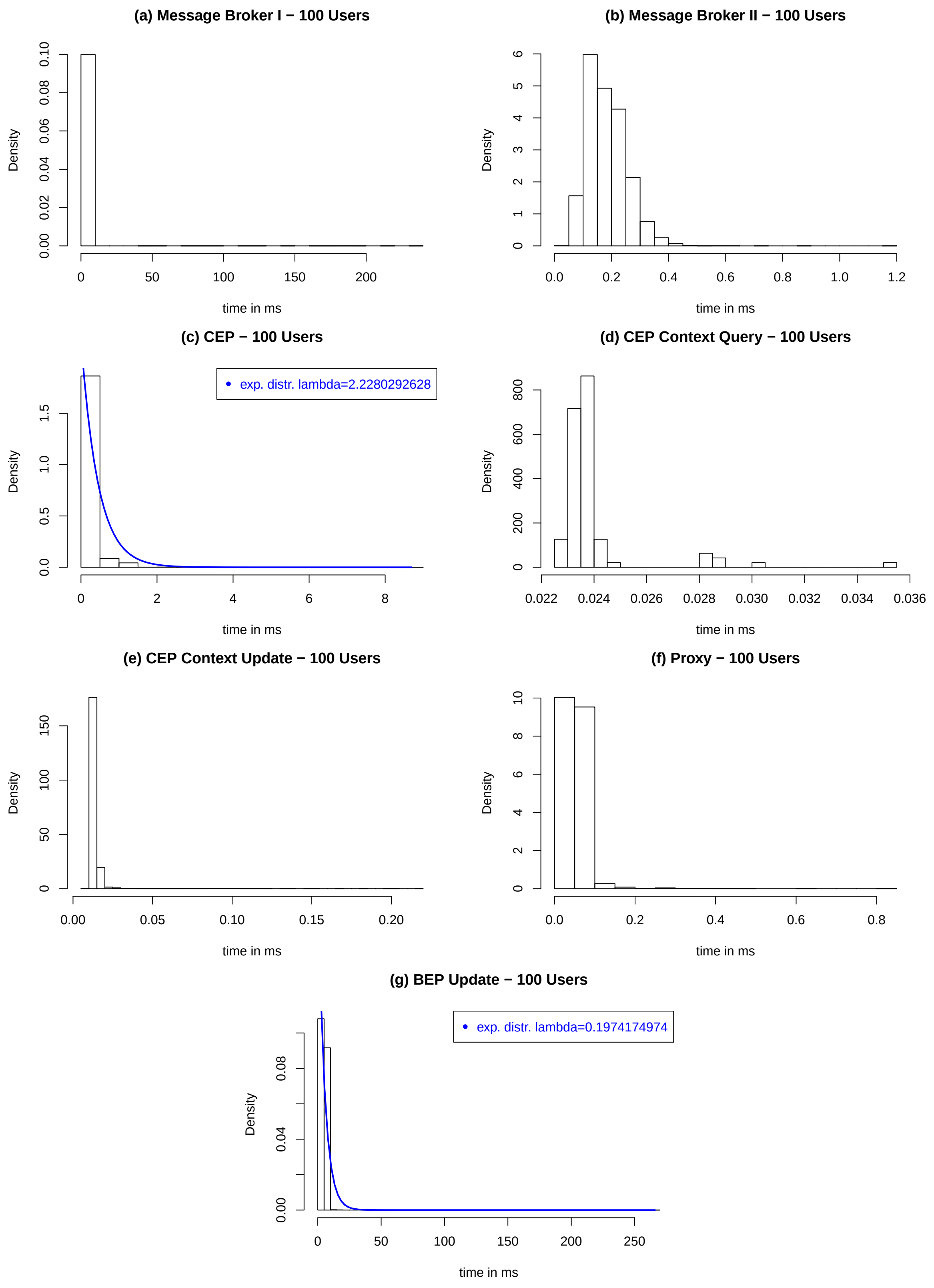


Figure 5.4: Distribution of Service Times in Components in Asynchronous Mode

components. Therefore only little deviation will be produced by using a constant service time for the proxy component.

For each distribution, a few outliers with values multiple magnitudes larger than median and mean value can be observed. This behavior is also taken into account by modeling the service time distributions exponentially. A great number of experiment values will cause these rare outliers with extreme values to appear.

Asynchronous Mode

In Fig. 5.4, the distributions of service times in the components are shown for the asynchronous mode. Compared to the synchronous mode, distributions cannot as easily be described with exponential distributions. For example, *message broker II* (b) and *CEP context query* (d) show a right-skewed, but somehow arbitrary distribution. As the values are negligible compared to the others, a constant or arbitrary (general) distribution should be chosen to model the service time distribution as realistically as possible. *Message broker I* (a), *CEP context update* (e) and *proxy* (f) component also show service times that are difficult to generalize, i.e., for which an underlying distribution is hard to find. But they also consist of many small and only a few large values. Hence, these components can be modeled using exponential distributions, also with regard to the outliers produced. Moreover, *CEP* (c) and *BEP update* (g) components can also be modeled using exponential distributions.

5.5.2 Cost Factor Number of Users

The number of users has been determined in Subsect. 4.1.1 to significantly influence the total response time. Hence, this cost factor is analyzed more precisely for synchronous as well as for asynchronous mode.

Synchronous Mode

In the histograms of the service times (Fig. 5.5) a significant visual change of distribution can be observed already for service times in the *CEP* component at 300 users. A closer examination of the experiment data reveals that for more than 130 users the system is overloaded, whereas below this threshold the performance curve is relatively constant. In stable state, the distribution can be characterized as exponentially distributed, but above 130 users many extreme outliers can be observed. This means that the component is overloaded and cannot handle the responses immediately, while the queue of the component is constantly growing. Based on this fact, the cost factor user the synchronous mode is modeled based on experiment data up to 130 users. The overload case has to be investigated more precisely in further research.

The results are service times as functions of the number of users, as presented in Table 5.2. Clearly, the *CEP* component is influenced most by the number of users

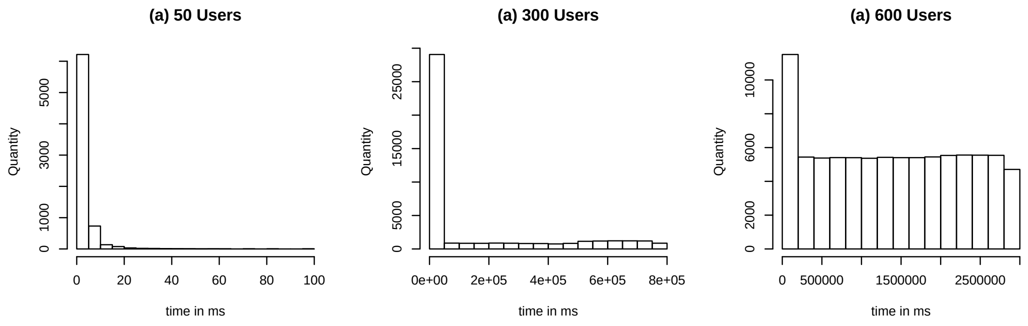


Figure 5.5: Histograms CEP Service Times Users Synchronous Mode

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$
CEP	CV	0.7977	-0.72436	0.06647***	-
Proxy	CV	0.8325	$2.358 * 10^{-2}$***	$-7.158 * 10^{-5}$**	$2.751 * 10^{-7}$.
BEP Query	BIC/CV	0.9694	1.1439854	0.0124863***	-
BEP Update	All	0.257	4.924098***	-0.003896.	-

Table 5.2: Service Time Polynomials for Cost Factor Number of Users in Synchronous Mode

with $c_1 = 0.06647$. *BEP* (query more than updates) and *proxy* components are influenced to a lesser extent, both having having a $c_1 < 0.015$. All polynomial models can be built with first degree polynomials, except for the proxy component, which is built with a second degree polynomial. *BEP update* component has a low $R^2 = 0.257$, since the measured values show high variations and no obvious trend.

Asynchronous Mode

When operating in asynchronous mode, the stable state can be measured up to 650 users. The results and service times as functions of the number of users are shown in Table 5.3. The most time-consuming and therewith most strongly influenced component is the *BEP Update* having a coefficient $c_1 = 0.06209$. All other components are influenced to a lesser extent, with c_1 and c_2 smaller than 0.002. All polynomials are of first or second degree. As illustrated by the graphs (Fig. A.2), the variations of the measured values are high for all components except *message broker 1* and *CEP*, which is also represented by the low R^2 values ($R^2 < 0.35$) for these components.

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$
MB1	All	0.9743	$7.703 * 10^{-1}$***	$1.476 * 10^{-3}$*	$1.175 * 10^{-6}$
CEP	All	0.9711	$4.884 * 10^{-1}$***	$-2.101 * 10^{-4}$	$1.897 * 10^{-6}$***
Proxy	AIC/BIC	0.00037	$3.138 * 10^{-2}$***	$2.127 * 10^{-7}$	-
CEP Query	All	0.1653	$2.069 * 10^{-2}$***	$-3.975 * 10^{-6}$	-
CEP Update	All	0.1269	$1.654 * 10^{-2}$***	$2.611 * 10^{-6}$	-
MB2	All	0.04496	$2.051 * 10^{-1}$***	$1.818 * 10^{-5}$	-
BEP Update	All	0.3435	-5.63988	0.06209*	-

Table 5.3: Service Time Polynomials for Cost Factor Number of Users in Asynchronous Mode

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$
CEP	CV	0.9858	88.01***	-467.59***	660.82***
Proxy	CV	0.1957	0.016465***	-0.004503	-
BEP Query	BIC	0.9985	0.3772	130.3701***	-
BEP Update	All	0.06802	4.0061***	0.2885	-

Table 5.4: Service Time Polynomials for Cost Factor Number of Events Per Second Per User in Synchronous Mode

5.5.3 Cost Factor Number of Events Per Second Per User

The number of events per second triggered per user has been determined in Subsect. 4.1.2 to significantly influence the total response time. Thus, the cost factor is analyzed more precisely in the following.

Synchronous Mode

For the synchronous mode, the stable state can be observed up to 0.52 events/s/user with 250 users, and only values below that threshold are taken into account for creating the polynomial models. The results and service times as functions of the number of events per second per user are shown in Table 5.4.

All polynomials are of first degree except the polynomial for the *CEP* component, which is of second degree. The absolute values of service times of *CEP* and *BEP* components are influenced most by this cost factor. *CEP* component has coefficients $c_1 = -467.59$ and $c_2 = 660.82$ in the second degree polynomial, and *BEP query* component has coefficient $c_1 = 130.3701$. The *proxy* component is influenced less by this cost factor, even having a negative coefficient, $c_1 = -0.004503$, and also having a low model quality with $R^2 = 0.06802$.

Component	**Inf. Cri.**	**R^2**	**c_0**	**c_1**	**c_2**
MB1	All	0.7553	−12.517***	39.926***	-
CEP	CV	0.979	63.67***	−301.30***	364.22***
Proxy	AIC/BIC	0.8843	0.022798***	0.042260***	-
CEP Query	All	0.001079	0.019709***	0.009878***	-
CEP Update	All	0.6752	0.005124	0.034046***	-
MB2	CV	0.8243	−4.5173***	15.5789***	-
BEP Update	CV	0.9861	16.317**	−73.881***	118.072***

Table 5.5: Service Time Polynomials for Cost Factor Number of Events Per Second Per User in Asynchronous Mode

Asynchronous Mode

In asynchronous mode, the system is relatively stable up to 0.76 events/s/user for 250 users, whereas all other values have been excluded from creation of the polynomial models. The results and service times as functions of the events per second per user are shown in Table 5.5. Only *CEP* and *BEP update* component can be modeled best using second degree polynomials, all other components can be represented by using first degree polynomials. Again, the *CEP* component is influenced strongly by this cost factor with coefficients $c_1 = -301.30$ and $c_2 = 364.22$ in a second degree polynomial, but also both *message broker* components and the *BEP* component are influenced to a large extent with coefficient values greater than 15. The quality of the models is good for the most components having $R^2 > 0.75$, only *CEP query* and *CEP update* component show smaller values: $R^2 = 0.001079$ for *CEP query* and $R^2 = 0.6752$ for *CEP update*.

5.5.4 Cost Factor Event Type Distribution

The event type distribution does not influence the service times of the components, but the routing of events through the system. Only for join nodes, the service time of the *CEP* component is dependent on the event type distribution. But as queueing networks are stateless, the service time cannot be increased if many events of the same type are stored in the component.

Events of different types can be routed through the queueing network in different ways depending on the LHS (queryAPI calls) and RHS (updateAPI calls) of the rules. For example, one rule might simply update the context depending on an event of type *addBuddy*. Another more complex rule might use an event of type *attendedMeeting* and several *queryAPI* calls to determine whether the LHS is fulfilled, and if all conditions are met, the BEP is updated. Therefore two distinct routings through the system are needed in order to describe the behavior of different event types accordingly. Events of type *addBuddy*, on the one hand, should only pass the *BEP* without even triggering a queryAPI call in the *BEP* component. On the other hand, events of type *attendedMeeting* circle through the *BEP* component for each *queryAPI* call to determine the context and to decide whether a *BEP* update is necessary. Only if the condition is fulfilled, the *updateAPI* is called once for this rule.

To model the distinct behavior of events of different types, several classes have to be used in the queueing network. In the above example, one class would be used to model *addBuddy* events and their routing and service times, and another class to represent *attendedMeeting* events and their routing as well as service times. Interarrival times must be chosen for every class independently, so that the different frequencies of events occurrences can be mapped accurately and the event type distribution is considered. Routing of events can only be mapped using probabilities in queueing networks, for example, the event behaving with a probability of 25% like a *queryAPI* call to the *BEP* component and with 75% like an *updateAPI* call, which is being routed to the sink afterwards.

In summary, this cost factor can be modeled using one class for each event type and setting interarrival times and distributions as well as probabilities for routing for each of the defined classes appropriately. To model this cost factor, no adaptations of service times are necessary, the total response time is rather compounded by multiple visits to the components with each visit requiring time.

5.5.5 Cost Factor Number of Alpha Nodes

The number of alpha nodes has been determined in Subsect. 4.2.2 to influence the total response time. Thus, the cost factor is analyzed more precisely for synchronous as well as for asynchronous mode in the following.

Synchronous Mode

For up to 200 alpha nodes the system performs in stable state. Above this threshold, the system is overloaded and becomes unpredictable. Hence, only values up to 200 alpha nodes are considered to construct the polynomial models. The results and service times as functions of the number of alpha nodes are shown in Table 5.6. Service times in *BEP query* and *BEP update* components are both of first degree, in the *CEP* component the service times are modeled best with a second degree polynomial, and for the *proxy* component a third degree polynomial is used.

As expected, the number of alpha nodes only has slight to none influence on the *proxy* and *BEP* components. The only crucially influenced component is the *CEP* component, in which the alpha node validation and activation of the according rules takes place. For this component, the coefficients are $c_1 = -1.511596$ and $c_2 = 0.015103$. The quality of all models is poor, only the *CEP* component has a reasonable $R^2 = 0.7923$, while all other components have an $R^2 < 0.5$.

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$
CEP	All	0.7923	292.226434***	-1.511596	0.015103**
Proxy	CV	0.4856	$2.055 * 10^{-2}$***	$-1.167 * 10^{-5}$*	$1.002 * 10^{-7}$.
BEP Query	All	0.08521	116.74004***	0.01379	-
BEP Update	All	0.2699	4.5995978***	-0.0004797*	-
Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_3}$	$\mathbf{c_4}$	$\mathbf{c_6}$
Proxy	CV	0.4856	$-2.666 * 10^{-10}$.	-	-

Table 5.6: Service Time Polynomials for Cost Factor Number of Alpha Nodes in Synchronous Mode

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$
MB1	All	0.8266	4.408*	$-1.306 * 10^{-2}$	$3.039 * 10^{-5}$**
CEP	All	0.8379	5.867***	$-1.318 * 10^{-2}$**	$2.130 * 10^{-5}$***
Proxy	CV	0.8516	$4.166 * 10^{-2}$***	$4.255 * 10^{-5}$***	-
CEP Update	BIC/CV	0.3584	$1.716 * 10^{-2}$***	$1.536 * 10^{-5}$**	-
MB2	CV	0.6701	-1.134	$1.213 * 10^{-2}$**	$-7.981 * 10^{-6}$.
BEP Update	CV	0.6547	5.755990***	0.006609**	-

Table 5.7: Service Time Polynomials for Cost Factor Number of Alpha Nodes in Asynchronous Mode

Asynchronous Mode

The system performs in a stable state up to 950 alpha nodes, therefore these results have been used to create the polynomial models. The results and service times as functions of the number of alpha nodes are shown in Table 5.7. Service times in *proxy*, *CEP update* and *BEP update* are modeled by using first degree polynomials, and both *message broker* components as well as *CEP* component are using second degree polynomials.

All components are only slightly influenced compared to the synchronous mode, having coefficients c_1 and c_2 smaller than 0.2. The most influenced component is in this case, again, the CEP component, having coefficients $c_1 = -1.318 * 10^{-2}$ and $c_2 = 2.130 * 10^{-5}$. The polynomial models for *message broker 1*, *CEP*, and *proxy* component show a good quality with $R^2 > 0.8$, *message broker 2* and *BEP update* are still acceptable with $R^2 > 0.65$. Only the *CEP update* component is badly modeled ($R^2 = 0.3584$), which is caused by the variations in the measurements (Fig. A.6).

