Online Performance Anomaly Detection

for Large-Scale Software Systems

OPAD

Diploma Thesis

Christian-Albrechts-Universität zu Kiel

Author

Tillmann Carlos Bielefeld

Advisory

Prof. Dr. Wilhelm Hasselbring
Dipl.-Inform. André van Hoorn
Dr. Stefan Kaes (XING AG)

Case Study at

XING AG

Submitted on March 28, 2012

Abstract

Provisioning satisfying *Quality of Service (QoS)* is a challenge when operating large-scale software systems. Performance and availability are important metrics for QoS, especially for Internet applications. Since these systems are accessed concurrently, bad performance can manifest itself in slow response time for all users simultaneously.

Software solutions for monitoring these metrics exist and abnormal behavior in the performance is often analyzed later for future improvement. However, in interactive applications, users notice anomalies immediately and reactions require automatic online detection. This is hard to achieve since large-scale applications are operated in grown, unique environments. These domains often include a network of subsystems with system-specific measures and characteristics. Thus, anomaly detection is hard to establish as it requires a custom setup for each system.

This work approaches these challenges by implementing means for online anomaly detection based on time series analysis, called Θ**PAD**. In a monitoring server different algorithms can be configured and evaluated in order to address system-specific characteristics.

The software is designed as a plugin for the performance monitoring and dynamic analysis framework Kieker. With the use of selected algorithms, it can detect and signal anomalies online and store them for post-mortem analyses. The social network system XING served as a case study and the evaluation of Θ**PAD** in this production environment shows promising results in terms of robustness and accuracy.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Software as a Service (SaaS) has gained much attraction in the past years, especially since 2006 [Kaplan 2008, p. 1]. By now, many big software vendors invest in building software that is served from distant data centers. Remarkable is the competition between SAP and Oracle for market share in the field of *cloud computing*. Many innovative tech companies already got swallowed by these two big players [Reuters 2012].

A big challenge for businesses offering services over the Internet is the *trustworthiness* towards users. One technical aspect of trust is the *Quality of Service (QoS)*. This term can be defined as a combination of the three factors *availability*, *reliability*, and *performance* [Becker et al. 2006, p. 8]. Hasselbring [2008] correlates the user's trust in Internet-based applications with the availability of their services. This is verified by a recent survey showing that 23% of the SaaS customers are still concerned about availability and performance [Kaplan 2008, p. 2].

Situations in which availability or performance are compromised can cause severe damage to business-customer relationships. In the growing market of SaaS, competitors can quickly build up competing services and lure unsatisfied customers away. In extreme cases costly law suits, e.g., for unmatched *service level agreements (SLAs)* or even the break down of the whole business model can be the consequence. In order to gain and keep the user's trust, production systems have to be under continuous monitoring and actions have to be taken immediately whenever low availability or bad performance could compromise QoS.

There is a variety of software packages for application-level performance monitoring. Examples are the commercial products New Relic and AppDynamics, or the open source framework Kieker [van Hoorn et al. 2012, p. 1]. For IT infrastructure monitoring, Nagios offers system-level availability monitoring and automatic alerting.

However, software systems evolve over time and get unique system-specific performance *measures* and characteristics. Since these measures are influenced by a variety of factors, interpretation of behavior requires certain knowledge about

the performance characteristics such as the distribution of the system's architecture. This is especially important for detecting abnormal behavior as such [Bechtold and Heinlein 2004, p. 44]. Apart from the costly and error-prone approach of analyzing performance graphs manually, the problem is often left unsolved.

This challenge of QoS also exists for systems providing *social network* services with their basic purpose being the communication and central data exchange. The so-called 'social web' (or *Web 2.0*) [O'Reilly 2005, p. 1] began to influence our lives considerably. Most of the early launched social networks that are still online today started as *business-to-customer (B2C)* platforms [Boyd 2007, p. 8]. However, the high attention of these applications attracted and lead to the development of many companies that now offer additional services. Hence, social network providers entered the B2B field as well. The most prominent example, Facebook, has an economic impact of € 15.3 billion according to Deloitte LLP [2012, p. 3].

The social network XING was dedicated to business customers from the beginning and grew to be one of the largest professional social networks and now serves millions of users [XING AG 2012] internationally. Its large-scale system architecture is monitored continuously by dedicated software. However, a system for automatic detection of abnormal behavior in performance has not been implemented yet. To address this need for online performance anomaly detection, this work develops an approach and implements it in software. Finally, a practical evaluation is conducted in a case study for XING's large-scale software system.

This diploma thesis investigates available performance anomaly detection algorithms based on time series analysis and furthermore the design, implementation, and evaluation of Θ**PAD**: an approach dealing with these issues and fulfilling the requirements. The implementation in software would need to address the needs of the the case study in practice and be configurable for other environments as well. The requirements are the following:

- Gathering different system-specific *measurements*.
- Offering multiple and adjustable anomaly detection approaches based on time-series analysis.
- Running and testing these approaches simultaneously and permanently.
- Provide *robustness* and availability for itself.

The implementation of Θ**PAD** is a server that is built on top of Kieker, a framework for application performance management and dynamic analysis. Θ**PAD** offers different anomaly detection algorithms that can be configured for the particular domain. For the case study, a particular configuration is found and evaluated in a long-term-test on real data from the production system described in the remainder of this section.

Case Study Overview

The global social network XING is the most important platform for business contacts in Europe. It has more than 11.7 million registered users [XING AG 2012] communicating in 40 thousand forum-like expert groups and every year more than

180 thousand events where members meet in person [XING AG 2010, p. 7] are organized in its communities.

XING's biggest revenue stream comes from the paid membership: In subscriptions, members can get premium features up from € 4.95 per month. As of the latest investor fact sheet of XING AG [2011, p. 3], the company hosts 779,000 paid memberships; many of whose businesses depend on the network in different ways. Since the platform's most valuable asset is its user base, over 100 engineers (out of 420 employees in total) maintain and operate this large-scale software system.[1]

When XING started as openBC in November 2003 the platform's software was written in Perl. In the course of openBC's lifetime, the user base grew and a solid architecture hosted in a data center in Hamburg was built. After the rebranding to XING the software evolved further and new technologies such as Ruby on Rails and data base *sharding* [Chodorow 2011, p. 5] were introduced.

In summer 2011, XING ran hundreds of physical machines in two data centers. The architecture comprises monitoring for the hardware and software and uses Nagios as an automatic notification system in case of system failures. Furthermore, a dedicated performance monitoring software was developed for system-level and application-level monitoring. This software, called Logjam, co-authored by Stefan Kaes, provided online data that is being observed by system administrators. When supervising Logjam, they can react to abnormal behavior immediately. However, a form of automatic detection does not exist. Apart from business hours, nobody would notice, when the XING platform would respond slowly or behave strangely.

This diploma thesis aims at developing a solution to this problem. Existing research done by the chair of Software Engineering at the University of Kiel is added to the existing monitoring system, Kieker, and be configured, installed, and evaluated at the case-study environment.

1.2 Goals

The goals of this thesis, as defined in a proposal submitted in July 2011, are as following:

▷ `T1`: Design of an Online Performance Anomaly Detection Concept

The first goal is to work toward the idea of detecting anomalies in the performance of large-scale software systems. Since the detection will be online, the abbreviation Θ**PAD** (online performance anomaly detection) will be used for the approach and the later implementation. It will also address the motivation described in Section 1.1. The concept will include research-based evidence as well as a proof-of-concept implemented in '$\mathcal{R}$ Project for Statistical Computing' [R Development Core Team 2011] that calculates anomaly scores from sample data. In order to achieve this, algorithms will be designed, implemented, and tested with various sample data. This basis will be used in the plugin (`T2`) later on.

[1] According to a personal conversation with XING's CTO Jens Pape on February 3rd, 2012

▷ **T2: ΘPAD Implementation as a Kieker Plugin**

In order to create software that detects anomalies and is usable in production environments, the existing Kieker Framework [van Hoorn et al. 2012] will be used and extended. Kieker is a mature framework with a plugin architecture, which makes it easy to extend. Since Kieker can serve as a platform for "specific project contexts" [van Hoorn et al. 2009, p. 1]. **ΘPAD** will be implemented as a plugin for this framework.