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$
CEP	CV	0.7789	-24.0026	0.7879***
Proxy	All	0.04047	$2.239 * 10^{-2}$***	$-2.903 * 10^{-7}$
BEP Query	AIC/BIC	0.0001058	90.04***	$1.144 * 10^{-4}$
BEP Update	All	0.04326	4.6182616***	0.0001678

Table 5.8: Service Time Polynomials for Cost Factor Number of Beta Nodes in Synchronous Mode

5.5.6 Cost Factor Number of Beta Nodes

The number of beta nodes has been determined in Subsect. 4.2.3 to influence the total response time. Accordingly, the cost factor is analyzed more precisely for synchronous as well as for asynchronous mode.

Synchronous Mode

In synchronous mode, the system is relatively stable up to 300 beta nodes, for which reason all other values have been excluded for the creation of the polynomial models. The results and service times as functions of the beta nodes are shown in Table 5.8. All service time polynomials are first degree polynomials for this cost factor. The component most influenced by the cost factor is, as expected, the *CEP* component having a coefficient $c_1 = 0.7879$. This component is used to process the joins in the beta nodes. From the other components only the *BEP update* is influenced noticeably, it is modeled with coefficient $c_1 = 0.0001678$. Additionally, the quality of the models is poor with $R^2 < 0.05$. Only the *CEP* component, which is the most important component for this cost factor, can be modeled accurately ($R^2 = 0.7789$).

Asynchronous Mode

In asynchronous mode, the system is relatively stable up to 600 beta nodes, and only values below this threshold are taken into account for creating the polynomial models. Results and service times as functions of the beta nodes are shown in Table 5.9. Only *proxy* and *BEP update* component service times can be modeled using first degree polynomials, all other components are modeled best by using second degree polynomials. This cost factor has the largest influence on *CEP*, having coefficients $c_1 = -6.468 * 10^{-3}$ and $c_2 = 2.242 * 10^{-5}$), and also on the *BEP update* component with coefficient $c_1 = 0.0052295$.

Component	**Inf. Cri.**	**R^2**	**c$_0$**	**c$_1$**	**c$_2$**
MB1	All	0.759	1.769*	$-1.081 * 10^{-2}$.	$3.717 * 10^{-5}$***
CEP	All	0.6552	$8.562 * 10^{-1}$	$-6.468 * 10^{-3}$	$2.242 * 10^{-5}$**
Proxy	All	0.2648	$4.011 * 10^{-2}$***	$8.287 * 10^{-6}$**	-
CEP Update	AIC/BIC	0.2357	$1.531 * 10^{-2}$***	$3.145 * 10^{-6}$*	$-4.417 * 10^{-9}$*
MB2	AIC/BIC	0.9401	$1.738 * 10^{-1}$***	$1.736 * 10^{-4}$*	$3.652 * 10^{-7}$**
BEP Update	BIC	0.5193	4.5140333***	0.0052295***	-

Table 5.9: Service Time Polynomials for Cost Factor Number of Beta Nodes in Asynchronous Mode

Component	**Inf. Cri.**	**R^2**	**c$_0$**	**c$_1$**	**c$_2$**
CEP	CV	0.9993	3.343	$5.924 * 10^{-1}$***	$-1.607 * 10^{-4}$
Proxy	BIC	0.7377	$3.177 * 10^{-2}$***	$-2.367 * 10^{-5}$***	-
BEP Query	All	0.5205	8.192***	$-4.470 * 10^{-3}$**	$5.113 * 10^{-6}$*
BEP Update	AIC/BIC	0.9605	$1.045 * 10^{1}$***	$-1.669 * 10^{-2}$***	$1.564 * 10^{-5}$***
Component	**Inf. Cri.**	**R^2**	**c$_3$**	**c$_4$**	**c$_5$**
CEP	CV	0.9993	$5.637 * 10^{-7}$***	-	-

Table 5.10: Service Time Polynomials for Cost Factor Number of Abstractions in Synchronous Mode

5.5.7 Cost Factor Number of Abstractions

The number of abstractions has been determined in Subsect. 4.2.5 to influence the total response time. Accordingly, the cost factor is analyzed more precisely for synchronous and asynchronous mode.

Synchronous Mode

The system performs stable up to 580 abstractions, and all measured values below this value have been used to create the polynomial models. The results and service times as functions of the number of abstractions are shown in Table 5.10. For abstractions, the *CEP* component is, again, the biggest cost driver having coefficients $c_1 = 5.924 * 10^{-1}$, $c_2 = -1.607 * 10^{-4}$, and $c_3 = 5.637 * 10^{-7}$ in a third degree polynomial. Service times in *BEP query* and *BEP update* components are modeled using second degree polynomials, and for the *proxy* component a first order polynomial is used. *CEP* and *BEP update* components are adequately modeled having $R^2 > 0.95$, the proxy component has a reasonable quality of $R^2 = 0.7377$, and the *BEP query* components' quality is $R^2 = 0.5205$ due to the variations observable in the graphs (Fig. A.9).

Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$
MB1	All	0.5809	1.632***	$-7.775 * 10^{-4}$***	$1.588 * 10^{-6}$***
CEP	All	0.9993	5.377***	$2.673 * 10^{-1}$***	$2.904 * 10^{-4}$***
Proxy	AIC/BIC	0.07004	$1.095 * 10^{-1}$***	$6.629 * 10^{-5}$	-
CEP Update	All	0.9833	$8.362 * 10^{-2}$***	$-9.313 * 10^{-5}$	$1.627 * 10^{-6}$***
MB2	All	0.8214	$4.069 * 10^{-1}$***	$1.080 * 10^{-3}$***	$-1.356 * 10^{-6}$***
BEP Update	All	0.6684	8.6175269***	−0.0044881***	-
Component	**Inf. Cri.**	$\mathbf{R^2}$	$\mathbf{c_3}$	$\mathbf{c_4}$	$\mathbf{c_5}$
CEP Update	All	0.9833	$-3.873 * 10^{-9}$**	$2.903 * 10^{-12}$**	-

Table 5.11: Service Time Polynomials for Cost Factor Number of Abstractions in Asynchronous Mode

Asynchronous Mode

In asynchronous mode, the system also performs in a stable state for up to 580 abstractions. All values below have been used to create the polynomial models for this cost factor. Service times as functions of the number of abstractions are shown in Table 5.11. Only the *CEP context update* component is modeled with a polynomial of forth degree, all others are modeled either in second degree (both *message broker* and *CEP* components) or in first degree (*proxy* and *BEP update* components). Similar to the synchronous case, the CEP component is influenced most, the polynomial model has coefficients $c_1 = 2.673 * 10^{-1}$ and $c_2 = 2.904 * 10^{-4}$. The second most influenced component is the BEP component, having a negative coefficient $c_1 = -0.0044881$. The *CEP* and *CEP update* components are modeled with a good quality ($R^2 > 0.95$), whereas both *message brokers* and the *BEP update* component have an acceptable quality with $R^2 > 0.55$. Only the *proxy* component is poorly modeled because of several variations ($R^2 = 0.07004$).

5.5.8 Cost Factor Number of updateAPI calls

The number of updateAPI calls has been determined in Subsect. 4.2.8 to influence the total response time. To consider the cost factor in the cost model, it is now discussed further. As the number of updateAPI calls does not change the service time of particular components, but rather the paths that requests take through the queueing networks, the visits to the stations and the routing need to be adapted to model this cost factor adequately.

In synchronous mode, each updateAPI call is one visit to the *BEP* component. In the asynchronous case, where both the *BEP* and local context in the *CEP* need to be updated, the request is duplicated and sent to *CEP context* component and *BEP* component. Each update of the context triggers a reevaluation of the CEP if rules are based on the previously updated context. For example, if a rule triggers

100 updateAPI calls, each update is done separately. In synchronous mode the *BEP* component would be visited 100 times, and thus taking 100 times longer than a request performing only one updateAPI call. Queueing networks do not provide a mechanism to let the requests traverse through paths in loops, only probabilities for each path can be set. Hence, requests repeatedly circulating through the *BEP* before they leave the system are hard to simulate. The higher the number of loops, the bigger the variance in the queueing network, which leads to more inaccurate total response times. Nevertheless, based on experience from former use cases, rules with a high number of updateAPI calls were not required so far and are expected to be rarely used. For smaller numbers of updateAPI calls the mapping with probabilities seems to be accurate enough.

5.5.9 Cost Factor Number of queryAPI calls

The number of queryAPI calls has been determined in Subsect. 4.2.9 to influence the total response time. The cost factor is therefore discussed in more detail. For the number of queryAPI calls, the same observations as for the updateAPI calls hold accordingly. Generally, the behavior can be mapped only by using probabilities. Hence, a distinct number of updates can not be described in the queueing network.

5.5.10 Summary

In this chapter, experiments to determine the influence of cost factors were described and polynomial models for cost factors

- number of users
- number of events per second
- number of alpha
- number of beta nodes,
- and number of abstractions

have been presented. Additionally, a possible solution for integrating the cost factors

- event type distribution
- number of updateAPI calls,
- and number of queryAPI calls

has been discussed. Within the next chapter, the cost model is presented, using the experimental results gained in this chapter. The required model inputs and the model application method are explained, so that an estimation of response time and throughput can be made.

Chapter 6

Cost Model

This chapter summarizes the modeling method and the resulting cost model, for which the cost factors have been identified and modeled as polynomial functions in the previous chapter. Furthermore, the last section of this chapter explains how to apply the cost model to estimate the response time for a specific use case.

6.1 Modeling Method

The cost model has been derived in two steps. First, the queueing networks based on the structure and architecture of the Gamification Platform have been defined in Sect. 3.2. To predict response times with the generated queueing networks, the service times and service time distributions of each component used in the queueing networks were determined by performing experiments. Subsequently, the experiment results were used to model the influence of each cost factor on service times of each component and mode with polynomial regression (Sect. 5.5) in combination with information criteria AIC, BIC and cross-validation.

6.2 Cost Model

The overall approach and connections between required and delivered inputs are illustrated in Fig. 6.1. The required inputs to describe the queueing network (as listed in Subsect. 2.2.4) are shown as rounded rectangles. Models are depicted as gray rectangles. The dotted shapes represent inputs which can be delivered based on either the architecture or the use case. From these inputs, the required inputs for the queueing network models can be calculated or provided directly.

For each cost factor, a polynomial model is delivered based on the experimental measurements. With these polynomial models and the input of a specific use case, the service times can be calculated, which, in turn, are used in the queueing network model. If all input values needed in the queueing network models are available,

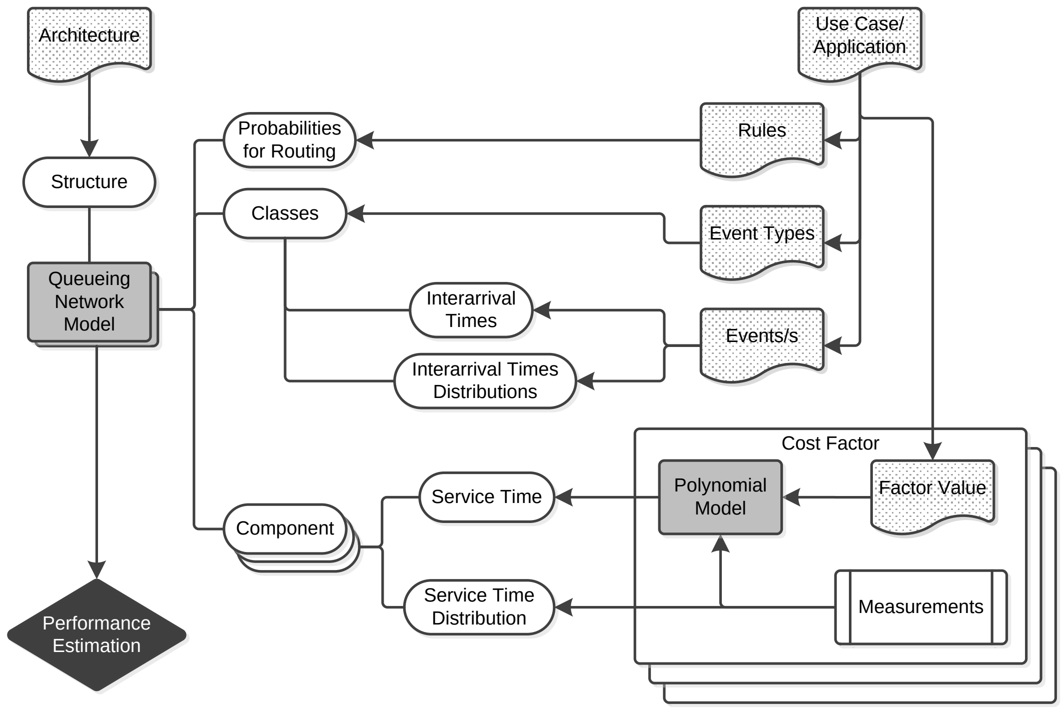


Figure 6.1: Schematic Overview of Cost Model

the performance in terms of response time and throughput can be calculated or simulated, e.g., by using the queueing network simulation program *JSIMgraph*.

The architecture is one of the two main inputs. The architecture of the Gamification Platform determines the structure of the queueing network models. A system analysis (Chap. 3) is used to gain insights into the system and to determine the important components of the Gamification Platform, which are used in the queueing networks. The architecture is transferred into two queueing networks, one for synchronous and one for asynchronous communication between the components.

Classes are defined by the types of events which are sent by the application, whereby each event type is represented by one class. Each class has its own service times, interarrival times, and probabilities for routing. The interarrival time of a class (time between two events of the same type) as well as the interarrival time distribution is given by the application's expected event rate. Probabilities for routing that define which route an event takes through the queueing network, are determined based on the rules.

Loops such as multiple visits at one component, e.g., at the *BEP* or *CEP Context* component for multiple queryAPI calls, are problematic to illustrate correctly in queueing networks. Rules whose behavior changes over time, e.g., only the first ten events of a type trigger an updateAPI call, also cannot be modeled correctly in queueing networks as probabilities defined for classes remain constant over time.

Queueing networks perform independently of former events. Join nodes, for example, are dependent on previous events and usually slow down as more events are already saved in left and right memories of the join node. This behavior cannot be accurately represented in the queueing network, too.

6.3 Model Application Method

The cost model has been developed to cover several scenarios or use cases in estimating response time and throughput. To generate an estimation, certain steps need to be performed.

First, the values of all cost factors need to be forecasted. For instance, the expected number of users has to be chosen, e.g., 200 users are expected to use the application regularly. Each cost factor has to be forecasted as accurately as possible. Of course, it is also possible to compare two different settings by varying values and performing all following steps twice.

Furthermore, it has to be decided whether the Gamification Platform should be used in synchronous or asynchronous mode. Usually, the response time in asynchronous case is faster, but as the context is duplicated, this is achieved at the expense of an increased memory space consumption.

The forecasted values have to be inserted into the polynomial formulas defined in Sect. 4 to estimate service times for each component. As for each cost factor a different value is estimated, these values need to be combined. For example, for 200 users the service time of the *CEP* component is expected to be 18ms. For the next cost factor, e.g., the event rate of 450 events/hour, the service time of the *CEP* component is expected to be 5ms. Since the base line is set at 250 users and 0.5 events/s, the cost factors have to be combined using these values.

The adaptation factor can be calculated by comparing the predicted value to the base line value, e.g.

$$18\text{ms}/20\text{ms} = 0.9$$

for the cost factor user. With this adaptation factor, the results from other estimations, which are based on different cost factors, can be adapted. For example, with cost factor events/s/user the estimated value is 5ms. By multiplying this estimation value with the adaptation factor, an estimation result can be calculated including both cost factors:

$$0.9 * 5\text{ms} = 4.5\text{ms}$$

Alternatively, the same approach can be used starting with the cost factor event/s/user. The adaptation factor for this cost factor is

$$5\text{ms}/20\text{ms} = 0.25,$$

Key Figure	**Cost Factor User**	**Cost Factor Events/s/User**
Base Line	250	0.5
Base Line Value	20ms	20ms
Estimation	*200*	*0.1*
Estimation Value	*18ms*	*5ms*
Adaptation Factor	$18/20 = 0.9$	$5/20 = 0.25$
Result	$0.25 * 18\text{ms} = \mathbf{4.5ms} = 0.9 * 5\text{ms}$	

Table 6.1: Service Time Estimation Using Multiple Cost Factors

which can be multiplied with the 18ms estimated by the cost factor user to get to the same estimation of

$$0.25 * 18\text{ms} = 4.5\text{ms}.$$

The approach is summarized in Table 6.1, where as a simplification the adaptation factors are only multiplied. To estimate the relationships more precisely, a full factorial design has to be used in order to determine the exact interactions. Since no full factorial design was carried out, the cost factors are assumed to be multiplicative as a simplification. Every polynomial is based on a certain configuration and has to be adapted to the given scenario.

Chapter 7

Validation of the Cost Model

In this chapter, the cost model is validated. First, a quantitative validation is performed, in which the performance predictions of the cost model are compared to measured data. Subsequently, a qualitative validation is carried out, in which the cost model is rated based on model estimate criteria, model method criteria, and application criteria. Furthermore, to illustrate the validation in a authentic environment, a concrete example (Sect. 1.6) is chosen as validation of one real use case.

7.1 Quantitative Validation

The quantitative validation is done for two different cases. *Case I* describes the predicted value using the queueing network models and service times of the components as measured. *Case II* describes the complete usage with two model layers: The queueing network models, and service times are predicted by the polynomial models. Both cases are compared to the actually measured values.

In Table 7.1 several different, randomly selected scenarios are presented with the measured response time (in ms), the predicted response time (in ms) using *Case I and II*, and the calculated MREs for both approaches. The scenarios are measured using the same rules and workload as described in Sect. 5.3 and Sect. 5.4. For example, for 50 users in synchronous mode, 11.462ms were measured. By using the measured service times for the components, 11.733ms are predicted with a MRE of 0.024. With the calculated service times, a response time of 10.624ms is estimated, compared to the measured response time the MRE is 0.073.

The validation is separated into validation of the cost model in synchronous mode at the top and asynchronous mode below. For each mode, 4 different scenarios are compared in measured and simulated values. In synchronous mode, the MMRE for *Case I* is 0.106 and for *Case II* it is 0.150. Additionally, PRED(25) of both cases is 1, since no extreme outliers are found. If more cases are simulated, the value for PRED(25) is expected to be smaller, because outliers will occur in a larger amount of scenarios.

	Scenario	Measured	Case I	MRE I	Case II	MRE II
Sync	50 Users	11.462	11.733	0.024	10.624	0.073
	100 Alpha Nodes	402.480	491.999	0.222	348.071	0.135
	100 Beta Nodes	49.561	47.354	0.045	61.015	0.231
	500 Abstractions	343.821	394.412	0.147	398.904	0.160
	MMRE			**0.106**		**0.150**
	PRED(25)			**1**		**1**
Async	200 Users	7.548	7.251	0.039	8.993	0.191
	500 Alpha Nodes	22.454	21.285	0.052	22.131	0.014
	100 Beta Nodes	7.710	7.148	0.073	8.935	0.159
	500 Abstractions	223.665	246.824	0.104	245.301	0.097
	MMRE			**0.067**		**0.115**
	PRED(25)			**1**		**1**

Table 7.1: Quantitative Validation of Cost Model Comparing Measured and Predicted Response Times

In asynchronous mode, the results are even better: MMRE of *Case I* is 0.067 and for *Case II* it is 0.115. PRED(25) is in both cases 1, since again, no scenario has outliers.

The difference in prediction quality between *Case I* and *Case II* is due to the variations in the measurements, which are smoothed in the polynomial models. As the variance is high in the measurements, the polynomial models which represent the service time function often show poor quality having R^2 near to 0. This leads to inaccurate service times in the simulations compared to the measured ones. Hence, the difference between simulated and measured response time is large in a few cases.

7.2 Qualitative Validation: Model Criteria

In this section, the cost model will be discussed using the model criteria from Sect. 2.1.3. This validation emphasizes room for improvement for further versions of the cost model as well as in the automation of the approach, which is further discussed in the outlook (Chap. 9).

For the *model estimate criteria* (Table 7.2), all criteria are met at a high intensity. However, also the inputs required to create the estimations are high, since for each cost factor a forecasted value is required, even all rules have to be known in detail to be able to predict the number of alpha and beta nodes.

The *model method criteria* (Table 7.3) show, on the one hand, that the repeatability of the method is high, i.e. the test can be repeated easily, and the transparency is also high. On the other hand, several assumptions need to be made (for example,

Model and Estimate Criteria	**Intensity**	**Description**
Quality of model and estimate	high	MMRE of four simulations are 0.111 (synchronous mode) and 0.167 (asynchronous mode), PRED(25) is 1 in both cases
Inputs required	high	values for all cost factors needed
Completeness	high	response time and throughput are delivered
Type of estimates	high	both average measure plus standard error are provided
Calibration	high	calibration needed to differentiate synchronous and asynchronous mode, no further calibration needed
Interpretability	high	results are response times and throughput, which can be interpreted easily even if one is unfamiliar with the cost model

Table 7.2: Model Estimate Criteria

Estimation Method Criteria	**Intensity**	**Description**
Assumptions	high	several restrictions and simplifications had to be used
Repeatability	high	test results are repeatable, queueing networks are defined by the architecture of the platform
Complexity	high	estimation method consists of several complex steps
Automation (Modeling)	low	each cost factor had to be tested individually, and polynomial models had to be applied thereafter, support by tools (java, R, JMT) are used wherever possible
Transparency	high	applied statistics are described and the whole process from data to cost model is transparently described

Table 7.3: Model Method Criteria

distribution of the service times based on few results), the complexity of the modeling method is high and also the automation of modeling is low, even though programming and scripts (e.g., Appendix C) are used, but they need to be adapted slightly for modeling every cost factor.

With the *application criteria* (Table 7.4), the application of the cost model is evaluated. The application coverage is medium, since many cost factors have been ex-

Application Criteria	**Intensity**	**Description**
Application Coverage	medium	a few cost factors were excluded, full factorial design has to be implemented in order to detect influences between cost factors
Generalizability	low	cost model is not reusable in other contexts as it is dependent on the architecture of the platform
Comprehensiveness	high	response time in total as well as service time of components can be estimated, bottlenecks can be found
Availability of estimates	medium	estimation can be done as early as forecasts for the cost factors levels are available
Automation (Method Usage)	low	manual calculation of service times before inserting these values into queueing network model to generate the response time estimation is necessary

Table 7.4: Application Criteria

cluded, and the applied test design lacks information on relationships between cost factors. Further measurements are needed in order to improve this point. Generalizability and automation of the method usage are low, the latter one could be improved by developing a tool which automatically calculates service times based on the cost factors and inserts these values into the queueing network model. Comprehensiveness is high since the response time can be easily interpreted, the availability of estimates is medium since the estimation can be done as soon as cost factor levels are available.

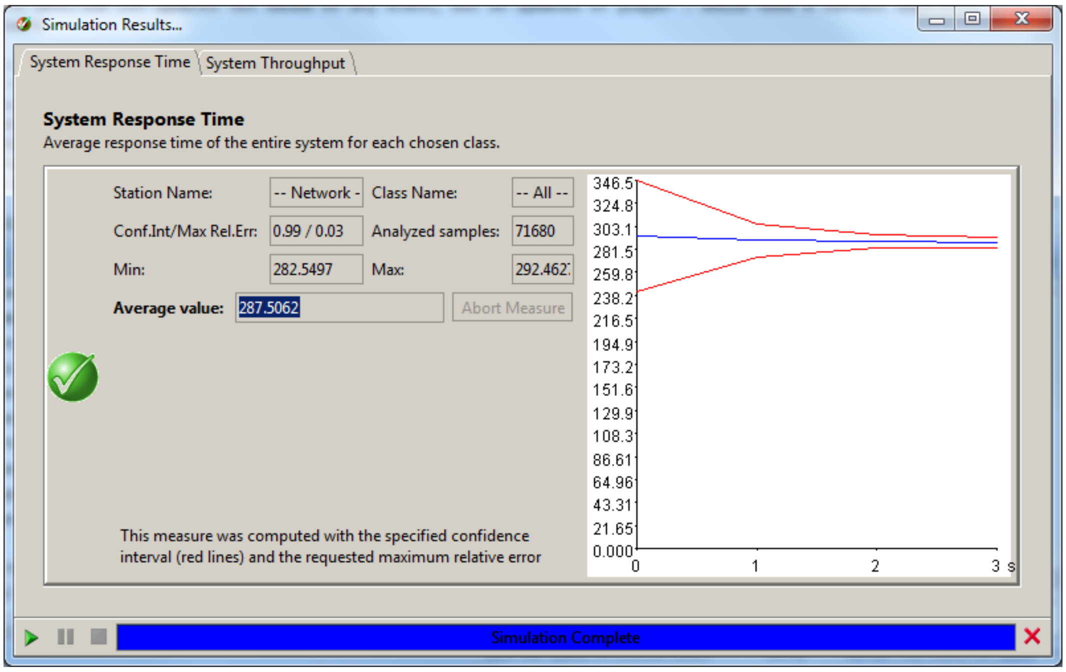
7.3 Validation Based On a Use Case

The application and its purpose used for this validation have been described in Sect. 1.6. For the implementation of the gamification concept, 43 rules (List. B.3-B.13) as defined in Appendix B are deployed. 11 rules are used to reward or count points based on events, for example, giving a buddy point for each added buddy. Additionally, 30 rules (List. B.14-B.43) check whether the user completed missions, and, if so, reward badges and assign new missions. A *new user* rule (List. B.1) is used to create the user in the BEP and assign him the first missions. The *met person* rule (List. B.2) uses a join node to count how many people from different departments one has met. To ensure that people are not rewarded twice if they meet again, an abstract event is created.

The real use case is simulated and measured for 300 seconds with 200 users in both synchronous and asynchronous mode. In synchronous mode, the interarrival time of all event types is approx. 500ms, which is an event rate of 0.01 events per second per user. The interarrival time in asynchronous mode is simulated being approx. 50ms,

Scenario	Measured (ms)	Estimation (ms)	MRE
200 User Sync Mode	200.259	287.506	0.436
200 User Async Mode	11.561	13.272	0.148

Table 7.5: Validation of Cost Model Comparing Measured and Predicted Response Times for Use Case


Simulation Results...

System Response Time | System Throughput

System Response Time

Average response time of the entire system for each chosen class.

Station Name: -- Network -- Class Name: -- All --

Conf.Int/Max Rel.Err: 0.99 / 0.03 Analyzed samples: 71680

Min: 282.5497 Max: 292.462

Average value: 287.5062

This measure was computed with the specified confidence interval (red lines) and the requested maximum relative error

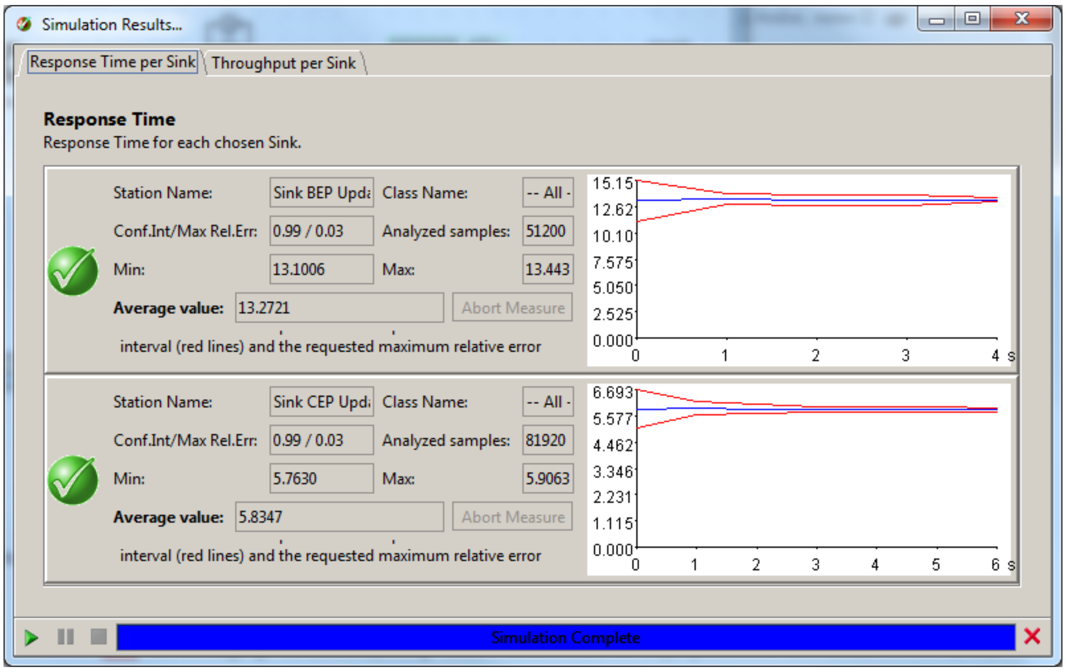
Simulation Complete

Figure 7.1: Synchronous Simulated Response Time for Use Case

that is an event rate of 0.1 events per second per user. The RETE tree for the deployed rules is shown in Fig. B.1. This tree has been extracted from the RETE tree view in Drools. In total, 14 alpha nodes, eight beta nodes and one abstraction are used within the rules. A warmup phase with four users is simulated before taking measurements. Furthermore, the first ten seconds have been excluded from the measurement results.

Results for asynchronous and synchronous mode are summarized in Table 7.5. The measured response time in synchronous mode for 200 users is 200.259ms. The simulation provides a result of 287.506ms (Fig. 7.1), so that the MRE is 0.436. For the asynchronous mode, which is simulating 200 users, too, the measured response time is 11.561ms, whereas the simulation estimates a response time of 13.272ms (Fig. 7.2). The MRE in that case is 0.148.

Deviations between measured and estimated response time are not negligible, but explicable. For example, the rule *metPerson* (List. B.13) includes a beta nodes joining specific events. This join node lets the working memory grow, as events are not


Simulation Results...

Response Time per Sink | Throughput per Sink

Response Time
Response Time for each chosen Sink.

Station Name: Sink BEP Upd:
Class Name: -- All -
Conf.Int/Max Rel.Err: 0.99 / 0.03
Analyzed samples: 51200
Min: 13.1006
Max: 13.443
Average value: 13.2721
Abort Measure
interval (red lines) and the requested maximum relative error

Station Name: Sink CEP Upd:
Class Name: -- All -
Conf.Int/Max Rel.Err: 0.99 / 0.03
Analyzed samples: 81920
Min: 5.7630
Max: 5.9063
Average value: 5.8347
Abort Measure
interval (red lines) and the requested maximum relative error

Simulation Complete

Figure 7.2: Asynchronous Simulated Response Time for Use Case

retracted and more and more events need to be joined in the node. Additionally, the probabilities of rules firing their actions do change over time. Most of the rules are provided to be fired only once for every user (to complete the mission and reward the user with a badge), so that the probabilities change over time. As already mentioned, such a behavior cannot be modeled accurately using stateless queueing networks. Furthermore, the factors in this use case are assumed to be multiplicative since the relationships between factors have not been measured. This very simplifying assumption is expected to also lead to inaccuracies.

7.4 Conclusion

The quantitative validation of the cost model shows that the response time can be predicted accurately. The MMRE is 0.167 and PRED(25) is 1 in synchronous mode, and in asynchronous mode an MMRE of 0.111 and PRED(25) of 1 are calculated for the response time prediction. The error is based on both variances in the polynomial models and queueing network models. Additionally, the validation based on the more complex use case provides an acceptable result for synchronous mode ($MRE = 0.436$) and a good result in asynchronous mode ($MRE = 0.148$).

The qualitative validation shows need for improvement in several areas, for example, automating the method usage, using less assumptions and restrictions, and including more cost factors to improve the accuracy of the estimations. In the outlook it is further discussed how these points can be addressed. The next chapter summarizes the main results of the thesis and gives an overview on the researched cost factors.

Chapter 8

Summary

This summary concludes the main results of the thesis. A two-layered cost model, consisting of a queueing network model and several polynomial models, is provided and was validated to show that the estimation accuracy and model quality is acceptable. Response time and throughput can be estimated for different scenarios by using the presented cost model. The accuracy of the cost model was calculated by comparing simulated and measured response time, and it shows an MMRE of 0.150 in synchronous mode, and an MMRE of 0.115 in asynchronous mode. For the use case involving several more complex rules, an MRE of 0.436 in synchronous mode and an MRE of 0.148 have been determined.

Several cost factors were discussed and researched within the thesis, which are listed in Table 8.1. For every cost factor, the relevant subsections of the thesis are linked, in which the cost factor is discussed and its exact influence is determined. A sign shows whether the cost factor is relevant, and, hence, included in the cost model.

In the user-related category three factors have been found to be relevant and are included in the cost model: Number of users, events/s/user and the event type distribution. For the category concerning rule-related cost factors, several cost factors have been found to significantly influence the response time, mostly because they affect the CEP behavior. In this category, the number of alpha and beta nodes, the number of abstractions, updateAPI calls, and queryAPI calls have been included in the cost model. Node types and growth of the working memory have been found to significantly influence the response time, but could not be modeled due to time limitations. Additionally, the influence of independent rule streams and table structure on the response time could not be taken into account. Several infrastructure-related cost factors were found not to influence the response time. Indeed, for a insufficiently dimensioned server RAM the rule engine does not perform correctly any longer, e.g., not rewarding the expected amount of points to the user. As such an incorrect behavior has to be avoided in any case, the infrastructure has to be adequately dimensioned at any time. A few factors, such as database (configuration) and queue scheduling algorithm are expected to influence the response time, but were also excluded in the cost model.