▷ **T3: ΘPAD Integration with Case-study System**

This goal is to determine the integration of the **ΘPAD** Kieker plugin into the case-study system. It has to adapt to the log format produced by its application servers of the case study. Eventually it will send the measurements combined with the calculated anomaly score to an alerting facility.

▷ **T4: Evaluation**

The anomaly detection results can vary in terms of accuracy and precision. It is not yet clear if they will be as effective as a person detecting anomalies manually. Furthermore, the underlying *forecast* model has to be evaluated if the calculated data match the real *online* data. A long-term test will tell if the approach is practicable to detect anomalies in a system's performance automatically and how the different factors, such as varying workload intensity, can be causes for anomalies.

1.3 Document Organization

Formalia

The following typographical conventions are used throughout this thesis:

Definitions and Special words
: First occurrences of definitions appear in *italic*. Subsequent use of these terms appear in normal font. Additionally, words with special meanings are indicated in the same way.

Software Libraries and Products
: referenced in the text appear emphasized. Further resources to these names are provided in the same-named glossary on page 117. Descriptions are taken from either the sections or referenced `URL`s in the glossary and are therefore not cited directly.

`Programmatic Terms`
: use the `typewriter` font to indicate parts of the source code or library names.

`URLs`
: appear in `typewriter` font with constant width and are linked in the PDF. They are listed in footnotes and bibliography only.

[References]
: are cited with details wherever possible. [Herbst 2012, p. 15], for instance, refers to a statement on page 15. On web pages without page numbering, sections are indicated: [van Kesteren 2012, Section 4.7].

Logos
: are represented graphically. They might deviate from official corporate identity in order to present them in a more readable format.
XING refers to the XING AG and its product, $\mathcal{R}$ to the equally-named programming language, and Θ**PAD** refers to the approach developed in this thesis and the corresponding implementation in software.

Document Structure

The remainder of this thesis is structured into the following:

- The upcoming Chapter, 2, states the mathematical foundations of anomaly detection and introduces commonly used web technologies of the Web 2.0. Additionally, it describes, which technology stack is used for the implementation and in the XING case study environment.
- Chapter 3 introduces the Θ**PAD** approach for online performance anomaly detection. In that chapter, *activity diagrams*, as specified in the *Unified Modelling Language (UML)* [Rupp et al. 2007, p. 267], that are used in practice for functional specification, describe the approach formally.
- The succeeding Chapter 4 explains, how this formal specification is designed and implemented in a software prototype. This documentation uses UML class diagrams to explain the transformation from data types and abstract concepts into software components. Furthermore, the development of Θ**PAD** as a Kieker plugin is described alongside structure diagrams and software engineering methods.
- Chapter 5 describes the configuration and adaption of Θ**PAD** in the case-study environment. Observed anomalies are compared against the *online* detection with the software. Metrics for detection efficiency are used to demonstrate the applicability of the approach in practice.
- Finally, Chapter 6 gives a summary and a critical overview of the lessons learned throughout the evaluation. It also points to possible future usage in other environments.

Chapter 2

Foundations

The introduction motivated the problem field of abnormal behavior in the performance of software systems. Developing a solution capable of detecting these performance anomalies can be supported by certain terms and concepts which are described in this chapter.

First, fundamental terms like performance are defined (Section 2.1), followed by the mathematical operations on time series. Forecasting of performance measures is explained in Section 2.2 alongside the most important algorithms. These algorithms are required to facilitate anomaly detection as described in Section 2.3. Throughout these first sections an example time series with invented values is constructed to support the mathematics. Section 2.4 introduces the technology that the Θ**PAD** implementation is built upon. Furthermore, that section explains important technologies in the field of web applications and large architectures. Some of them gained popularity with the rise of the Web 2.0 in the last years. The description of the monitoring framework Kieker is followed by am overview of the case-study environment.

2.1 Performance Metrics

Performance, according to Smith and Williams [2002, p. 4], is the degree to which a system meets its timeliness. Another definition is given by Koziolek [2008, Section 17.1] as the “time behavior and resource efficiency" and is equivalent to *efficiency*. That term is used by the ISO/IEC 9126 standard for *internal and external quality* [ISO/IEC 2001, p. 41].

Following are basic definitions and metrics of system performance analysis.

- ***Measurand***: Object, which gets values assigned in the measuring process.
- ***Measure***: An algorithm producing measurements from measurands.
- ***Measurement***: A comparable value produced by a measure. Informally, it is also used as the ‘process of measuring’. We define a ***raw measurement*** as coming from the monitored system (see Section 2.2.2).

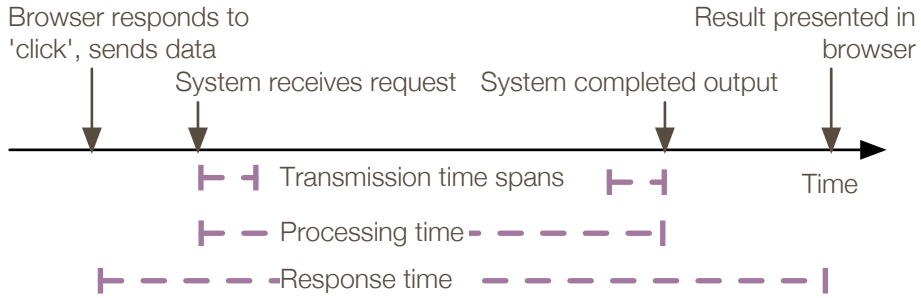

Figure 2.1: According to Jain [1991, p. 37], these time spans can be measured within the interaction between the user's browser and a system.

These terms are part of the *Structured Metrics Metamodel* as defined by OMG [2012, p. 2]. This terminology will be used throughout this thesis to reason about system performance.

Metrics, which correlate to the normal behavior of a system, can define a so-called *reference model* (see Section 2.3.2). Comparing the reference model with the actual measurements is one approach in anomaly detection. Thus it is crucial to use appropriate metrics for reasoning about the performance of a system [Jain 1991, p.4] and performing anomaly detection.

2.1.1 Response Time and Processing Time

The term *response time* applies to the time period between user interaction and the system presenting a result [Shneiderman 1984, p. 267]. Possible examples are pressing a button on an *user interface (UI)* or an *application programming interface (API)* call [Fowler 2003, p. 7]. Figure 2.1 illustrates high-level steps in the interaction with web application systems. For a user working with a *browser*, this example shows the click on a link and the subsequent rendering of a web page. This response was generated by the web application and is encoded in *Hypertext Markup Language (HTML)*, as drafted by Hickson [2012]. We refer to the time a systems needs to generate the response as *processing time*.

The term *server* is used in context of dedicated hardware or software that responds to service requests [Fielding et al. 1999, p. 9]. From the user's perspective, responses from large-scale software systems appear similar to those generated by single servers. Thus, the diagram in Figure 2.1 use the terms 'server' and 'system' synonymously. This example takes only the HTML output into account and leaves out images, *stylesheets* and other resources that are commonly used for rendering web pages in modern Web 2.0 applications. The request transmission time span ends when the system has read all incoming data. Correspondingly, the output time span begins when the system starts sending data to the client. The model stated here is derived from the more complex interaction model of Jain [1991, p. 37].

This interaction is shown from the *browser* in Figure 2.2. The example web page contained 3.78 KB and was requested via the *Hypertext Transfer Protocol*

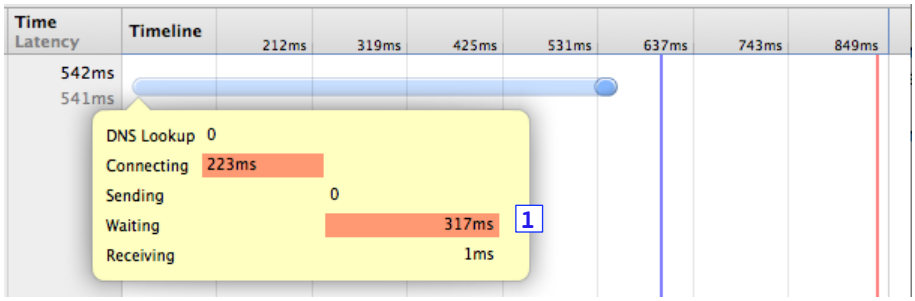

Figure 2.2: Request from the browser's view point. Shown is a screenshot of the Chrome Browser version 16 requesting the web page `http://forkosh.com/pstex/latexcommands.htm`. The *waiting time* at 1 is equivalent to the *processing time* in 2.1.

GET method [Fielding et al. 1999, p. 53]. The 'waiting' time is the time that the system needs to process the request. Transmission time spans are low in comparison to the network and protocol overhead summed up under the 'Connecting' label.

2.1.2 Metrics for Web Application Systems

Systems offering software that is used by an Internet browser are commonly known as *web applications*. The response time includes all steps necessary for the user to continue working, but also involves the network latency and the browser's rendering time. How long this time may be until the user gets disappointed or distracted by side tasks is a matter of ongoing research without simple answer [Shneiderman 1984, p. 274] (However, studies have shown that users abandon web sites responding slower than seven to eight seconds [Seow 2008, p. 58]).

Modern Web 2.0 technologies try to address latency problems by offering background processing. For instance, *Asynchronous JavaScript and XML (Ajax)* [Holdener 2008] can be used to send computation-heavy processes to the server while letting the user continue browsing the page [van Kesteren 2012, Section 4.7]. Other means are caching or offline data storage, which is a novel approach included in the *HTML5* standard [Hickson 2012, Section 5.6.2].

However, the response time is influenced by the uncertainty of network latency and browser rendering time. Measuring the processing time only, takes the observations to the system domain. Since all latencies between the incoming request and outgoing response are influenced by the system and its architecture, this metric is easier to use and promises more accurate results.

2.1.3 Availability and Reliability

Sommerville [2007, p. 51] defines the availability of software as the successful functioning in time point t. In other words, a system is available at t if it has not failed till that point.

The IEEE [1990, p. 32] provides the underlying terminology: *failure* are moments when the system cannot deliver the desired results due to a *fault* in a software's state. Faults are incorrect steps in the computer processes or data that can be caused by *error*. Examples for faults are incorrect process instructions, *bugs* introduced by programmers or just erroneous human interaction with the system.

The earlier described availability is primarily used for repairable systems. It can be calculated based on the system's *up time* and *down time* indicating the states of successful and unsuccessful functioning. The formula given by Pham [2000, p. 32] is as follows:

$$\text{Availability} = \frac{\text{System up time}}{\text{System up time} + \text{System down time}} \quad . \tag{2.1}$$

Reliability is given when a system does not fail in the entire time of a mission according to Pham [2000, p. 32]. For centrally used and administered web applications, this measure is less relevant since they are normally repairable. In case a request failed, users wait and retry by hitting the *reload button* on the web *browser*. After transitioning out of a faulty state, the web application will continue to service the request successfully.

2.2 Time Series Analysis

Every measure in a system's performance is influenced by the current usage as well as the preceding behavior. Box and Jenkins [1990, Preface] hence define *time series analysis* as techniques to analyze dependent observations.

2.2.1 Basic Definitions

A *time series* X is a discrete function that represents real-valued measurements $x_i \in \mathbb{R}$ for every time point t_i in an equally spaced set of n time points $t = t_1, t_2, ..., t_n$ as described by Mitsa [2009, p. 22]:

$$X = \{x_1, x_2, ..., x_n\} \quad .$$

The *duration* of a time series is defined by the distance between the first and the last point: $D_X = t_n - t_1$. Further on, we define the distance between every pair of subsequent time points as the *step size*. For a time series X it is denoted as Δ_X. It equals to the quotient of window length and the number of time points as follows:

$$\Delta_X = t_{i+1} - t_i = \frac{D_x}{n} \quad .$$

For every t_i, the time series has a corresponding value of $x_i \in \mathbb{R}^d$ with d being the number of dimensions [Mitsa 2009, p. 47]. If $d = 1$ the time series is called *univariate* and *multivariate* if $d > 1$. In this document the set of *univariate* time series is used and denoted $\mathbb{TS}$.

A time series $W = \{w_1, ..., w_m\}$ can be a *window* of time series $X = \{x_1, ..., x_n\}$. For this window $W \subseteq X$ the following assertions are:

1. Both step sizes are equal: $\Delta_X = \Delta_W$.
2. The values of the window have to correspond to the values in the longer time series: $\exists\, h \in \mathbb{N},\ \forall i \in 1, .., m : x_{h+i} = w_i$.

One of the statistical measures that define time series is the *mean* value defined as follows:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \quad . \tag{2.2}$$

Figure 2.3 shows the basic properties of time series and the mean calculated over the whole series in an example. This will be used throughout this chapter to explain the different means of forecasting and anomaly detection.

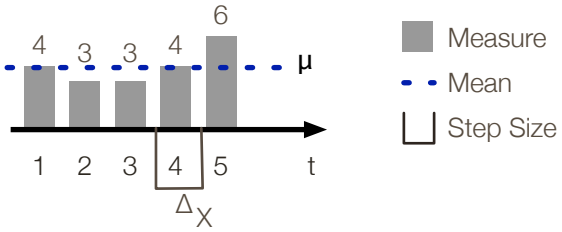

Figure 2.3: Time series with mean $\mu = 4$ and step size $\Delta_X = 1$

Time series are characterized by certain *features*. Within one of the most significant features stated by Wang et al. [2008, p. 4] is the *seasonality* which defines the length of a period of recurring patterns of the mean value. *Trend* is a feature to define the long-term change of the mean in a time series. In combination, trend and seasonality can be used to model the course of a time series, as depicted in Figure 2.4.

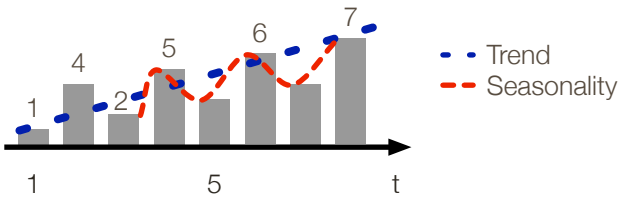

Figure 2.4: The features *trend* and *seasonality* can be used to describe a time series

2.2.2 Temporal Data Discretization

All forecasting algorithms used in this context take time series as input. As defined in Section 2.2.1 subsequent data points are equidistant. Although measurements of performance can be gathered discretely, they are not obliged to occur on these defined time points.

A series of measurements with same measurand $\mathcal{M}$ is a *temporal sequence* as defined by OMG [2012, p. 2] (see Section 2.1). In consequence, time series are

temporal sequences with equidistant measurements. We define a sequence of n raw measurements $\{r_1, ..., r_n\}$ between two time points t_{start} and t_{end} as follows:

$$ES_{t_{start},t_{end},\mathcal{M}} = \{r_i = (t_i, \mathcal{M}, m_i) \mid 0 \leq i < n,\ t_{start} \leq t_i < t_{end}\}. \tag{2.3}$$

This stream of raw measurements has to be *preprocessed* in order to yield time series usable for analysis. This preprocessing is also called *discretization*. Figure 2.5 shows how a continuous, temporal sequence get discretized with function f into time series X.

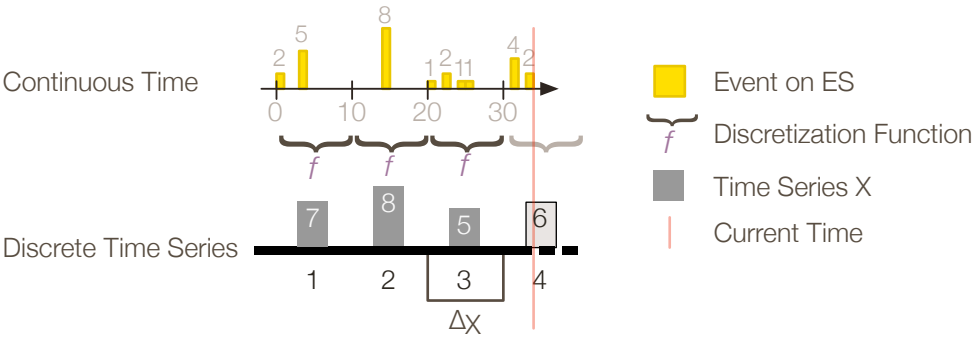

Figure 2.5: A univariate time series $X = \{x_1, x_2, x_3\}$ for the measurand `total_time` with $\Delta_X = 10$ is constructed from a temporal sequence by the function f. The input data of this process is a temporal sequence $ES_{0,34,\texttt{total\_time}}$, which is also called "basic window" by Shasha and Zhu [2004, p. 107]. The red line shows the current time as 34 and indicates a possibly ongoing preprocessing in future.