	Cost Factor	**Relevant**	**Modeled**
Users	Number of Users (4.1.1/5.5.2)	X	X
	Events/s/User (4.1.2/5.5.3)	X	X
	Event Type Distribution (4.1.3/5.5.4)	X	X
Rules	Number of Rules (4.2.1)	–	–
	Number of Alpha Nodes (4.2.2/5.5.5)	X	X
	Number of Beta Nodes (4.2.3/5.5.6)	X	X
	Node Types (4.2.4)	X	–
	Number of Abstractions (4.2.5/5.5.7)	X	X
	Working Memory Growth Rate (4.2.6)	X	–
	Independent Rule Streams (4.2.7)	?	–
	Number of UpdateAPI calls (4.2.8/5.5.8)	X	X
	Number of QueryAPI calls (4.2.9/5.5.9)	X	X
	Size of Tables (4.2.10)	–	–
	Structure of Tables (4.2.11)	?	–
Infrastructure	Database (4.3.1)	X	–
	Queue Scheduling Algorithm (4.3.2)	X	–
	Connection Pools: max. Number of Threads (4.3.3)	–	–
	Connection Pools: Accept Count (4.3.3)	–	–
	Transmission Packet Sizes (4.3.4)	?	–
	RAM of Server (4.3.5)	–	–

X = yes; – = no; ? = unknown

Table 8.1: Summary of Studied Cost Factors

Regarding the research questions, the most relevant cost factors have been identified and measured in their influence on the performance of the Gamification Platform. The cost model presented in the thesis and its usage were described and exemplarily applied for a use case.

Chapter 9

Outlook

This chapter discusses further research areas, which have been identified throughout the thesis. It also discusses problems and restrictions of the thesis and how these can be treated in future. The aspects which offer room for improvement are the following.

Automation of Model Usage Estimating the response time with the model is a sophisticated process consisting of many steps. Facilitating the usage of the cost model is of high priority. Automatic analysis of rules or whole gamification concepts, but also the integration of the cost model into the Gamification Platform are possible. With an automated cost model usage, decisions between alternatives and comparisons of different gamification concepts would also be simplified.

Investigate Relationships Between Factors In the presented cost model, all factors are assumed to be multiplicative. The measured error for the use case already demonstrates that this assumption is in need of improvement. Cost factors are influencing each other, and the relationships need to be researched, e.g., by using a full factorial design. With this test design, independent factors can be identified, too. This knowledge can be utilized to optimize gamification concepts to exploit such independent relations.

Calibrate and Improve the Cost Model The cost model is very flexible and allows easy calibration and adaptations. As soon as more cost factors are investigated, they need to be added to the cost model. Cost factors which have been excluded, such as network and transmission times, or working memory growth, definitely need to be included to assess whether a feedback can be given in less than 400ms to the user in any case. The current cost model is using queueing networks as its modeling technique, but in order to include other factors, a different or extended modeling technique, for example, queueing petri nets need to be considered as well. Besides, data from new applications using the Gamification Platform should be used to calibrate the model and to even build a knowledge base which would enable to calibrate the cost model automatically.

Earlier Estimation of Performance The presented cost model can only give an accurate estimation if precise information is already available. A cost and performance estimation might be useful for funding a gamification concept even though the details needed for the cost model are not defined yet. Hence, a cost model providing estimations in an earlier phase is desirable, e.g. similar to COCOMO, which provides three models, each for a different project phase. Bottlenecks and problems would already become obvious at early stages and could, thus, be solved easier.

Investigate a Broader Range of Workloads In this thesis, only workload causing a steady state has been investigated. Since workloads in real applications are seldom as smooth as the simulated and measured ones, other workloads should be researched. Spike tests with heavy peak loads or stress tests under extreme conditions need to be conducted just like the investigation of the ramp-up phase. Service times and response times need to be observed under different circumstances to be predictable for any workload. Additionally, workload data generated by real applications should also be used to simulate real scenarios.

Allow Optimization of the Gamification Platform Based on Cost Model Estimations The Gamification Platform is configurable in several ways. For example, the scheduling of queues can be optimized for a given scenario as discussed in Kleinrock [1976]. The optimal configuration for queue scheduling could be ascertained by simulating different scheduling algorithms in the cost model. Furthermore, database parameters, RAM size, cache size, and several other configuration parameters could be optimized by including them into the cost model and determining the best possible configuration with the cost model.

Bibliography

Adiri, I. (1969). Computer Time-Sharing Queues with Priorities. *J. ACM*, 16(4):631–645.

Akaike, H. (1974). A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716 – 723.

Albert, L. (1989). Average Case Complexity Analysis of RETE Pattern-Match Algorithm and Average Size of Join in Database. In *Proceedings of the Ninth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 223–241, London, UK, UK. Springer-Verlag.

Albert, L. and Fages, F. (1988). Average Case Complexity Analysis of the Rete Multi-Pattern Match Algorithm. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, ICALP '88, pages 18–37, London, UK, UK. Springer-Verlag.

Albert, L. and Régnier, M. (1991). Complexity of recursive production rules execution. In *Proceedings of the 3rd symposium on Mathematical fundamentals of database and knowledge base systems*, MFDBS 91, pages 188–200, New York, NY, USA. Springer-Verlag New York, Inc.

Alfons, A. (2013). R Documentation - Cross-validation tools for regression models. Available online at http://cran.r-project.org/web/packages/cvTools/cvTools.pdf (visited on 12.03.2013).

Apache Software Foundation (2013). ActiveMQ. Available online at http://activemq.apache.org/how-does-a-queue-compare-to-a-topic.html (visited on 14.02.2013).

Barachini, F. (1994). Frontiers in run-time prediction for the production-system paradigm. *AI Mag.*, 15(3):47–61.

Barachini, F., Mistelberger, H., and Gupta, A. (1992). Run-time prediction for production systems. In *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, pages 478–485. AAAI Press.

Bertoli, M., Casale, G., and Serazzi, G. (2009). JMT - Performance Engineering Tools for System Modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15.

Black, K. (2009). *Business Statistics: Contemporary Decision Making.* Wiley Plus Products Series. John Wiley & Sons.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., and Selby, R. (1995). Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. In *ANNALS OF SOFTWARE ENGINEERING*, pages 57–94.

Boehm, B. W. (1984). Software Engineering Economics. *Software Engineering, IEEE Transactions on*, SE-10(1):4 –21.

Box, G. E. and Hunter, J. S. (2000). The 2k-p fractional factorial designs part I. *Technometrics*, 42(1):28–47.

Boyer, B. (2008a). Robust Java benchmarking, Part 1: Issues Understand the pitfalls of benchmarking Java code. Available online at http://www.ibm.com/developerworks/java/library/j-benchmark1/index.html (visited on 07.03.2013).

Boyer, B. (2008b). Robust Java benchmarking, Part 2: Statistics and solutions: Introducing a ready-to-run software benchmarking framework. Available online at http://www.ibm.com/developerworks/library/j-benchmark2/index.html (visited on 07.03.2013).

Briand, L. C., El Emam, K., Surmann, D., Wieczorek, I., and Maxwell, K. D. (1999). An assessment and comparison of common software cost estimation modeling techniques. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 313 –323.

Briand, L. C. and Wieczorek, I. (2002). *Resource Estimation in Software Engineering.* John Wiley and Sons, Inc.

Chrysler, E. (1978). Some basic determinants of computer programming productivity. *Commun. ACM*, 21(6):472–483.

Cohen, J. (1968). Multiple regression as a general data-analytic system. *Psychological Bulletin*, 70:426–443.

Davis, R., Buchanan, B., and Shortliffe, E. (1977). Production rules as a representation for a knowledge-based consultation program. *Artificial Intelligence*, 8(1):15–45.

Deterding, S., Sicart, M., Nacke, L., O'Hara, K., and Dixon, D. (2011). Gamification. using game-design elements in non-gaming contexts. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 2425–2428, New York, NY, USA. ACM.

Doorenbos, R. B. (1995). *Production matching for large learning systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Faraway, J. J. (2002). *Practical Regression and Anova using R.* CRC Press. Available online at http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf (visited on 14.02.2013).

Feitelson, D. (2002). Workload Modeling for Performance Evaluation. In Calzarossa, M. and Tucci, S., editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pages 114–141. Springer Berlin Heidelberg.

Forgy, C. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligences*, 19(1):17–37.

Forgy, C. L. (1979). *On the efficient implementation of production systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Gray, J., editor (1993). *The Benchmark Handbook for Database and Transaction Systems (2nd Edition).* Morgan Kaufmann.

Gupta, A. (1984). Parallelism in Production Systems: The Sources and Expected Speed-up. Technical report, Fifth International Workshop Agence de l'Informatique.

Gupta, A., Forgy, C., Newell, A., and Wedig, R. (1986). Parallel algorithms and architectures for rule-based systems. In *Proceedings of the 13th annual international symposium on Computer architecture*, ISCA '86, pages 28–37, Los Alamitos, CA, USA. IEEE Computer Society Press.

Hayashi, F. (2000). *Econometrics.* Princeton Univ. Press, Princeton, NJ [u.a.].

Herzig, P., Ameling, M., and Schill, A. (2012a). A Generic Platform for Enterprise Gamification. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 219 –223.

Herzig, P., Strahringer, S., and Ameling, M. (2012b). *Gamification of ERP Systems – Exploring Gamification Effects on User Acceptance Constructs*, pages 793–804. GITO.

Jain, R. (1991). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley.

Jorgensen, M. and Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. *Software Engineering, IEEE Transactions on*, 33(1):33 –53.

Kendall, D. G. (1953). Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24(3):pp. 338–354.

Kenneth P. Burnham, D. R. A. (2004). Multimodel Inference: Understanding AIC and BIC in Model Selection. *Sociological Methods & Research*, 33(2):261–304.

Kirk, R. (2007). *Statistics: An Introduction.* International student edition. Thomson/Wadsworth.

Kleinbaum, D. (2007). *Applied Regression Analysis and Other Multivariable Methods.* Duxbury applied series. Brooks/Cole.

Kleinrock, L. (1975). *Queueing Systems: Theory.* Queueing Systems. Wiley.

Kleinrock, L. (1976). *Queueing Systems: Computer applications.* Wiley-Interscience Publication. Wiley.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann.

Kurose, J. F. and Ross, K. (2013). *Computer Networking: A Top-down Approach.* Pearson Education, Limited, Boston, MA, USA, 6th edition.

M2 Research (2012). Gamification in 2012. Available online at http://gamingbusinessreview.com/wp-content/uploads/2012/05/Gamification-in-2012-M2R3.pdf (visited on 05.02.2013).

Menasce, D. and Gomaa, H. (2000). A method for design and performance modeling of client/server systems. *Software Engineering, IEEE Transactions on*, 26(11):1066–1085.

Menasce, D. A. and Almeida, V. (2001). *Capacity Planning for Web Services: metrics, models, and methods.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Menzies, T., Chen, Z., Hihn, J., and Lum, K. (2006). Selecting Best Practices for Effort Estimation. *Software Engineering, IEEE Transactions on*, 32(11):883 –895.

Paton, N. W. and Díaz, O. (1999). Active database systems. *ACM Comput. Surv.*, 31(1):63–103.

Port, D. and Korte, M. (2008). Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 51–60, New York, NY, USA. ACM.

Rawlings, J., Pantula, S., and Dickey, D. (1998). *Applied Regression Analysis: A Research Tool.* Springer Texts in Statistics. Springer.

SAP AG (2013). MaxDB - Database Parameter. Available online at http://maxdb.sap.com/doc/7_8/44/bd1ec6a5d51388e10000000a155369/content.htm (visited on 12.04.2013).

Skousen, R., Lonsdale, D., and Parkinson, D. (2002). *Analogical Modeling: An Exemplar-Based Approach to Language.* Human cognitive processing. J. Benjamins Pub.

Solomatine, D. P., See, L. M., and Abrahart, R. J. (2008). Data-Driven Modelling: Concepts, Approaches and Experiences. In Abrahart, R. J., See, L. M., and Solomatine, D. P., editors, *Practical Hydroinformatics*, volume 68 of *Water Science and Technology Library*, pages 17–30. Springer Berlin Heidelberg.

Stuckenschmidt, H. and Broekstra, J. (2005). Time – Space Trade-offs in Scaling up RDF Schema Reasoning. In *Proceedings of the 2005 international conference on Web Information Systems Engineering*, WISE'05, pages 172–181, Berlin, Heidelberg. Springer-Verlag.

Sun Microsystems, Inc (2012). Java Message Service Specification Version 1.1. Available online at http://download.oracle.com/otndocs/jcp/7195-jms-1.1-fr-spec-oth-JSpec/ (visited on 13.02.2013).

The Apache Software Foundation (2013). Apache Tomcat Configuration Reference. Available online at http://tomcat.apache.org/tomcat-6.0-doc/config/http.html (visited on 29.04.2013).

The JBoss Drools Team (2013). Drools Expert User Guide, Version 5.5.0.Final. Available online at http://docs.jboss.org/drools/release/5.5.0.Final/drools-expert-docs/pdf/drools-expert-docs.pdf (visited on 29.01.2013).

The R Foundation for Statistical Computing (2013a). Introduction to R. Available online at http://www.r-project.org/about.html (visited on 27.02.2013).

The R Foundation for Statistical Computing (2013b). R Documentation - Fitting Linear Models. Available online at http://stat.ethz.ch/R-manual/R-patched/library/stats/html/lm.html (visited on 27.02.2013).

The R Foundation for Statistical Computing (2013c). R Documentation - Summarizing Linear Model Fits. Available online at http://stat.ethz.ch/R-manual/R-patched/library/stats/html/summary.lm.html (visited on 08.05.2013).

Thom, J., Millen, D., and DiMicco, J. (2012). Removing gamification from an enterprise SNS. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1067–1070, New York, NY, USA. ACM.

Wagenmakers, E.-J. and Farrell, S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin & Review*, 11:192–196.

Whitley, E. and Ball, J. (2002). Statistics review 6: Nonparametric methods. *Critical Care*, 6:1–5.

Yu, T. and Jajodia, S. (2007). *Secure Data Management in Decentralized Systems*. Advances in Information Security. Springer Science+Business Media, LLC.

Zichermann, G. and Cunningham, C. (2011). *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. O'Reilly Media.