Every discrete point x_i of time t_i in the extracted univariate time series X yields a real value calculated by a preprocessing function $f : \mathbb{ES} \rightarrow \mathbb{R}$. There are many preprocessing functions possible, such as the earlier described *mean* or trivial *maximum* or *minimum* functions. When it is required to construct absolutely comparable values the *sum* (or *aggregation* function in this context) is appropriate:

$$f(ES) = \sum_{r_i=(t_i,\mathcal{M},m_i)\ \in\ ES} m_i \quad . \tag{2.4}$$

2.2.3 Time Series Forecasting

Forecasting algorithms are applied to time series in order to calculate the values that are most likely to occur next. The calculation is based on past data only and does not include anticipated values or suggestions from outside.

Let $W = \{w_1, ..., w_m\}$ be a time series with step size Δ_W and length $D_W = m$. W is a so-called *sliding window* (as used by Shasha and Zhu [2004, p. 107] and the later introduced software Esper), that always resides at the end of another time series.

W is a window of X with the same step size $\Delta_W = \Delta_X$ but with a smaller or equal length $D_W \leq D_X$. A forecasting algorithm $\lambda_{fc} \in \Lambda$ can be applied on W which produces the output time series $F = \{f_1, ..., f_l\}$. Its length, $D_F = l$, is called *lead time*. This parameter is described by Box and Jenkins [1990, p. 1] and is passed to λ_{fc} as following:

$$F = \lambda_{fc}(W, l) \quad . \tag{2.5}$$

The remainder of this section lists several forecasting algorithms in the following paragraphs. They were chosen by variation and simplicity and got explained in detail and compared in the bachelor's thesis of Frotscher [2011]. All algorithms are available as packages of the '$\mathcal{R}$ Project for Statistical Computing' as stated later in Section 2.4.3.

Moving Average

Let $W = \{w_1, ..., w_m\}$ be a sliding window of an underlying time series X. Mitsa [2009] state that the window size D_W has to be chosen carefully in order to track the original series closely and discard small unimportant outliers at the same time.

Figure 2.6 shows the *mean forecaster* that uses a sliding window. The statistical mean is applied to predict the next value.

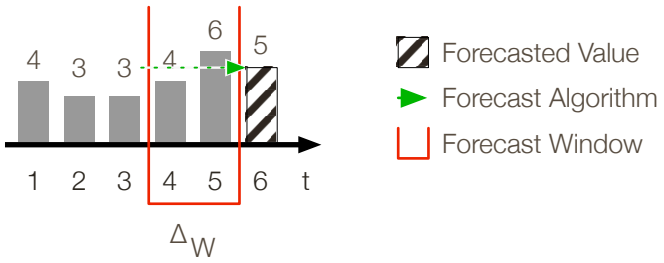

Figure 2.6: Forecasting by the statistical mean of a sliding window with size $D_W = 2$. The forecast at time point t_6 is marked with a shaded bar in this example.

In $\mathcal{R}$ the `filter` method can be applied on the whole time series:

```
1  series <- c(4,3,3,4,6)
2  filter(series, sides=1, rep(1,2)/2)
```

Listing 2.1: Calling the filter method in $\mathcal{R}$. The variable `series` holds values corresponding to the sample data depicted in Figure 2.6.

Listing 2.1 gives the parameter `sides=1` indicating the result on the right side of the window. The output time series is `NA 3.5 3.0 3.5 5.0`, with an average of $t_5 = 4.0$ corresponding to the forecast shown in Figure 2.6.

ARIMA

For time series without stability over a period of time, called *nonstationary*, Box and Jenkins [1990, p. 85] developed *ARIMA* models. These forecasting algorithms depend on the selected model using three as steps in the processing:

- **AR** *Autoregression*: Performing the forecast with a regression operator (see [Box and Jenkins 1990, p. 88]), which weighs time points according to their recency, as depicted Figure 2.7.
- **I** *Integration*: An optional summation in the MA process.
- **MA** *Moving Average*: Building the average over a sliding window as described in Figure 2.6. This step is used to "smooth out short-term fluctuations" [Mitsa 2009, p. 24].

For *nonstationary* time series, ARIMA is a solution that requires the selection of a transformation model. For $\mathcal{R}$ projects, the CRAN lists a package called `forecast`, which offers a configurable set of ARIMA forecasting algorithms [Hyndman and Khandakar 2008, p. 8]. This package for forecasting in $\mathcal{R}$ includes selectable ARIMA models based on a given differentiation parameter (`order`). These parameters denote the components as $(p,\ d,\ q)$, with p being the order of the AR process, d whether integrated or not, and q the order of the MA process. Listing 2.2 gives an example.

```
1  Arima(c(4,3,3,4,6), c(1,0,1))
2  # output:
3  Series: c(4, 3, 3, 4, 6)
4  ARIMA(1,0,1) with non-zero mean
```

Listing 2.2: The ARIMA method with given $(1,0,1)$ order in $\mathcal{R}$

Hyndman and Khandakar [2008], the authors of the $\mathcal{R}$ `forecast` package, state the necessity of model selection by "a common obstacle". Thus, an automatic selection is possible. An example of the according method call of the ARIMA function in $\mathcal{R}$ is given in Listing 2.3 showing the selection of the `Arima000` forecaster.

```
1  auto.arima(c(4,3,3,4,6))
2  # output:
3  auto.arima(c(4,3,3,4,6))
4  Series: c(4, 3, 3, 4, 6)
5  ARIMA(0,0,0) with non-zero mean
```

Listing 2.3: On the example time series, the `forecast` package of $\mathcal{R}$ automatically chooses the ARIMA000 process.

Further reading of the underlying mathematics can be taken from Box and Jenkins [1990, p. 85], a practial comparison amongst other algorithms was made by Frotscher [2011].

SES Autoregression

A special case of ARIMA [Box and Jenkins 1990, p. 93] configured with $(p,\ d,\ q) = (0,1,1)$ is *single exponential smoothing (SES)*. This model assumes that recent observations are more relevant. Thus, it weighs the observations based on their distance to the latest time point. Equation 2.6 shows the formula by Mitsa [2009]. Let $X = \{x_1, ..., x_n\}$ be a time series and $0 < \alpha < 1$ the *smoothing constant*. The next value f_{n+1} of the forecasted time series $F = \{f_{n+1}, ..., f_{n+l}\}$ is hence defined as follows:

$$f_{n+1} = \alpha x_t + \alpha(1-\alpha)^2 x_{t-1} + \alpha(1-\alpha)^3 x_{t-2} + ... \quad . \tag{2.6}$$

Figure 2.7 exemplifies the above mentioned smoothing constant by showing increasing circles towards the latest measurement. With given values of $X = \{3,3,4,6\}$ and $\alpha = 0.9$ the resulting $f_5 = 5.7897$. $\mathcal{R}$ uses Hyndman's implementation [Makridakis et al. 1978] as demonstrated in Listing 2.4 with `h=1` being the number of forecasting steps.

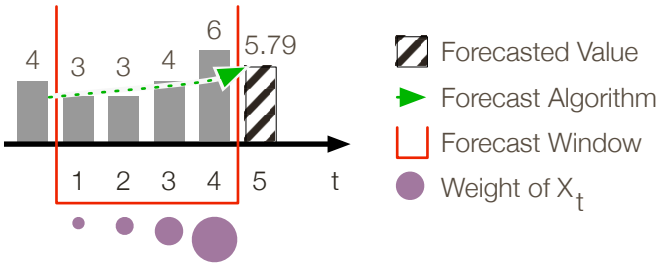

Figure 2.7: Single Exponential Smoothing (SES). The circles indicate an increasing weight for the forecasted value in t_5.

```
1  forecast(ets(c(3,3,4,6), alpha=0.9), h=1)
```

Listing 2.4: SES forecast method in $\mathcal{R}$

Forecast by Observation

This method takes the measurement of the beginning of the forecasting window of the input time series as shown in Figure 2.8. The next predicted value is hence $f_{n+1} = x_{n-D_W}$.

If the input time series has a strong trend, there has to be a preceding step of detrending that eliminates this deviation. Another, more practical approach, is to take the mean or median of the last known pattern.

Oftentimes, this forecasting method relies on detecting the seasonality first. This requires certain domain knowledge, e.g. holidays or usage patterns of a system.

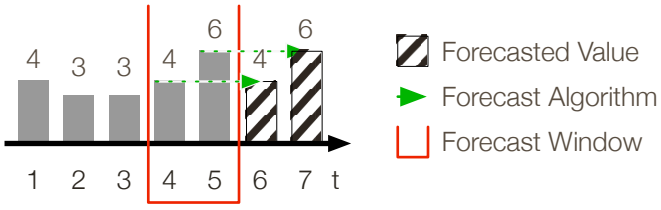

Figure 2.8: Forecasted steps based on the last observations in D_W.

2.3 Anomaly Detection

Anomaly detection is the research on data in order to find behavior that does not appear normal [Chandola et al. 2009, p, 1]. The preceding step is finding behavior or data that is accepted to be normal. Abnormal data can be distinguished by creating a reference model as described in Section 2.3.2. Based on this reference, the distance can be a measure of abnormality. If the distance exceeds a certain point, the test data is declared as abnormal.

This section describes the basic definitions in the following section, the creation of a reference model and finally Sections 2.3.3 and 2.3.4 introduce the *anomaly score* concept that defines this *level* of abnormality.

2.3.1 Taxonomy

Anomaly detection can be classified in respect to the anomaly types and the methods, which are used. Figure 2.9 shows the hierarchy used by Banerjee et al. [2008, p. 27] to classify approaches in this field.

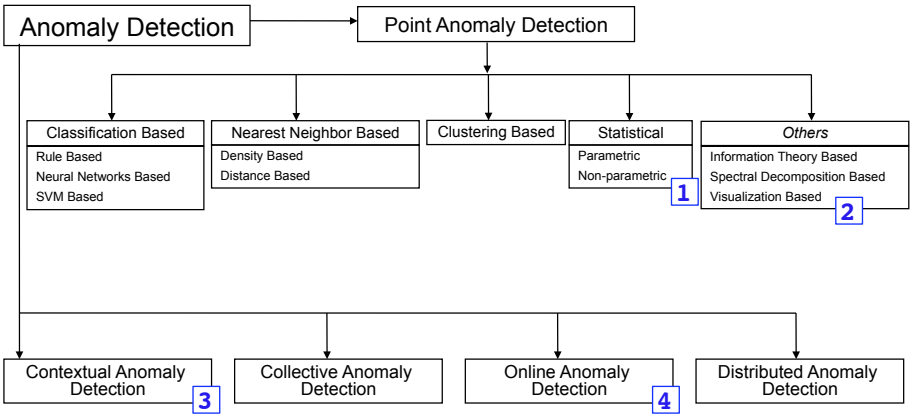

Figure 2.9: The taxonomy classifies the approaches in the problem field "Anomaly Detection" and the relations amongst them. This hierarchy from Banerjee et al. [2008, p. 27] is based on a related technical report [Chandola et al. 2007, p. 7]. 1, 2, 3, and 4 show the fields addressed by ΘPAD.

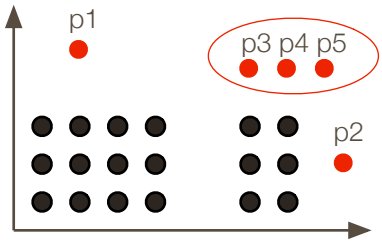

Figure 2.10: A two-dimensional data set with contextual anomalies based on [Chandola et al. 2009, p. 7]. $p1$ and $p2$ are collective anomalies $p3$-$p5$. $p4$ appears normal in the context of $p3$ and $p5$.

2.3.2 Basic Definitions

Anomalies are patterns in data that do not conform to expected behavior. The *detection* of anomalies refers to the problem of finding these patterns [Chandola et al. 2009, p. 1].

Outliers, i.e., points or sets of anomalous measures can either deviate from the context, described as *contextual* anomalies by Chandola et al. [2009, p. 7]. Or they are classified as *collective anomalies*, not conforming to the whole data set. In the 2-dimensional marker chart shown in Figure 2.10 the points p_1 and p_2 are contextual outliers deviating from the other sets shown as black dots while the set $p_{3..4}$ is a collective anomaly deviating from the whole data set.

Reference Model

Anomaly Detection relies on comparing a measured behavior with a reference model which describes the normal behavior of a system (see Yao et al. [2010, p. 3]). For instance, intrusion detection systems have reference models defined by attack-free training sets [Chan et al. 2003, p.].

Apart from employing the right metrics as a data basis, a big challenge is to find the right behavior of this model, which can be described as being normal. As stated in Section 1.1, anomaly detection is hard to be solved generically. This is mainly due to the fact that every system induces own parameters and behavior that define the reference model.

Every reference model is based on a chosen metric. In this context, this is performance, for instance measured by the response time or the count of concurrent requests of a system. Ongoing from that, the normal behavior has to be analyzed, so that patterns can be found. Two typical examples of SaaS applications are:

- The concurrent user count shows a *seasonality* pattern repeating every day due to the business hours. Especially for business applications, weekends could additionally induce some differences in the usage.
- The response time, as well as the usage, should not differ much from the last measured point. Bad performance would show a rapid increase in the

response time. Thus, one approach for the reference model is assuming that the last measured performance is a good reference for the next values. This approach deals with contextual anomalies as classified by the taxonomy in Figure 2.9. Another benefit of this simple model is the automatic adjustment: Every time a new measurement gets appended, the reference model is updated to fit the real system behavior.

Anomalies are deviations from these reference models. The following Section 2.3.3 introduces distance measures to quantify the deviations.

2.3.3 Metrics

Since anomalies are deviations from the reference model, it can be determined how big the deviation is, respectively how 'far away' a non-conforming point is. Distance measures can be used with any d-dimensional points $x = x_1, ..., x_d$ and $y = y_1, ..., y_d$ in a data set with points in $\mathbb{R}^d$. For time series with equal length, the *euclidean distance* is defined as follows:

$$D_{\text{eukl}}(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_d - y_d)^2} \tag{2.7}$$

$$= \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \quad . \tag{2.8}$$

Assuming that y is a reference point (see Section 2.3.2) that is perceived of being normal at time point t_i. With a distance measure D the so-called *anomaly score* can be normalized determined for every x_i as follows

$$A(x_i, y_i) = \left| \frac{D(x_i, y_i)}{x_i + y_i} \right| \quad . \tag{2.9}$$

2.3.4 Anomalies in Univariate Time Series

As defined in Section 2.3.2 contextual and collective anomalies can be distinguished. The same can be applied for univariate time series. Figure 2.11 shows this distinction in the example we use throughout this chapter.

Anomaly Score Calculation

In order to apply anomaly metrics on data points in a univariate time series, the reference model of the data has to be determined first. Since univariate time series have only one variable changing over time, the reference model can be defined as a univariate time series as well.

Detecting anomalies can be broken down to comparing two univariate time series against each other. This can be achieved by applying the euclidean distance

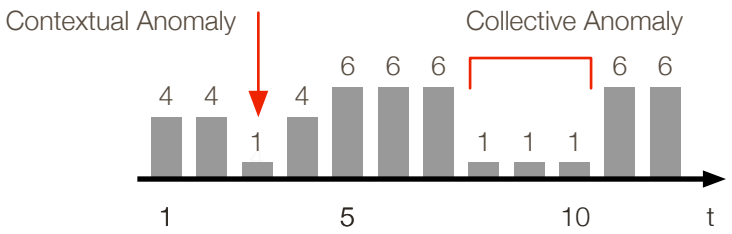

Figure 2.11: In the time series X t_3 and $[t_8, ..., t_{10}]$ exemplify the two types of anomalies. t_9 is not a contextual anomaly since its neighbors have the same value.

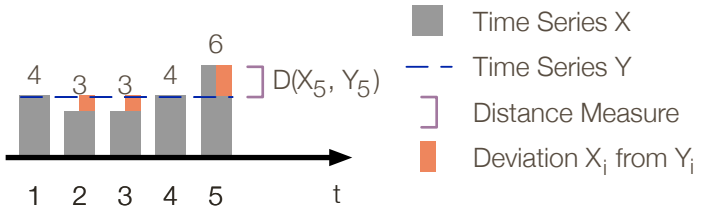

Figure 2.12: Time Series X and $Y = \{4,4,4,4,4\}$ are compared in every point with distance measure D. The line chart above the bars is the normalized distance as anomaly score.