Appendix A

Test Results – Polynomial Model Charts

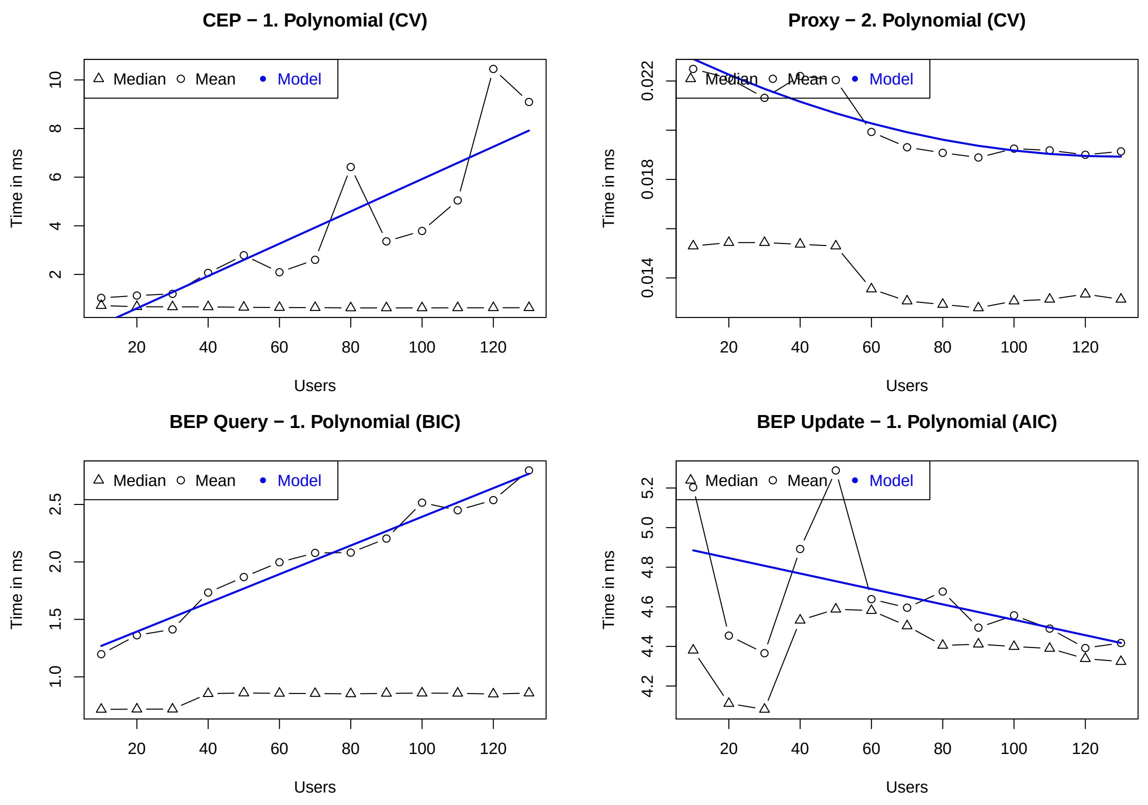


Figure A.1: User Sync Component Models

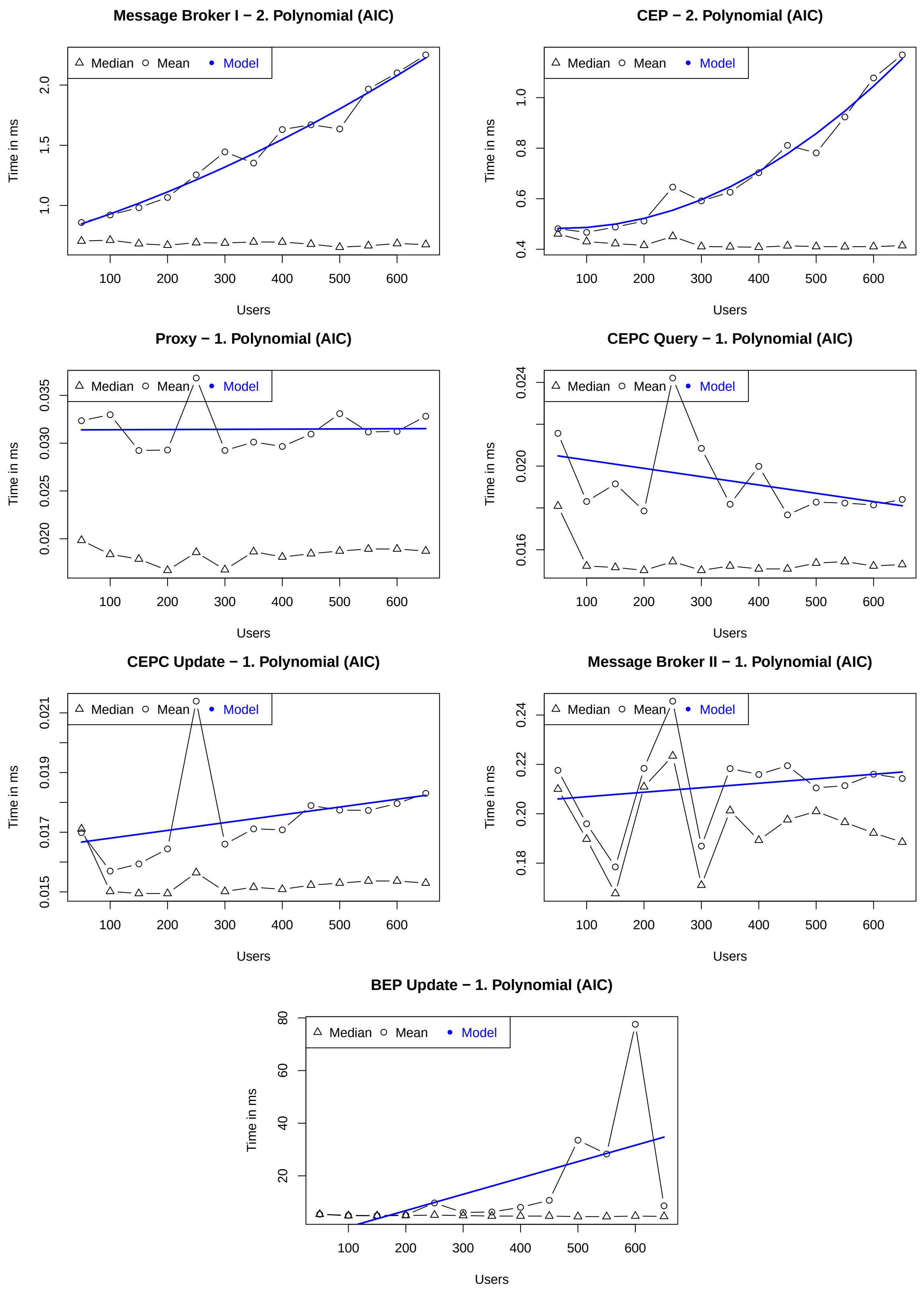


Figure A.2: User Async Component Models

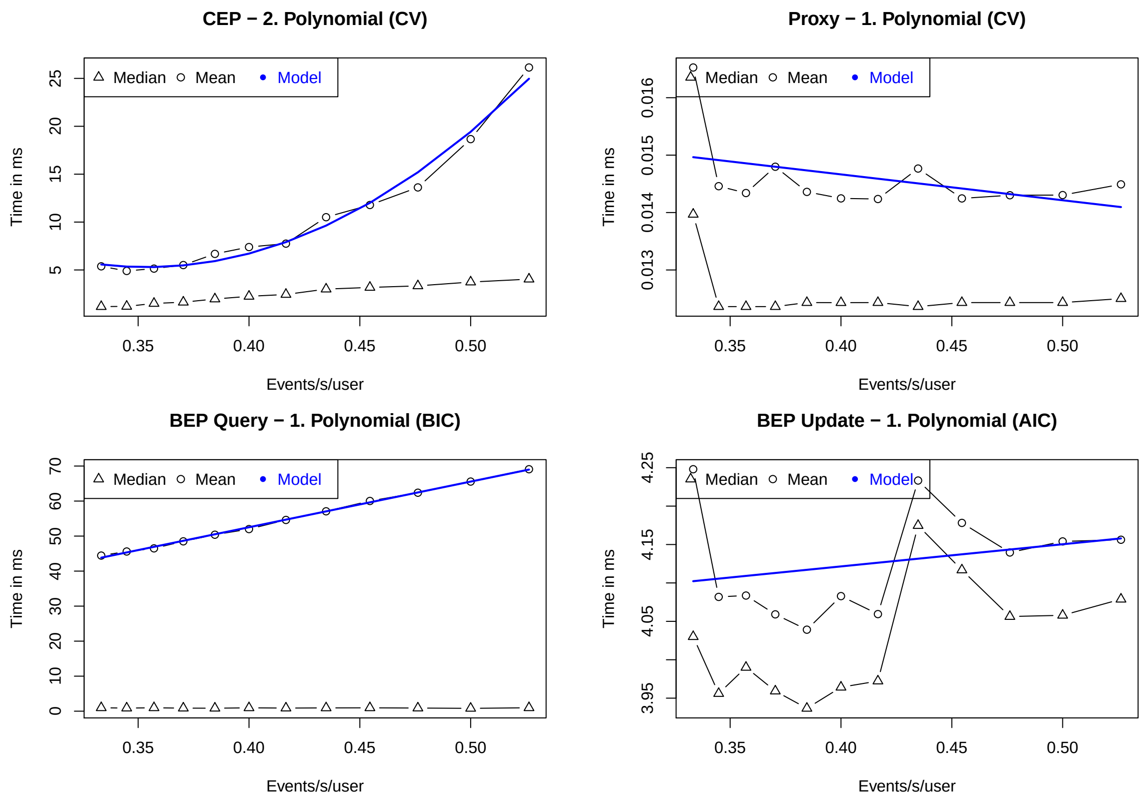


Figure A.3: Events Per Second Per User Sync Component Models

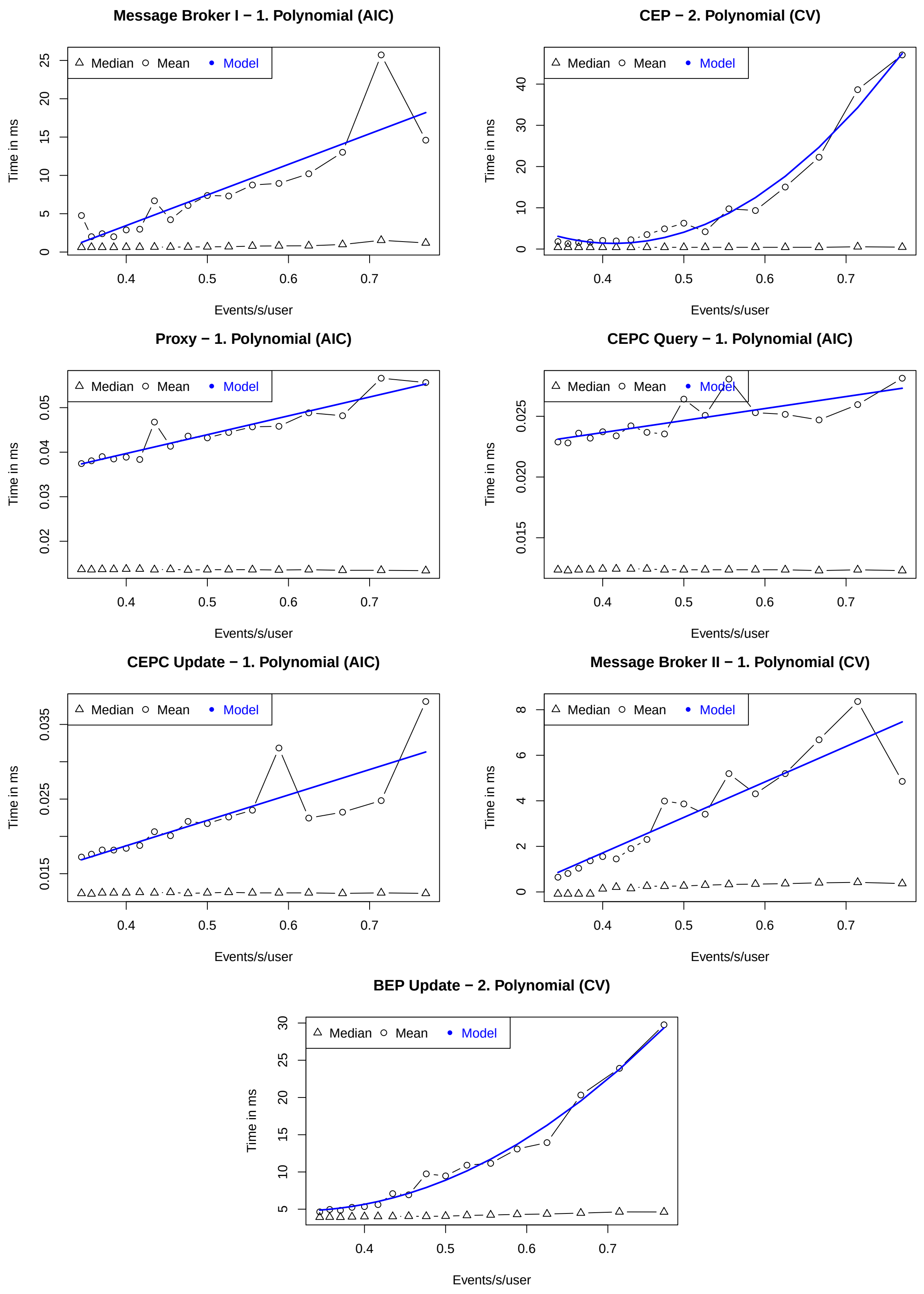


Figure A.4: Events Per Second Async Component Models

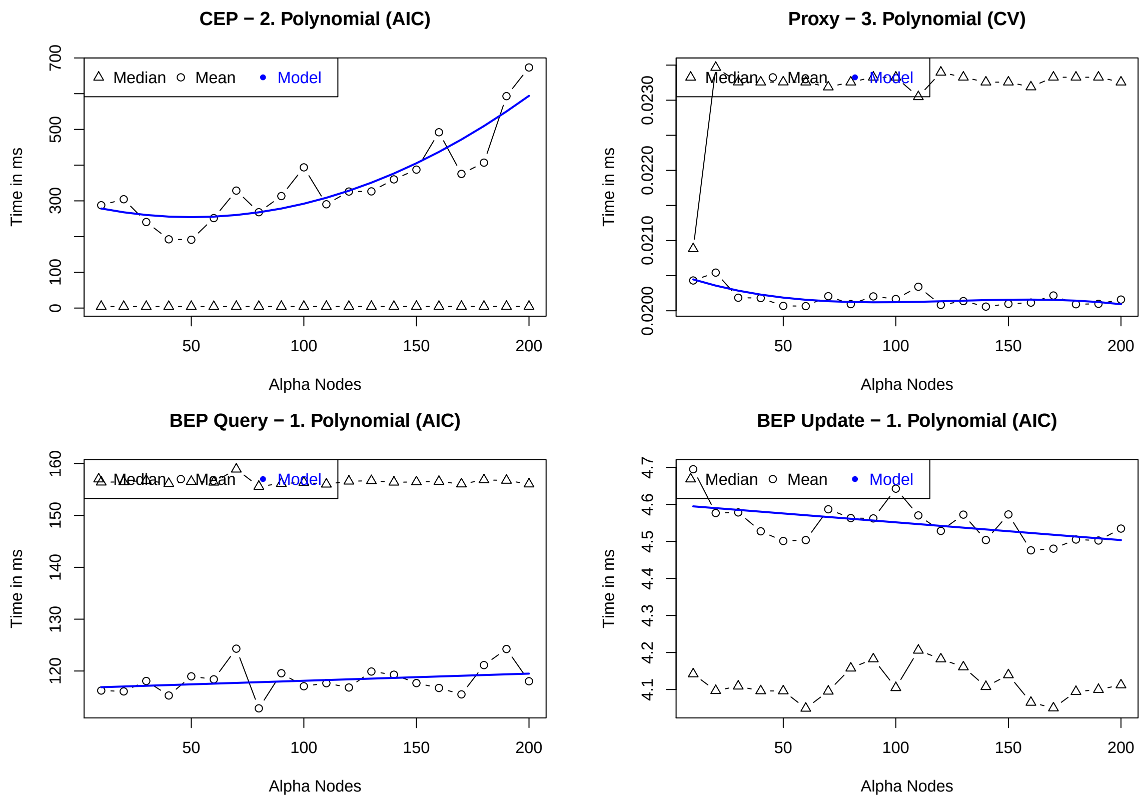


Figure A.5: Alpha Nodes Sync Component Models

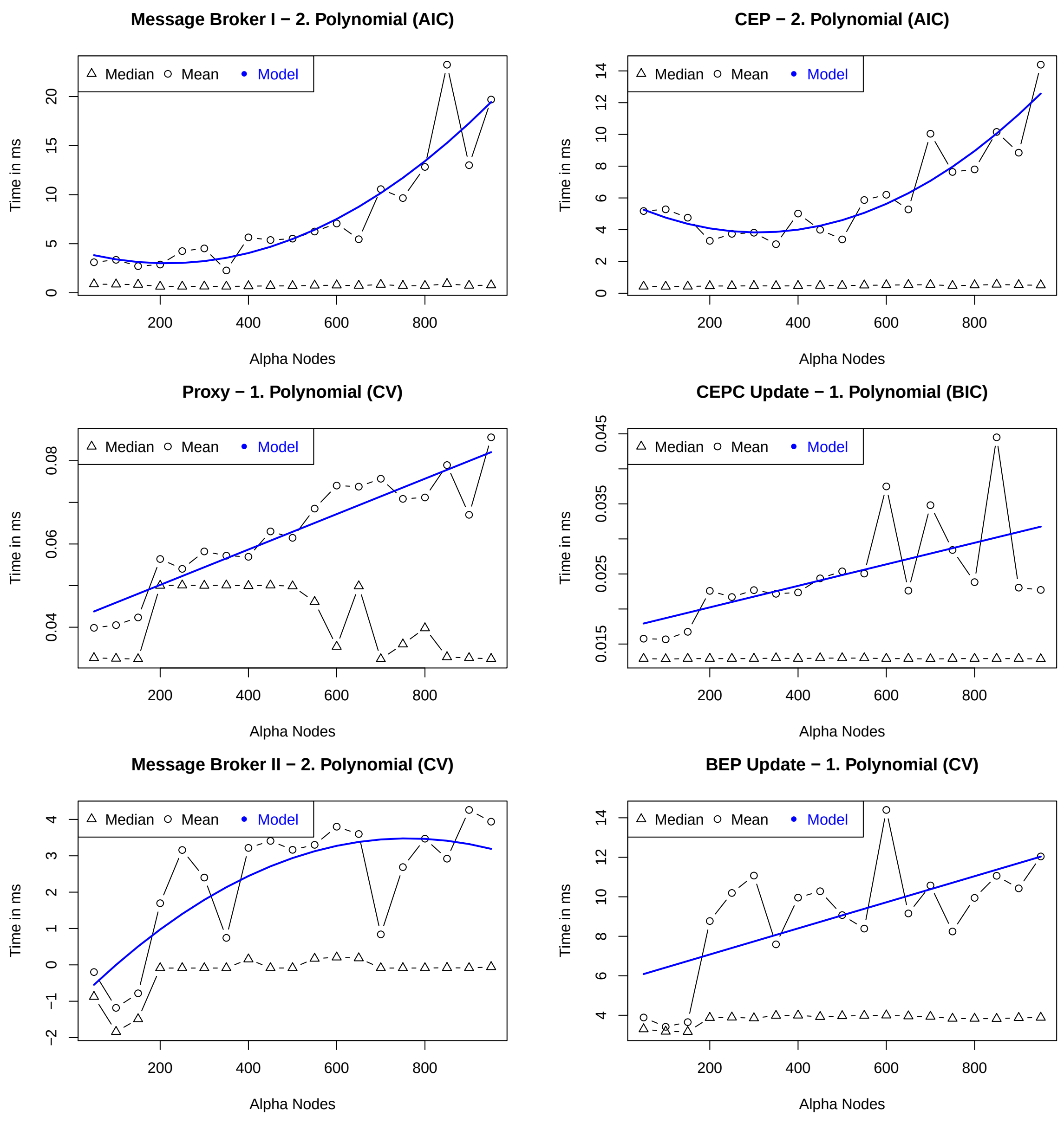


Figure A.6: Alpha Nodes Async Component Models

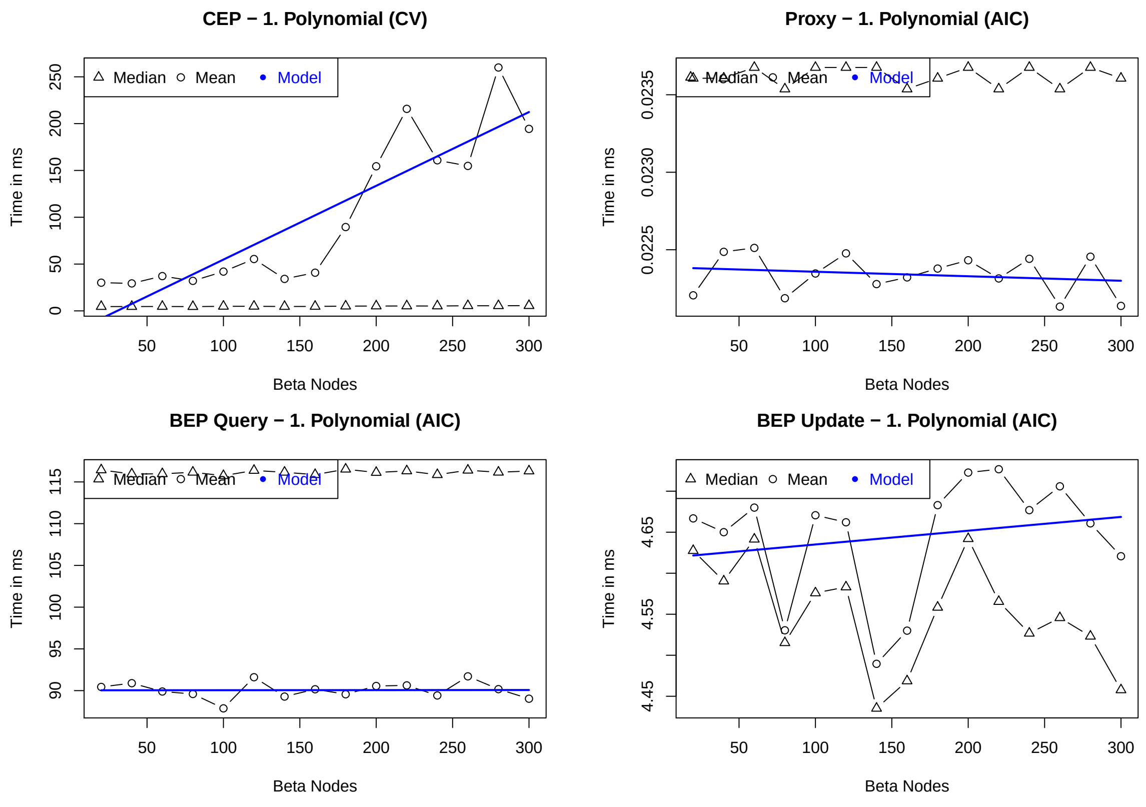


Figure A.7: Beta Nodes Sync Component Models

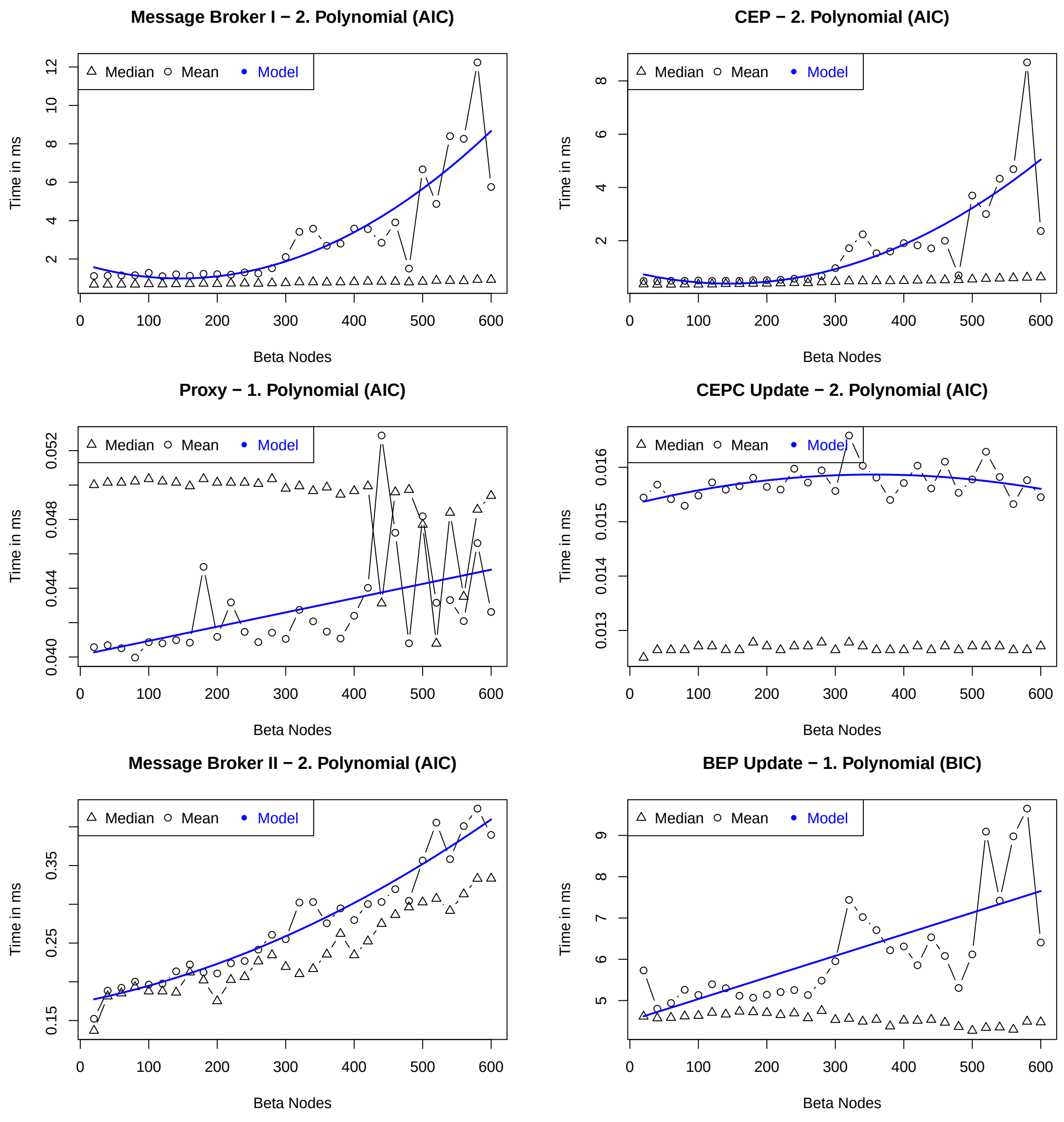


Figure A.8: Beta Nodes Async Component Models

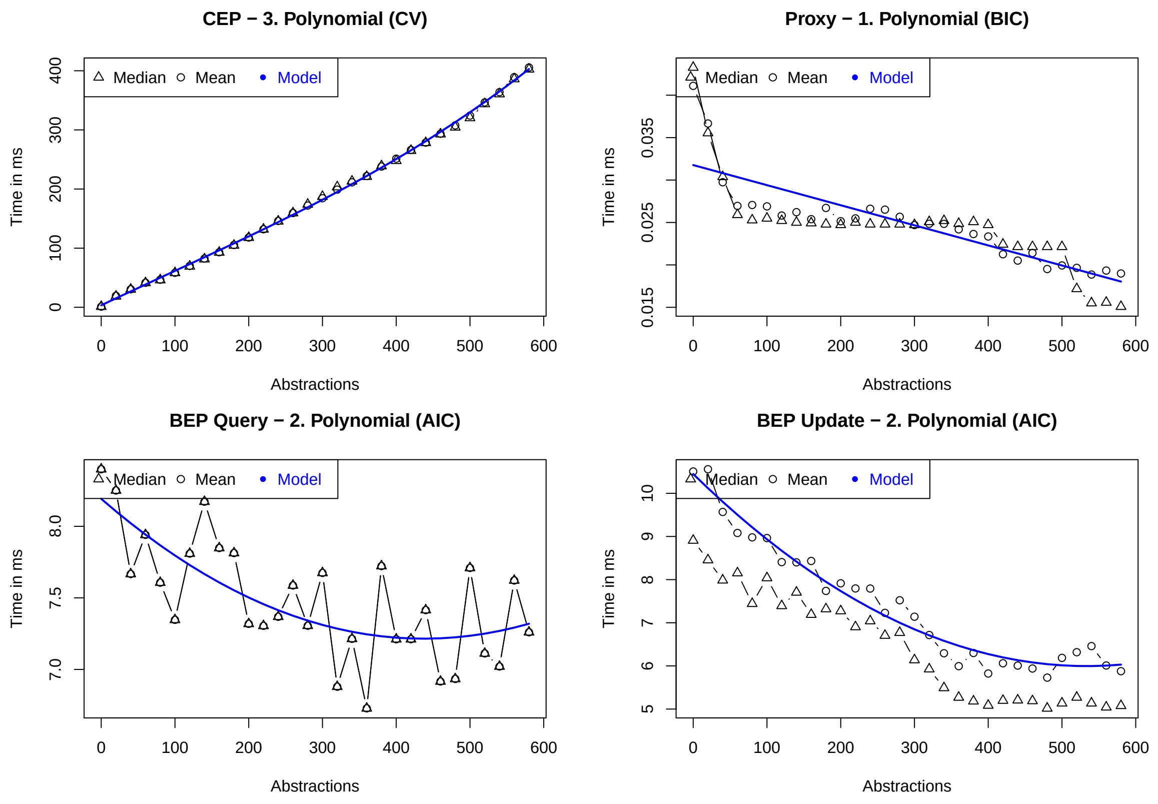


Figure A.9: Abstractions Sync Component Models

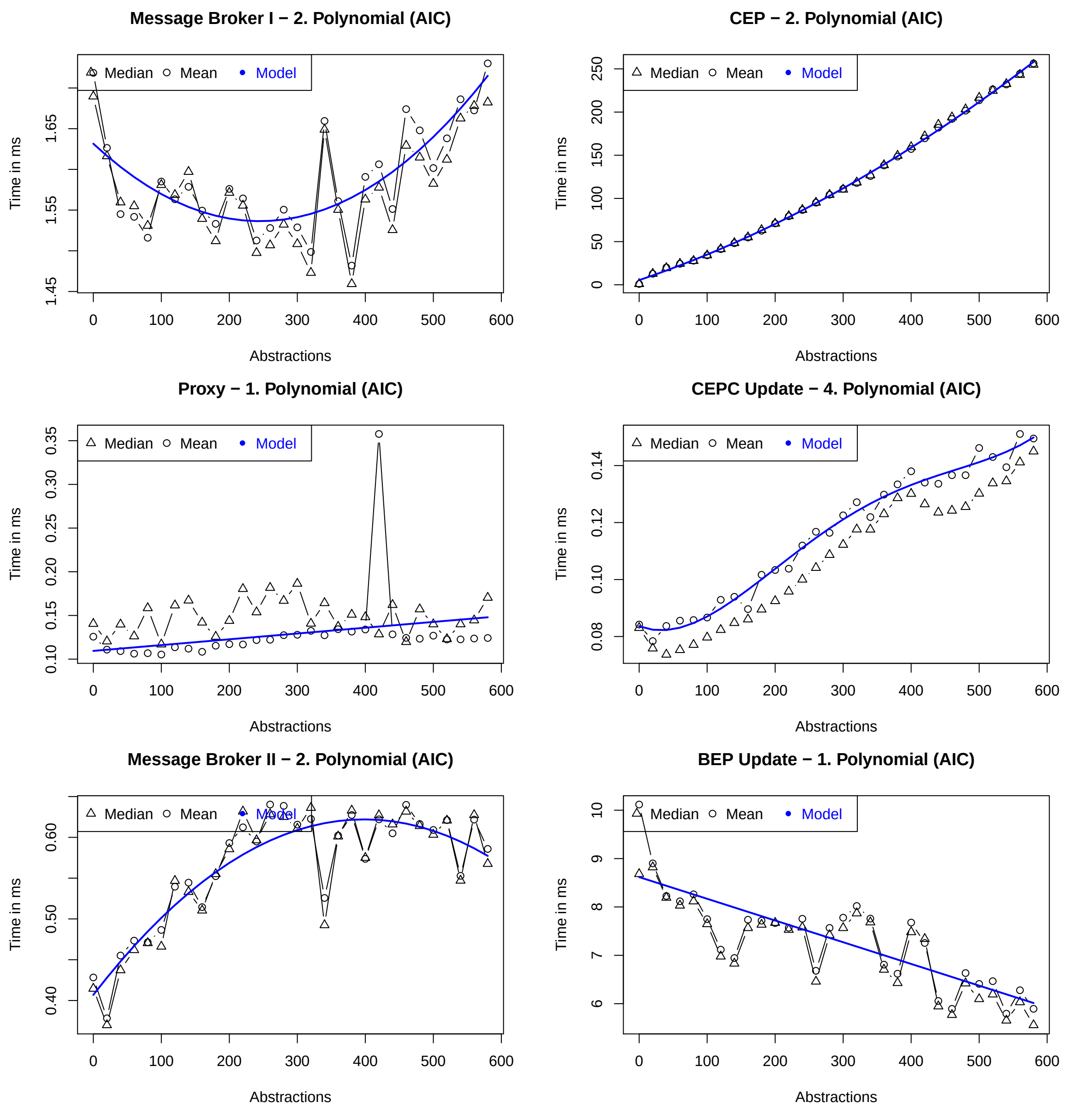


Figure A.10: Abstractions Async Component Models

Appendix B

Use Case Rules

```
rule "newUser"
  when
    $evt : EventObject(type=='newUser', $playerid:playerid) from entry-point eventstream
  then
    adminApi.createPlayer($playerid, "SAP", true);
    updateAPI.addMissionToPlayer($playerid, 'Get Ready to Network!');
    updateAPI.addMissionToPlayer($playerid, 'On My Calendar');
    updateAPI.addMissionToPlayer($playerid, "I've Got Buds!");
    retract($evt);
    update(queryAPI.getPlayer($playerid)); //only in synchronous mode
end
```

Listing B.1: New User Rule

```
rule "metPerson"
  when
    $attendedMeeting1 : EventObject(type=='attendedMeeting', $meetingid:data['meetingid'], $costCenter1:data['costCenter'], $playerid:playerid) from entry-point eventstream
    $attendedMeeting2 : EventObject(type=='attendedMeeting', data['meetingid']==$meetingid, $costCenter1!=data['costCenter'], $costCenter2:data['costCenter'], playerid!=$playerid, $playerid2:playerid) from entry-point eventstream
    not(EventObject(type=='metPerson', playerid==$playerid, data['person']==$playerid2) from entry-point internalstream)
  then
    EventObject evt = new EventObject();
    evt.setType('metPerson');
    evt.setPlayerid($attendedMeeting1.getPlayerid());
    evt.put('costCenter', $costCenter1);
    evt.put('person', $attendedMeeting2.getPlayerid());
    evt.put('personCostCenter', $costCenter2);
    entryPoints['internalstream'].insert(evt);
end
```

Listing B.2: Met Person Rule

```
rule "addedBuddy"
  when
    $addBuddy : EventObject(type=='addBuddy', $playerid:playerid) from entry-
    point eventstream
  then
    updateAPI.givePoints($playerid, 'Buddies', 1, 'Added a new Buddy');
    retract($addBuddy);
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.3: Added Buddy Rule

```
rule "addedTag"
  when
    $addTag : EventObject(type=='addTag', $playerid:playerid, $prominent:data['