(Equation 2.8) to every corresponding two real values in these time series. Let $X = \{x_1, x_2, ..., x_k\}$ and $Y = \{y_1, y_2, ..., y_k\}$ be two time series of same length k. Since values in univariate time series are in $\mathbb{R}$ the dimensionality d (see equation 2.7) is simply 1.

$$D_i(x_i, y_i) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2} = \sqrt{(x_i - y_i)^2} \tag{2.10}$$

$$= |x_i - y_i| \tag{2.11}$$

Figure 2.12 shows the absolute distance of every time point of two time series denoted as the colored bars. The graphs in this thesis follow the *SUCCESS* principles by Hichert [2011], defining several best practices for data visualization.

The previously defined distance between two or more time series is not absolute. In order to compare multiple deviations, this distance can be normalized. Let $AD : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a normalization function, X the base time series and Y the time series of forecasts as used throughout this chapter. The resulting time series of anomaly scores $\Psi = \{\psi_1, ..., \psi_n\}$ with $\Delta_\Psi = \Delta_X$ is hence defined as follows:

$$\Psi_i = AD(x_i, y_i) \quad . \tag{2.12}$$

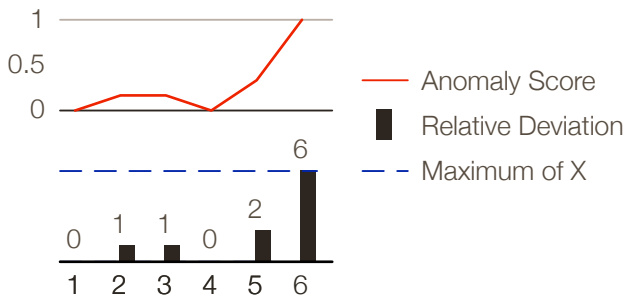

Figure 2.13: The Anomaly Score calculated with function AD between two univariate time series X and Y with an assumed maximum of $max(X) = 6$. At time point t_6 the distance reaches its maximum and causes the anomlay score to increase to 1.

As an example, we define $AD : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ as the relative distance to a fixed time series maximum in Equation 2.13. The resulting time series yields values from 0 to 1 and is depicted in Figure 2.13:

$$AD \quad = \quad \frac{D(x_i, y_i)}{max(X)} \quad . \tag{2.13}$$

Classification and Alerting

Chan et al. [2003, p. 2] suggest that anomaly detectors try to describe the anomaly as precisely as possible. A known approach is the previously defined anomaly score as a description of how abnormal the current observation is. In order to determine whether this behavior can be classified as an anomaly, a human has to interpret it with knowledge of the attached measurand and the reference model.

When the reference model is known, a *threshold* can be used that is dependent on the measure. If the anomaly score exceeds a certain value, the measurement is classified as abnormal. The setting of the threshold is attached to the choice of the reference model and accordingly the forecasting algorithm done by humans. As hinted by Oliner et al. [2012, p. 4], setting the threshold is difficult with respect of finding the right balance of *true positivess* and false alarms.

Figure 2.14 shows the detection of anomalies based on the anomaly score time series $A = \{a_1, ..., a_n\}$ and a given threshold θ. For every $\theta > a_i$ the behavior is classified as abnormal. Hence, for every threshold a particular number of true positives and false negatives can be generated according to Figure 2.15.

In the classification model of Salfner et al. [2010, p. 8], a detection is also called ‘positive’, whereas no detection accordingly is defined as ‘negative’. If it is known if the behavior is actually normal or abnormal, these detection states can be proven. Hence, there are true and false *positives* and true and false *negatives* as shown in Figure 2.15.

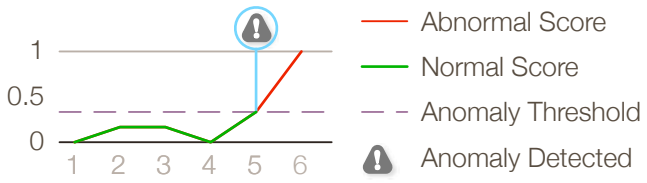

Figure 2.14: The anomaly score time series A is compared with threshold $\theta = 0.4$. When the score exceeds the threshold for the first time in $t = 5$, an anomaly is detected.

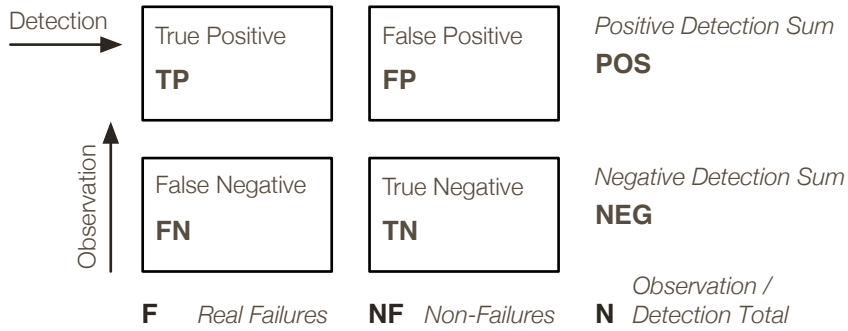

Figure 2.15: Grid of detection cases following the model of Salfner et al. [2010, p. 8]. TP and TN sum up to the *real failures* F, the number of non-failures (NF) and the total of all decisions N.

Detection Performance Comparison

As stated in the previous section, an anomaly detection algorithm produces true and false positives in the detection process. Compared with the real world, these results can be accumulated to form measures, which gives clues on the quality of the algorithm.

Henceforth we use two important metrics according to Salfner et al. [2010, p. 8]: The *True Positive Rate (TPR)* and the *FPR*, defined as follows.

True Positive Rate, the sensitivity of the algorithm. This number does not necessarily lead to a better algorithm since it can still produce a high number of false positives.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = \frac{\mathrm{TP}}{\mathrm{F}} \tag{2.14}$$

False Positive Rate, The ratio of anomalies the algorithms detected incorrectly.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = \frac{\mathrm{FP}}{\mathrm{NF}} \tag{2.15}$$

For every evaluation of an anomaly detection algorithm or a particular configuration of a detector, the FPR and the TPR can be calculated. As of [Maxion and Roberts 2004, p. 2], *receiver operating characteristic (ROC)* curves display these metrics in a two-dimensional chart. Figure 2.16 shows the resulting model.

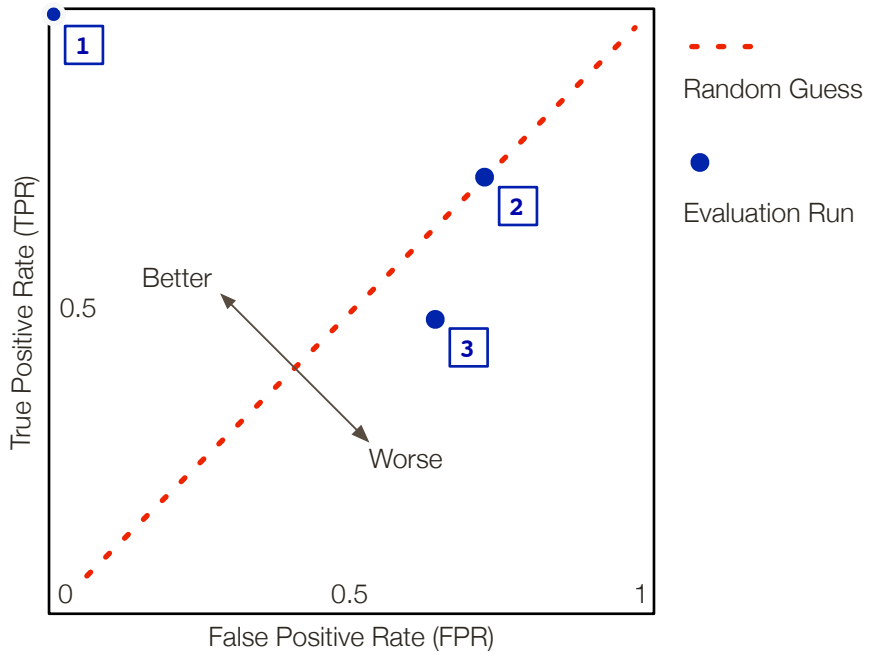

Figure 2.16: The model of the Receiver Operating Characteristic (ROC) curve compares algorithms and visualizes the tradeoff between the detection rates of false and true positives. It helps making the tradeoff between how many false alarms shall be accepted, so that a certain detection rate is attained.

One example execution of algorithm 1 never detected false positives. 2 resides on the dashed line indicating a random detection. 3 is misleading since it produced more false positives than true positives.

Every point in the chart is one instance of a detection algorithm. Better algorithms cumulate above the diagonal axis.

Bechtold and Heinlein [2004, p. 44] confirms the important of ROC curves since false negatives are especially bad for detection algorithms, especially in the intrusion detection field. In ROC curves, TPR and TPR are used together. This can also be expressed in the following two metrics.

Precision, indicating how many anomalies got detected out of all actual observed anomalies:

$$\text{PREC} = \frac{\text{TP}}{\text{POS}} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad . \tag{2.16}$$

Accuracy, indicating the ratio of correct detections to all observations. In ROC curves, algorithms with high accuracy reside at the top of the chart:

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{N}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \quad . \tag{2.17}$$

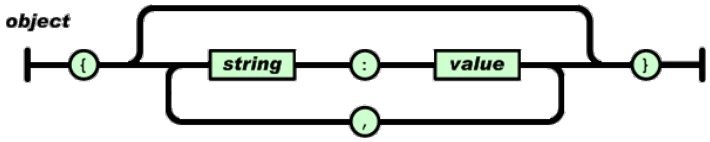

Figure 2.17: Object definition in JSON from `http://www.json.org`

In Figure 2.16, the perfect algorithm at point `E1` never detects incorrectly, thus $\text{FPR} = \frac{\text{FP}}{\text{FP}+\text{TN}} = 0$ and it's accuracy is $\text{ACC} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{FN}+\text{TN}} = \frac{\text{TP}+\text{TN}}{\text{TP}+0+0+\text{TN}} = 1$.

2.4 Technology Stack

This section introduces the main protocols and technologies used for modern *web applications*. Without concrete definition, this term is used in the HTML standard since version 5 by Hickson [2012, abstract]. This document aims to standardize modern web technologies like offline storage and enhanced browser APIs (Application Programming Interface) and is still heavily influenced by big software vendors.

The standard protocol for transporting HTML content is Hypertext Transfer Protocol. The underlying layer usually is *TCP/IP* with the standard port being 80 [Fielding et al. 1999, p. 13]. For the following formats all use these bases as underlying layers of transportation or transmission.

2.4.1 Protocols, Formats, and Concepts

Some new data storage and transmission concepts that came up in the recent years rely on new data formats that are both human-readable and processable by machines, such as Javascript Object Notation (JSON). The following sections define these formats and protocols so that the technical characteristics of Θ**PAD** as well as the case study environment can be interpreted.

JSON and BSON

Binary JSON (BSON) is a standard which defines a binary encoding for serialized objects. The serialization relies on the JSON, which is, amongst other implementations, interpretable by Javascript parsers through the `eval()` method.

Apart from the compatibility to Javascript[1], JSON has a small syntactical overhead but does not offer sophisticated schema definitions or transformation standards. These attributes made it popular for use cases involving web browsers and high performance environments as found in modern web applications.

Figure 2.17 shows the definition of an arbitrary JSON object which can be nested as `value` inside other objects [IETF 2006, p. 7]. JSON's syntax is character-

[1] Which, amongst others, helped defining the ECMA script standard [Fulman and Wilmer 1999, p. 2]

```
timeseries:
  deltat: 2000
  start: 1329552143   # Comment: This is Sa 18 Feb 2012 09:02:23 CET
  values: [3,3,5,5,4]
  nextprediction: 3.5
```

Figure 2.18: The time series example in YAML syntax

based and thus human-readable [Holdener 2008, p. 92]. The format is schema-less and therefore any JSON encoded data is freely extensible. An example of a JSON formatted object is shown in Figure 2.6.

YAML

The YAML Ain't Markup Language (YAML) markup was designed with the assumption that any data structure can be broken down to scalars and lists. The latter can be either ordered or associative. The first specification was in 2001 and described it as a 'Minimal XML language'. Since then, YAML evolved into a subset to JSON as the latest draft by Ben-Kiki et al. [2009, Status] explains.

As the code in Figure 2.18 demonstrates, the YAML markup is designed to be human-readable [Ben-Kiki et al. 2009, Section 10.3.]. For machines however, it takes more effort to generate and parse. Thus, it is often used for configuration files[2] that have to be written by humans and processed by machines on system startup time.

AMQP

The Advanced Messaging Protocol (AMQP) is an open protocol standard defining message-queuing communications. It defines *message producers* sending messages of arbitrary formats to *brokers* which then asynchronously provide clients with the messages they subscribed to. Brokers themselves employ *message queues* and *exchanges* that define how the message get routed from producers to clients.

Figure 2.19 uses the AMQP notation which is used throughout this thesis for message queueing. Depicted is the message flow through these different actors:

- **Producers** send messages to previously defined exchanges.
- **Exchanges** route messages to zero or multiple queues depending on the type of the exchange and the configured *bindings*.
- **Queues** hold the messages and forward them to subscribed clients in a *First In First Out (FIFO)* manner.
- **Consumers** subscribe to message queues and receive messages whenever possible.
- Consumers and Producers are called **Clients** that can define exchanges, queues and bindings according to their privileges.

[2] A search for `config` and YAML on github shows $157,696$ results on February 18, 2012: `https://github.com/search?language=YAML&q=config`

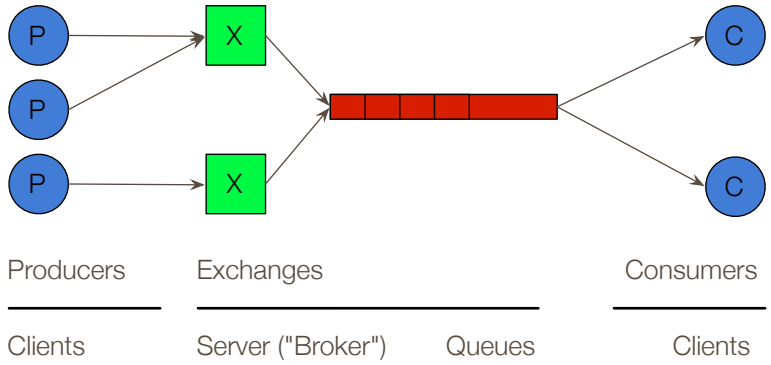

Figure 2.19: Actors defined by AMQP

One common pattern, proposed by Frank Schmuck [Birman 2010, p. 9] is called *Pub/Sub*: Producers *publish* messages onto a queuing server and *subscribed* consumers only get the messages if they declare a demand. Messages are discarded if there is no consumer subscribed.

AMQP is vendor-neutral, i.e., it can be implemented by any queuing software confirming to the specified standard and that the interoperability across multiple software vendors is encouraged. Implementations are StormMQ, Apache Qpid and RabbitMQ, which is currently used by XING.

AMQP works on top of the *network layer*, i.e., the *TCP/IP* protocol. Thus, AMQP conforming queues define one or more endpoints that are addressable via IP addresses. Through the queues binary data can be sent which makes it open to carry any data format.

Document Store Databases

This subgroup of NoSQL databases is a 'relatively new breed' of databases that has no concept of tables, SQL or rows [Membrey et al. 2010, p. 3]. Data is stored in entities called *documents* that hold encoded data. The storage usually relies on standard formats such as the previously described YAML, JSON or even binary formats that are colloquially considered as documents such as PDF. Since documents are schema-less, they can be altered and extended without the need to migrate existing data.

Like Key/Value data stores, which provide high availability due to replication [DeCandia et al. 2007, p. 205], documents are accessible via unique identifiers, so-called *indexes*. The distinct feature of document store databases is the possibility to query the stored values, in this case: documents. To achieve that functionality, this type of database has to interpret the stored documents Weber [2010, Section 4.1].