    prominentTag']) from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Tags', 1, 'Added a new Tag');
    if ($prominent.equals('true')) {
      updateAPI.givePoints($playerid, 'Prominent Tags', 1, 'Added a new
    prominent Tag');
    }
    retract($addTag);
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.4: Added Tag Rule

```
rule "addedNote"
  when
    $addNote : EventObject(type=='addNote', $playerid:playerid) from entry-point
     eventstream";
  then
    updateAPI.givePoints($playerid, 'Notes', 1, 'Added Note');
    retract($addNote);
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.5: Added Note Rule

```
rule "addedAvailability"
  when
    $addAvailability : EventObject(type=='addAvailability', $playerid:playerid)
    from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Added Availabilities', 1, 'Added
    Availability');
    retract($addAvailability);
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.6: Added Availability Rule

```
rule "acceptedMeeting"
```

```
  when
    $acceptedMeeting : EventObject(type=='acceptedMeeting', $playerid:playerid)
     from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Accepted Meetings', 1, 'accepted a meeting
     ');
    updateAPI.givePoints($playerid, 'Accepted Meetings in a Row', 1, 'accepted a
      meeting in a row');
    retract($acceptedMeeting);
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.7: Accepted Meeting Rule

```
rule "declinedMeeting"
  when
    $declinedMeeting : EventObject(type=='declinedMeeting', $playerid:playerid)
     from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Declined Meetings', 1, 'declined meeting');
    updateAPI.deleteAllPoints($playerid, 'Accepted Meetings in a Row');
    retract($declinedMeeting);
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.8: Declined Meeting Rule

```
rule "oneToOneLunchAttended"
  when
    $attendedOneToOneTopicLunch : EventObject(type=='attendedMeeting', $playerid
      :playerid, data['meetingType']=='1to1Lunch') from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Attended 1:1 Lunches', 1, 'attended one2one
      lunch');
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.9: 1:1 Lunch Attended Rule

```
rule "fourPersonLunchAttended"
  when
    $attended4PersonLunch : EventObject(type=='attendedMeeting', $playerid:
     playerid, data['meetingType']=='4PersonLunch') from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Attended 4 Person Lunches', 1, 'attended 4
     person lunch');
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.10: 4 Person Lunch Attended Rule

```
rule "coffeeAttended"
  when
    $attendedCoffee : EventObject(type=='attendedMeeting', $playerid:playerid,
     data['meetingType']=='Coffee') from entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Attended Coffees', 1, 'attended coffee');
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.11: Coffee Attended Rule

```
rule "groupTopicLunch"
  when
    $attendedGroupTopicLunch : EventObject(type=='attendedMeeting', $playerid:
    playerid, data['meetingType']=='GroupTopicLunch', $host:data['host']) from
    entry-point eventstream
  then
    updateAPI.givePoints($playerid, 'Attended Group Topic Lunches', 1, 'attended
     group topic lunch');
    if ($host.equals('true')) {
      updateAPI.givePoints($playerid, 'Hosted Group Topic Lunches', 1, 'hosted
    group topic lunch');
    }
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.12: Group Topic Lunch Rule

```
rule "metNewPerson"
  when
    $metPerson : EventObject(type=='metPerson', $playerid:playerid) from entry-
    point internalstream
  then
    updateAPI.givePoints($playerid, 'Met Persons From Different Cost Center', 1,
     'met new person');
    update(queryAPI.getPlayer($playerid));
end
```

Listing B.13: Met New Person Rule

```
rule "readyToNetwork"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($p.getId(), 'Get Ready to Network!') == true)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Added Availabilities').
    getAmount() >= 1)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Tags').getAmount() >= 1)
  then
    updateAPI.completeMission($p.getId(), 'Get Ready to Network!');
    updateAPI.addMissionToPlayer($p.getId(), 'Playing Tag');
    updateAPI.addMissionToPlayer($p.getId(), 'Common Ground');
    updateAPI.addMissionToPlayer($p.getId(), 'Ice Breaker');
    updateAPI.addMissionToPlayer($p.getId(), 'Connect 4');
    updateAPI.addMissionToPlayer($p.getId(), 'Java Joe');
    updateAPI.addMissionToPlayer($p.getId(), 'In the Crowd');
    updateAPI.addMissionToPlayer($p.getId(), 'Emcee');
    updateAPI.addMissionToPlayer($p.getId(), 'Company Explorer');
    updateAPI.addBadgeToPlayer($p.getId(), 'Ready to Network!', 'You filled out
    one availability and tagged yourself with one tag!');
    update($p);
end
```

Listing B.14: Ready To Network Rule

```
rule "tenTags"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($p.getId(), 'Playing Tag') == true)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Tags').getAmount() >= 10)
  then
    updateAPI.completeMission($p.getId(), 'Playing Tag');
    updateAPI.addBadgeToPlayer($p.getId(), 'Playing Tag', 'You tagged yourself
     with 10 tags!');
    updateAPI.addMissionToPlayer($p.getId(), 'The Many Sides of Me');
    update($p);
end
```

Listing B.15: 10 Tags Rule

```
rule "fiveProminentTags"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($p.getId(), 'Common Ground') == true)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Prominent Tags').getAmount()
     >= 5)
  then
    updateAPI.completeMission($p.getId(), 'Common Ground');
    updateAPI.addBadgeToPlayer($p.getId(), 'Common Ground', 'You tagged yourself
      with 5 tags from the 50 most prominent tags!');
    update($p);
end
```

Listing B.16: 5 Prominent Tags Rule

```
rule "twentyTags"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($p.getId(), 'The Many Sides of Me') == true)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Tags').getAmount() >= 20)
  then
    updateAPI.completeMission($p.getId(), 'The Many Sides of Me');
    updateAPI.addBadgeToPlayer($p.getId(), 'The Many Sides of Me', 'You tagged
     yourself with 20 tags!');
    update($p);
end
```

Listing B.17: 20 Tags Rule

```
rule "acceptedFirstMeeting"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($p.getId(), 'On My Calendar') == true)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Accepted Meetings').getAmount
     () >= 1)
  then
    updateAPI.completeMission($p.getId(), 'On My Calendar');
    updateAPI.addBadgeToPlayer($p.getId(), 'On My Calendar', 'You accepted a
     meeting!');
    updateAPI.addMissionToPlayer($p.getId(), 'People Person');
    updateAPI.addMissionToPlayer($p.getId(), 'Shakespeare');
    update($p);
end
```

Listing B.18: Accepted First Meeting Rule

```
rule "accepted5MeetingsInARow"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($p.getId(), 'People Person') == true)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Accepted Meetings in a Row').
     getAmount() >= 5)
  then
    updateAPI.completeMission($p.getId(), 'People Person');
    updateAPI.addBadgeToPlayer($p.getId(), 'People Person', 'You accepted five
     meetings in a row!');
    updateAPI.addMissionToPlayer($p.getId(), 'Disco! 15 in a Row!');
    update($p);
end
```

Listing B.19: Accepted 5 Meetings in a Row Rule

```
rule "firstNote"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Shakespeare') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Notes').getAmount() >= 1)
  then
    updateAPI.completeMission($playerid, 'Shakespeare');
    updateAPI.addBadgeToPlayer($playerid, 'Shakespeare', 'You wrote a note about
      someone you met!');
    update($p);
end
```

Listing B.20: First Note Rule

```
rule "accepted15MeetingsInARow"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($p.getId(), 'Disco! 15 in a Row!') == true)
    eval(queryAPI.getPointsForPlayer($p.getId(), 'Accepted Meetings in a Row').
     getAmount() >= 15)
  then
    updateAPI.completeMission($playerid, 'Disco! 15 in a Row!');
    updateAPI.addBadgeToPlayer($playerid, 'Disco! 15 in a Row!', 'You accepted
     15 meetings in a row!');
    update($p);
end
```

Listing B.21: Accepted 15 Meetings in a Row Rule

```
rule "tenBuddies"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, "I've Got Buds!") == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Buddies').getAmount() >= 10)
  then
    updateAPI.completeMission($playerid, "I've Got Buds!");
    updateAPI.addBadgeToPlayer($playerid, "I've Got Buds!", 'You added 10
     buddies!');
    updateAPI.addMissionToPlayer($playerid, 'Circle of Friends');
    update($p);
end
```

Listing B.22: 10 Buddies Rule

```
rule "twentyfiveBuddies"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Circle of Friends') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Buddies').getAmount() >= 25)
  then
    updateAPI.completeMission($playerid, 'Circle of Friends');
    updateAPI.addBadgeToPlayer($playerid, 'Circle of Friends', 'You added 25
     buddies!');
    update($p);
end
```

Listing B.23: 25 Buddies Rule

```
rule "first1to1LunchAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Ice Breaker') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended 1:1 Lunches').
     getAmount() >= 1)
  then
    updateAPI.completeMission($playerid, 'Ice Breaker');
    updateAPI.addBadgeToPlayer($playerid, 'Ice Breaker', 'You attended a 1:1
     lunch!');
    updateAPI.addMissionToPlayer($playerid, 'Power Luncher');
    update($p);
end
```

Listing B.24: First 1:1 Lunch Attended Rule

```
rule "ten1to1LunchesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Power Luncher') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended 1:1 Lunches').
     getAmount() >= 10)
  then
    updateAPI.completeMission($playerid, 'Power Luncher');
    updateAPI.addBadgeToPlayer($playerid, 'Power Luncher', 'You attended 10 1:1
     lunches!');
    updateAPI.addMissionToPlayer($playerid, 'To Lunchinity and Beyond!');
    update($p);
end
```

Listing B.25: 10 1:1 Lunches Attended Rule

```
rule "twentyfive1to1LunchesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'To Lunchinity and Beyond!') ==
     true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended 1:1 Lunches').
     getAmount() >= 25)
  then
    updateAPI.completeMission($playerid, 'To Lunchinity and Beyond!');
    updateAPI.addBadgeToPlayer($playerid, 'To Lunchinity and Beyond!', 'You
     attended 25 1:1 lunches!');
    update($p);
end
```

Listing B.26: 25 1:1 Lunches Attended Rule

```
rule "first4PersonLunchAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Connect 4') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended 4 Person Lunches').
     getAmount() >= 1)
  then
    updateAPI.completeMission($playerid, 'Connect 4');
    updateAPI.addBadgeToPlayer($playerid, 'Connect 4', 'You attended a 4 person
     lunch!');
    updateAPI.addMissionToPlayer($playerid, '40 People and Climbing');
    update($p);
end
```

Listing B.27: First 4 Person Lunch Attended Rule

```
rule "ten4PersonLunchesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, '40 People and Climbing') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended 4 Person Lunches').
     getAmount() >= 10)
  then
    updateAPI.completeMission($playerid, '40 People and Climbing');
    updateAPI.addBadgeToPlayer($playerid, '40 People and Climbing', 'You
     attended 10 4 person lunches!');
    updateAPI.addMissionToPlayer($playerid, 'Epic Luncher');
    update($p);
end
```

Listing B.28: 10 4 Person Lunches Attended Rule

```
rule "twentyfive4PersonLunchesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Epic Luncher') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended 4 Person Lunches').
     getAmount() >= 25)
  then
    updateAPI.completeMission($playerid, 'Epic Luncher');
    updateAPI.addBadgeToPlayer($playerid, 'Epic Luncher', 'You attended 25 4
     person lunches!');
    update($p);
end
```

Listing B.29: 25 4 Person Lunches Attended Rule

```
rule "firstCoffeeAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Java Joe') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended Coffees').getAmount()
     >= 1)
  then
    updateAPI.completeMission($playerid, 'Java Joe');
    updateAPI.addBadgeToPlayer($playerid, 'Java Joe', 'You attended a coffee
     meeting!');
    updateAPI.addMissionToPlayer($playerid, 'Coffee Chatter');
    update($p);
end
```

Listing B.30: First Coffee Attended Rule

```
rule "tenCoffeesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Coffee Chatter') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended Coffees').getAmount()
     >= 10)
  then
    updateAPI.completeMission($playerid, 'Coffee Chatter');
    updateAPI.addBadgeToPlayer($playerid, 'Coffee Chatter', 'You attended 10
     coffee meetings!');
    updateAPI.addMissionToPlayer($playerid, 'Caffene Devotee');
    update($p);
end
```

Listing B.31: 10 Coffees Attended Rule

```
rule "twentyfiveCoffeesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Caffene Devotee') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended Coffees').getAmount()
     >= 25)
  then
    updateAPI.completeMission($playerid, 'Caffene Devotee');
    updateAPI.addBadgeToPlayer($playerid, 'Caffene Devotee', 'You attended 25
     coffee meetings!');
    update($p);
end
```

Listing B.32: 25 Coffees Attended Rule

```
rule "firstGroupTopicLunchAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'In the Crowd') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended Group Topic Lunches').
     getAmount() >= 1)
  then
    updateAPI.completeMission($playerid, 'In the Crowd');
    updateAPI.addBadgeToPlayer($playerid, 'In the Crowd', 'You attended a group
     topic lunch!');
    updateAPI.addMissionToPlayer($playerid, 'Topic Talker');
    update($p);
end
```

Listing B.33: First Group Topic Lunch Attended Rule

```
rule "fiveGroupTopicLunchesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Topic Talker') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended Group Topic Lunches').
     getAmount() >= 5)
  then
    updateAPI.completeMission($playerid, 'Topic Talker');
    updateAPI.addBadgeToPlayer($playerid, 'Topic Talker', 'You attended 5 group
     topic lunches!');
    updateAPI.addMissionToPlayer($playerid, 'Interested in Everything');
    update($p);
end
```

Listing B.34: 5 Group Topic Lunches Attended Rule

```
rule "tenGroupTopicLunchesAttended"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Interested in Everything') ==
     true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Attended Group Topic Lunches').
     getAmount() >= 10)
  then
    updateAPI.completeMission($playerid, 'Interested in Everything');
    updateAPI.addBadgeToPlayer($playerid, 'Interested in Everything', 'You
     attended 10 group topic lunches!');
    update($p);
end
```

Listing B.35: 10 Group Topic Lunches Attended Rule

```
rule "firstGroupTopicLunchHosted"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Emcee') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Hosted Group Topic Lunches').
     getAmount() >= 1)
  then
    updateAPI.completeMission($playerid, 'Emcee');
    updateAPI.addBadgeToPlayer($playerid, 'Emcee', 'You hosted a group topic
     lunch!');
    updateAPI.addMissionToPlayer($playerid, 'Talkshow Host');
    update($p);
end
```

Listing B.36: First Group Topic Lunch Hosted Rule

```
rule "fiveGroupTopicLunchesHosted"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Talkshow Host') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Hosted Group Topic Lunches').
     getAmount() >= 5)
  then
    updateAPI.completeMission($playerid, 'Talkshow Host');
    updateAPI.addBadgeToPlayer($playerid, 'Talkshow Host', 'You hosted 5 group
     topic lunches!');
    updateAPI.addMissionToPlayer($playerid, 'Master of Ceremonies');
    update($p);
end
```

Listing B.37: 5 Group Topic Lunches Hosted Rule

```
rule "tenGroupTopicLunchesHosted"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Master of Ceremonies') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Hosted Group Topic Lunches').
     getAmount() >= 10)
  then
    updateAPI.completeMission($playerid, 'Master of Ceremonies');
    updateAPI.addBadgeToPlayer($playerid, 'Master of Ceremonies', 'You hosted 10
      group topic lunches!');
    update($p);
end
```

Listing B.38: 10 Group Topic Lunches Hosted Rule

```
rule "met1PersonFromDifferentCostCenter"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Company Explorer') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Met Persons From Different Cost
      Center').getAmount() >= 1)
  then
    updateAPI.completeMission($playerid, 'Company Explorer');
    updateAPI.addBadgeToPlayer($playerid, 'Company Explorer', 'You met someone
    from a different department/cost center!');
    updateAPI.addMissionToPlayer($playerid, 'Silo Breaker');
    update($p);
end
```

Listing B.39: Met 1 Person From Different Cost Center Rule

```
rule "met10PersonsFromDifferentCostCenters"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Silo Breaker') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Met Persons From Different Cost
      Center').getAmount() >= 10)
  then
     updateAPI.completeMission($playerid, 'Silo Breaker');
    updateAPI.addBadgeToPlayer($playerid, 'Silo Breaker', 'You met 10 people
    from different departments/cost centers!');
    updateAPI.addMissionToPlayer($playerid, 'Emissary');
    update($p);
end
```

Listing B.40: Met 10 Persons From Different Cost Centers Rule

```
rule "met25PersonsFromDifferentCostCenters"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Emissary') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Met Persons From Different Cost
      Center').getAmount() >= 25)
  then
    updateAPI.completeMission($playerid, 'Emissary');
    updateAPI.addBadgeToPlayer($playerid, 'Emissary', 'You met 25 people from
    different departments/cost centers!');
    updateAPI.addMissionToPlayer($playerid, 'Ambassador');
    update($p);
end
```

Listing B.41: Met 25 Persons From Different Cost Centers Rule

```
rule "met50PersonsFromDifferentCostCenters"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerMission($playerid, 'Ambassador') == true)
    eval(queryAPI.getPointsForPlayer($playerid, 'Met Persons From Different Cost
      Center').getAmount() >= 50)
  then
    updateAPI.completeMission($playerid, 'Ambassador');
    updateAPI.addBadgeToPlayer($playerid, 'Ambassador', 'You met 50 people from
    different departments/cost centers!');
    update($p);
end
```

Listing B.42: Met 50 Persons From Different Cost Centers Rule

```
rule "attended4DifferentMeetingTypes"
  when
    $p : Player($playerid : uid)
    eval(queryAPI.hasPlayerBadge($playerid, 'Ice Breaker') == true)
    eval(queryAPI.hasPlayerBadge($playerid, 'Connect 4') == true)
    eval(queryAPI.hasPlayerBadge($playerid, 'Java Joe') == true)
    eval(queryAPI.hasPlayerBadge($playerid, 'In the Crowd') == true)
    eval(queryAPI.hasPlayerBadge($playerid, 'Attended 4 Different Meeting Types
    ') == false)
  then
    updateAPI.addBadgeToPlayer($playerid, 'Attended 4 Different Meeting Types',
    'You attended 4 different meeting types!');
    update($p);
end
```

Listing B.43: Attended 4 Different Meeting Types Rule

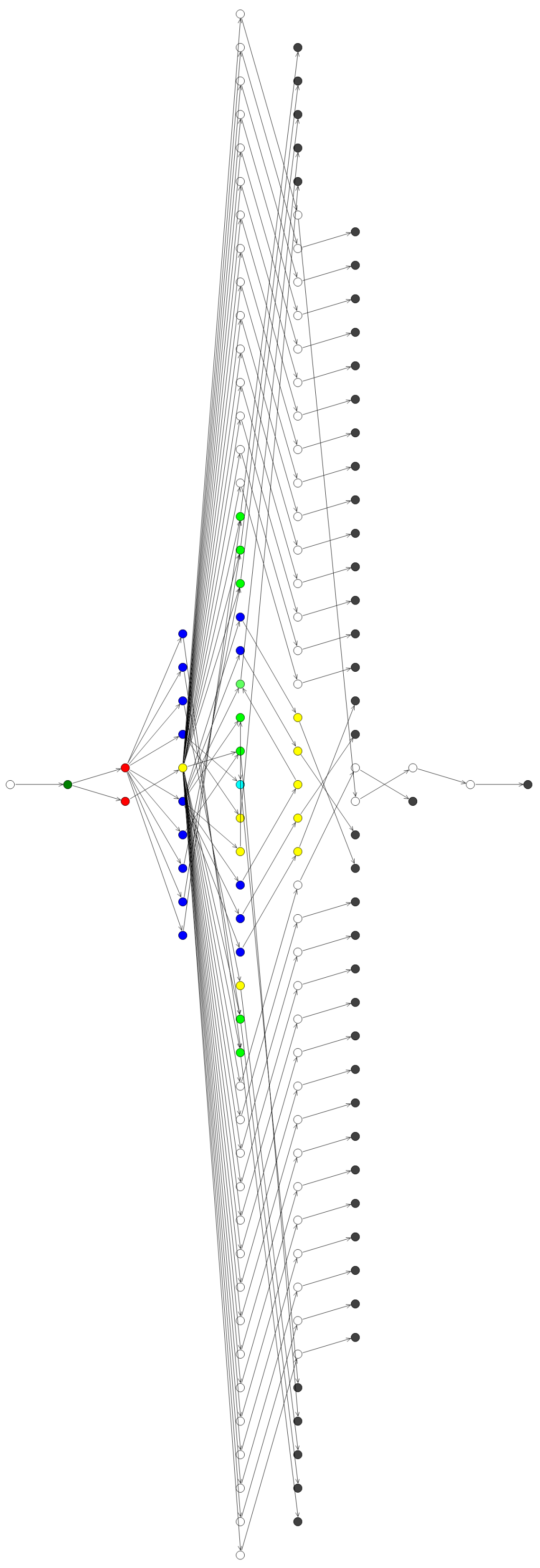


Figure B.1: RETE Tree for Application Rules

Appendix C

R Skript

```r
library(gplots)
library(cvTools)
require(graphics)
library(vcd)

root_path<-'C:/Users/D059598/Desktop/Thesis/Users/async/'
mode<-"ASYNC"
users<-seq(from = 50, to = 1000, by = 50)

simpleTests<-FALSE
factorName<-"Users"

getTotalRuntime<-function(data) {
  result<-data$S2E+data$Sync+data$RuleEngine+data$E2A;
  result;
}

bestModelWithAIC<-function(name,a,b,mean,med) {
  polynom = 0
  model = NULL
  bestModel = NULL
  while (is.null(bestModel) || (AIC(bestModel) > AIC(model) && !is.infinite(AIC(
    model)))) {
# go on finding better value
    if (!is.null(model)) {
      bestModel <- model
    }
    polynom = polynom+1
    model <- lm(b~poly(a, polynom, raw=TRUE))
  }
  plot(users, med, main=paste(name, " - ", polynom-1,". Polynomial (AIC)" , sep=
    ""), xlab=factorName, ylab="Time in ms", type="b", pch=2, ylim=c(min(mean,
    med),max(mean,med)))
  lines(users, mean, type="b", pch=1)
  points(a, predict(bestModel), type="l", col="blue", lwd=2)
  legend("topleft", ncol = 3, c("Median", "Mean", "Model"), pch=c(2,1,20), col=c
    ("black","black","blue"), text.col=c("black","black","blue"))

  #print also in pdf picture files...
  pdf(paste(root_path,name,"_AIC_",min(users),"-",max(users),".pdf",sep=""),
    width=6, height=4.5)
  plot(users, med, main=paste(name, " - ", polynom-1,". Polynomial (AIC)" , sep=
    ""), xlab=factorName, ylab="Time in ms", type="b", pch=2, ylim=c(min(mean,
    med),max(mean,med)))
  lines(users, mean, type="b", pch=1)
  points(a, predict(bestModel), type="l", col="blue", lwd=2)
  legend("topleft", ncol = 3, c("Median", "Mean", "Model"), pch=c(2,1,20), col=c
    ("black","black","blue"), text.col=c("black","black","blue"))
  dev.off()

  bestModel;
}

bestModelWithBIC<-function(name,a,b,mean,med) {
  polynom = 0
  model = NULL
  bestModel = NULL
  while (is.null(bestModel) || (BIC(bestModel) > BIC(model) && !is.infinite(BIC(
    model)))) {
# go on finding better value
    if (!is.null(model)) {
      bestModel <- model
    }
    polynom = polynom+1
    model <- lm(b~poly(a, polynom, raw=TRUE))
  }
  plot(users, med, main=paste(name, " - ", polynom-1,". Polynomial (BIC)" , sep=
    ""), xlab=factorName, ylab="Time in ms", type="b", pch=2, ylim=c(min(mean,
    med),max(mean,med)))
  lines(users, mean, type="b", pch=1)
  points(a, predict(bestModel), type="l", col="blue", lwd=2)
  legend("topleft", ncol = 3, c("Median", "Mean", "Model"), pch=c(2,1,20), col=c
    ("black","black","blue"), text.col=c("black","black","blue"))

  #print also in pdf picture files...
  pdf(paste(root_path,name,"_BIC_",min(users),"-",max(users),".pdf",sep=""),
    width=6, height=4.5)
  plot(users, med, main=paste(name, " - ", polynom-1,". Polynomial (BIC)" , sep=
    ""), xlab=factorName, ylab="Time in ms", type="b", pch=2, ylim=c(min(mean,
    med),max(mean,med)))
  lines(users, mean, type="b", pch=1)
  points(a, predict(bestModel), type="l", col="blue", lwd=2)
  legend("topleft", ncol = 3, c("Median", "Mean", "Model"), pch=c(2,1,20), col=c
    ("black","black","blue"), text.col=c("black","black","blue"))
  dev.off()

  bestModel;
}

bestModelWithCrossValidation<-function(name,a,b,mean,med) {
  polynom = 0
  model = NULL
  bestModel<-NULL
  finished<-FALSE
  cvBestModel<-0
  cvModel<-0

  while (is.null(bestModel) || cvBestModel > cvModel) {
  # go on finding better value
    if (!is.null(model)) {
      bestModel <- model
      cvBestModel <-cvModel
    }
    polynom = polynom+1
    model <- lm(b~poly(a, polynom, raw=TRUE), x=TRUE, y=TRUE)
    c<-poly(a, polynom, raw=TRUE)
    cvModel<-cvFit(lm, formula = b~c, data = data.frame(a,b), y = b, K = 5)$cv
  }
  plot(users, med, main=paste(name, " - ", polynom-1,". Polynomial (CV)" , sep=""), xlab=factorName, ylab="Time in ms", type="b", pch=2, ylim=c(min(mean,med),max(mean,med)))
  lines(users, mean, type="b", pch=1)
  points(a, predict(bestModel), type="l", col="blue", lwd=2)
  legend("topleft", ncol = 3, c("Median", "Mean", "Model"), pch=c(2,1,20), col=c("black","black","blue"), text.col=c("black","black","blue"))

  #print also in pdf picture files...
  pdf(paste(root_path,name,"_CV_",min(users),"-",max(users),".pdf",sep=""), width=6, height=4.5)
  plot(users, med, main=paste(name, " - ", polynom-1,". Polynomial (CV)" , sep=""), xlab=factorName, ylab="Time in ms", type="b", pch=2, ylim=c(min(mean,med),max(mean,med)))
  lines(users, mean, type="b", pch=1)
  points(a, predict(bestModel), type="l", col="blue", lwd=2)
  legend("topleft", ncol = 3, c("Median", "Mean", "Model"), pch=c(2,1,20), col=c("black","black","blue"), text.col=c("black","black","blue"))
  dev.off()

  bestModel;
}

modelComponent<-function(component, name) {
  mean<-vector()
  med<-vector()
  userValues<-vector()
  allValues<-vector()
  for(x in 1:length(users)) {
    assign(paste(component,x,sep=""),subset(eval(as.symbol(paste("data", x, sep=""))), Component==component))
    plotData<-eval(as.symbol(paste(component, x, sep="")))$ms
    if (nrow(eval(as.symbol(paste(component, x, sep="")))) > 0) {
      hist(plotData, main=paste("Histogram",name,"Time,",users[x],factorName), xlab="time in ms", ylab="Quantity")
      mean <- append(mean, mean(plotData))
      med <- append(med, median(plotData))
    }
    userValues <- append(userValues, rep(users[x],length(plotData)))
    allValues <- append(allValues, plotData)
  }
  if (length(mean) > 0) {
    plot(users, mean, main=paste("Mean in",name), xlab=factorName)
    plot(users, med, main=paste("Median in",name), xlab=factorName)

    model2<-bestModelWithAIC(name,users,mean,mean,med)
    out<-capture.output(summary(model2))
    cat(out,file=paste(root_path,component,"-AIC_",min(users),"-",max(users),".
      txt",sep=""),sep="\n",append=FALSE)

    model3<-bestModelWithBIC(name,users,mean,mean,med)
    out<-capture.output(summary(model3))
    cat(out,file=paste(root_path,component,"-BIC_",min(users),"-",max(users),".
      txt",sep=""),sep="\n",append=FALSE)

    model5<-bestModelWithCrossValidation(name,users,mean,mean,med)
    out<-capture.output(summary(model5))
    cat(out,file=paste(root_path,component,"-CV_",min(users),"-",max(users),".
      txt",sep=""),sep="\n",append=FALSE)
  }
}

if (simpleTests == TRUE) {
tryCatch({
  for(x in users) assign(paste("data",x,sep=""), read.csv(paste(root_path, "
    result_new_", x, "User.csv", sep=""), header=TRUE, sep=";"))

  for(x in 1:length(users)) {
    assign(paste("errors",x,sep=""), read.csv(paste(root_path, "errorRate_", x,
      ".csv", sep=""), header=TRUE, sep=";"))
    assign(paste("errors",x,"actual",sep=""), eval(as.symbol(paste("errors",x,
      sep="")))$sum.actual.experience.points)
    assign(paste("errors",x,"generated",sep=""), eval(as.symbol(paste("errors",x
      ,sep="")))$sum.generated.experience.points)
  }
  x<-users;

  # total runtimes calculations
  mins<-vector()
  maxs<-vector()
  meds<-vector()
  avgs<-vector()
  percentile25s<-vector()
  percentile75s<-vector()
  percentile99s<-vector()
  responsesOver500ms<-vector()
  responsesOver1000ms<-vector()
  responsesOver500msRelative<-vector()
  responsesOver1000msRelative<-vector()

  for(i in c("1000","100000"))
  {
    runtimes<-subset(eval(as.symbol(paste0("data",i))), Component=="Total")$ms
    mins<-append(mins,min(runtimes))
    maxs<-append(maxs,max(runtimes))
    meds<-append(meds,median(runtimes))
    avgs<-append(avgs,mean(runtimes))
    percentiles<-quantile(runtimes, c(.25, .75, .99))
    percentile25s<-append(percentile25s,percentiles[1])
    percentile75s<-append(percentile75s,percentiles[2])
    percentile99s<-append(percentile99s,percentiles[3])
    responsesOver500ms<-append(responsesOver500ms,sum(runtimes>500))
    responsesOver1000ms<-append(responsesOver1000ms,sum(runtimes>1000))
    responsesOver500msRelative<-append(responsesOver500msRelative,sum(runtimes
      >500)/length(runtimes))
    responsesOver1000msRelative<-append(responsesOver1000msRelative,sum(runtimes
      >1000)/length(runtimes))
    assign(paste("runtime", i, sep=""),runtimes)
  }
  t.test(runtime1000,runtime100000)
  wilcox.test(runtime1000,runtime100000)

# wm growth
  pdf(paste(root_path,"ResponseTime.pdf",sep=""), width=4.5, height=4.5, title=
    root_path)
  par(mfrow=c(1,1))
  t.test(runtime1[1:20],runtime1[1:2000])
  dev.off()

  pdf(paste(root_path,"DistributionOfResponseTimes.pdf",sep=""), width=9, height
    =4.5, title=root_path)
  par(mfrow=c(1,2))
  hist(runtime1, main="(a) Alpha Nodes: 10", xlab="Response Time in ms")
  hist(runtime31, main="(b) Alpha Nodes: 300", xlab="Response Time in ms")
  dev.off()

  pdf(paste(root_path,"Errors.pdf",sep=""), width=4.5, height=4.5, title=root_
    path)
  par(mfrow=c(1,1))
  barplot(c(errors4actual,errors3actual,errors2actual,errors1actual), ylab="
    Number of Points",xlab="Memory Size",names.arg=c("64MB","256MB","1024MB","8
    GB"))
  dev.off()

  pdf(paste(root_path,"EventTypeDistribution.pdf",sep=""), width=9, height=4.5,
    title=root_path)
  par(mfrow=c(1,2))
  barplot(c(10,490), names.arg = c("addBuddy","addTag"), ylab="Amount of Events"
    , main = "(a) Unbalanced")
  barplot(c(250,250), names.arg = c("addBuddy","addTag"), ylab="Amount of Events
    ", main = "(b) Balanced")
  dev.off()

#interarrival times
  interarrival<-vector()
  assign("sender", read.csv(paste(root_path, "sender200.csv", sep=""), header=
    TRUE, sep=";"))
   #filter by type
  sender<-subset(sender, Type=="addBuddy")
  sender<-sender[order(sender$Timestamp),]
  for(x in 2:nrow(sender)) interarrival<-append(interarrival,(sender$Timestamp[x
    ]-sender$Timestamp[x-1])/1000000)
  li <- 1/mean(interarrival)
   #median(interarrival)
```

```r
  hist(interarrival, breaks=100, freq=FALSE, main="Interarrival Time Distribution", xlab="Interarrival Time")

  pdf(paste(root_path,"InterarrivalTimeDistribution.pdf",sep=""), width=4.5, height=4.5, title=root_path)
  par(mfrow=c(1,1))
  hist(interarrival, breaks=500, xlim=c(0,5000), freq=FALSE, main="Interarrival Time Distribution", xlab="Interarrival Time in ms")
  curve(li*exp(1)^(-li*x), from=0, to=25000, add=T, lwd=2, col="blue")
  legend("topright", ncol = 1, c(paste0("exp. distr. lambda=",round(x=li, digits=10))), pch=c(20), col=c("blue"), text.col=c("blue"))
  dev.off()


  component<-"Total"
  a<-subset(data200, Component==component)$ms
  mean(a)
  median(a)
  sd(a)
  hist(a, freq=FALSE, breaks=20, main=paste("Histogram of Service Time in", component))
  li <- 1/mean(a)
  curve(li*exp(1)^(-li*x), from=0, to=max(a), add=T, lwd=2, col="blue")

  component<-"BEP_Query"
  factorName<-"BEP Query"

# histograms for components
  pdf(paste(root_path,"HistogramBEPUpdate.pdf",sep=""), width=6, height=4.5, title=root_path)
  par(mfrow=c(1,1))

  x<-1
  assign(paste(component,x,sep=""),subset(eval(as.symbol(paste("data", x, sep="")))), Component==component))
  plotData<-eval(as.symbol(paste(component, x, sep="")))$ms
  hist(plotData, main=paste("(g)",factorName,"-",users[x],"Users"), xlab="time in ms", breaks=20, freq=FALSE)
  li <- 1/mean(plotData)
  #exp. distr.
  curve(li*exp(1)^(-li*x), from=0, to=max(plotData), add=T, lwd=2, col="blue")
  # poisson
  curve(dnorm, col = 2, add = TRUE)
  curve(dpois(x,mean(plotData), log = FALSE), from=0, to=max(plotData), add=T, lwd=2, col="blue")
  legend("topright", ncol = 1, c(paste0("exp. distr. lambda=",round(x=li, digits=10))), pch=c(20), col=c("blue"), text.col=c("blue"))
  dev.off()

  x<-6
  assign(paste(component,x,sep=""),subset(eval(as.symbol(paste("data", x, sep="")))), Component==component))
  plotData<-eval(as.symbol(paste(component, x, sep="")))$ms
  hist(plotData, main=paste("(a)",users[x],factorName), xlab="time in ms", ylab="Quantity")
  x<-12
  assign(paste(component,x,sep=""),subset(eval(as.symbol(paste("data", x, sep=""))), Component==component))
  plotData<-eval(as.symbol(paste(component, x, sep="")))$ms
  hist(plotData, main=paste("(a)",users[x],factorName), xlab="time in ms", ylab="Quantity")
  dev.off()

}, warning = function(war) {
  print(war)
}, error = function(err) {
  print(err)
}, finally = {
  print("run sucessfully")
})
}

tryCatch({
  for(x in 1:length(users)) assign(paste("data",x,sep=""), read.csv(paste(root_path, "result_new_", users[x], "User.csv", sep=""), header=TRUE, sep=";"))
  pdf(paste(root_path,"/Charts_",min(users),"-",max(users),".pdf",sep=""),paper="a4", width=12, height=20, title=root_path)
  par(mfrow=c(4,2))
  if (mode == "ASYNC") {
    modelComponent("MB1","Message Broker I")
    modelComponent("CEP","CEP")
    modelComponent("PRO","Proxy")
    modelComponent("MB2","Message Broker II")
    modelComponent("CEPC_Query","CEPC Query")
    modelComponent("CEPC_Update","CEPC Update")
    modelComponent("BEP_Query","BEP Query")
    modelComponent("BEP_Update","BEP Update")
    modelComponent("Total","Total")
  } else {
    modelComponent("CEP","CEP")
    modelComponent("CEP_QR","CEP Query")
    modelComponent("PRO","Proxy")
    modelComponent("BEP_Query","BEP Query")
    modelComponent("BEP_Update","BEP Update")
    modelComponent("Total","Total")
  }
  dev.off()
}, warning = function(war) {
  print(war)
}, error = function(err) {
  print(err)
}, finally = {
  print("run sucessfully")
})
```

Listing C.1: R Skript