With the increasing number of implementations and approaches, this type of database gained popularity the last years. A comparison lists more than eleven

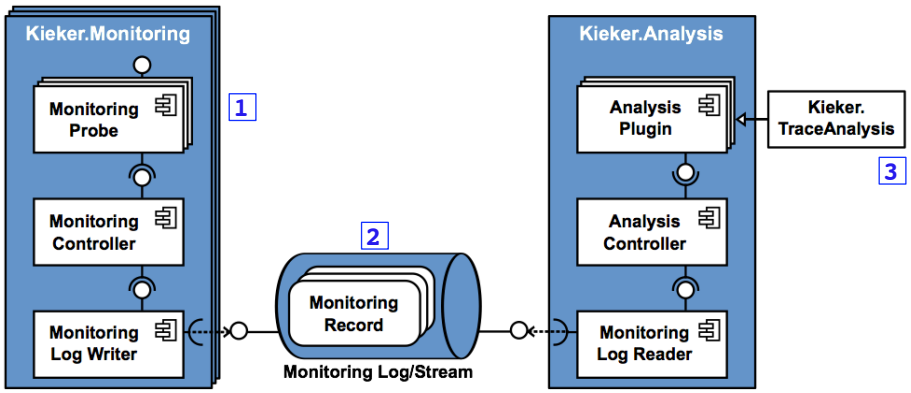

Figure 2.20: The Kieker architecture is designed with layers that deal with different levels of complexity. Measurements are gathered by *probes* at application level and are passed down to the monitoring stream at 2. This lowest layer can be configured to use a variety of technologies. At the analysis side (3), plugins can be loaded and executed [Ehmke et al. 2011, p. 3].

implementations for different formats and use cases.[3] Many of them are production-ready and under an open source license.

2.4.2 Kieker Monitoring Framework

Kieker is an extensible framework for continuous monitoring of distributed software systems [van Hoorn et al. 2012]. Its open source code base is maintained and enhanced by the Software Engineering Group of the University of Kiel.[4] It supports injecting so-called *probes* into the monitored system to analyze and visualize architectural characteristics regarding structure and behavior. This instrumentation can be done either manually or in an *Aspect-Oriented Programming (AOP)* fashion. The configurability of Kieker allows offline analysis as well as the use in online production systems. For the online monitoring, it is designed to induce only a small overhead into the system under monitoring.

Additionally, its plugin architecture allows the usage of different analysis plugins as shown in Figure 2.20. In general, a plugin architecture allows implementations for different use cases to be configured and run centrally [Fowler 2003, p. 500].

The benefit of using Kieker in production systems, is the possible capacity planning, which makes it especially interesting for SaaS. It is tested, run in industry, and proven to be stable. Since it is open to measure any kind of metric implemented in the class `Monitoring Probe` (Figure 2.20), it is also imaginable to gather performance attributes that correlate with faulty behavior.

[3] `http://nosql-database.org`
[4] `http://www.se.informatik.uni-kiel.de`

```
1  series <- ets{c(3,4,5,7), alpha=0.99)
2  fc <- forecast(ts)
```

Listing 2.5: $\mathcal{R}$ forecast method

2.4.3 $\mathcal{R}$ Project for Statistical Computing

In 1992 normal programming languages were used to perform statistical computing. For businesses, computing statistical data normally involved programmers writing custom software. Bell Labs[5] provided a solution called `S`, which was more accessible for statisticians. However, that software was proprietary and expensive.

$\mathcal{R}$ evolved as a project of the University Auckland and is a runtime environment with its own `S`-like syntax. It is part of GNU, licensed open source under *GPL*, and continuously is improved by a large community that contributed many software packages.

These software packages are collectively held in a register called CRAN. One popular contribution, the `forecast` package for *time series analysis* is introduced in the following section.

The $\mathcal{R}$ forecast package

Business often need forecasts for large data sets as Hyndman et al. [2012, p. 1] state. For this broad use they created the `forecast` package available in the CRAN. Amongst an abundance of forecasting algorithms for *univariate time series analysis* it holds 90 data sets which can be used to test algorithms.

This `forecast` called package includes varieties of exponential smoothing and ARIMA models that were described previously. For ARIMA they also included automatic model selection [Hyndman et al. 2012, p. 8] supporting the difficult selection process.

The interface of forecasting follows the delegation pattern that encapsulates different means in the following function call on a `timeseries` object and selects the parameters according to the input data. Listing 2.5 shows the call on an univariate input series of type `ets` with the `alpha` parameter given as documented in Hyndman et al. [2012, p. 20].

All forecasting methods introduced in Section 2.2.3 are included in this package and can support the implementation of anomaly detection. Further benefits can be seen in the automatic model selection.

2.4.4 Case-Study Environment

XING's platform, `xing.com` with its over 11.7 million registered users evolved over a period of nine years [XING AG 2011, p. 3]. There are hundreds of servers in-

[5] `http://www.alcatel-lucent.com/wps/portal/BellLabs`

volved, which handle the high traffic this user base produces. Apart from this traffic, the technological diversity amongst the internal services is a constant challenge.

Application servers use standard frameworks and libraries such as Ruby on Rails and intercommunicate via *Representational State Transfer (REST)*ful interfaces. Additionally, the architecture is separated and information is channelled through AMQP queues. An own logging software, Logjam, was built to give administrative control.

In the following sections, details on the technologies are given, which allow XING to innovate continuously and remain stable for a high volume of requests at the same time.

Ruby on Rails

Ruby on Rails is the prominent web development framework of the programming language Ruby. It provides all important layers of web applications including a template engine and a database mapper. Is is open source and has a large community that contributed many freely available software packages.

Internally, it employs the MVC pattern [Raymond 2007] that separates the view from the model and controller logic. This clean separation leads to better understandable and testable code. Amongst these advantages, Ruby on Rails serves as a standard, which helps learning from existing applications. The key point for businesses is the fast incorporation of new employees and the richness of compatible open source packages for this framework.

Architecture

Apart from using Ruby on Rails for application servers, XING still uses Perl for large parts of the code base and many database servers running MySQL. The architecture is spread over many servers running Debian Linux with individual tasks. The servers communicate internally over AMQP (asynchronously) and HTTP REST for synchronous calls. The following is a list of services provided internally:

- Application Servers in Ruby on Rails and Perl handling the user interaction.
- SQL servers storing about 11 million registered users [XING AG 2011, p. 3] and their attached network, messages, events and groups.
- A Neo4J graph database that caches the connections amongst users.
- Servers with RabbitMQ offering the AMQP queues.
- Background server for sending newsletters, cleaning tasks, statistics and billing
- Servers for the *Xing Web Services (XWS)*, to let mobile and external applications access the platform.
- Monitoring servers (see next section).

Internally, XING follows the *page controller* pattern [Fowler 2003, p. 333] that structures the Hypertext Transfer Protocol calls on different controllers, in this case called *pages*. Hence, every application only services the assigned pages. The results are cleaner debugging and separation of concerns [Starke and Hruschka 2011].

Monitoring Systems

XING basically uses five different software products to monitor its behavior:

1. Logjam: open source software written by Stefan Kaes to aggregate performance logs of every production server.
2. A server for detailed log inspection and filtering running Graylog2. This open source software is written by Lennard Koopmann and sponsored by XING.
3. Munin for IT infrastructure and resource monitoring.
4. Nagios for alerting system failures and down times.
5. Omniture to measure the platform's performance from different computers worldwide.

All servers in the architecture write log lines onto a message queue. A dedicated server aggregates them into several *importer* queues to reduce the data processing amount. For instance, application servers publish their logs into a queue which is identified by the page name. This identifier comes from XING's separation of the request to certain 'page' called controllers. From a mathematical perspective as defined in Section 2.2.2, every log line is a raw measurement in a temporal sequence. These entities carry measurements and are attributed with a measurand which is the page name from the origin application server.

Logjam

When XING's app grew from one Perl server to a sophisticated architecture distributed over several application and database servers, the logs produced by the Ruby on Rails and Perl web servers exceeded the size to be monitored per server individually.

To address this, the software architects channelled the log lines into AMQP queues that could be read and aggregated by one single server. This so-called importer consumes every log line and writes the aggregates of every minute into a MongoDB instance. With this approach, logs provided a level of granularity that was still good enough for debugging but also solved the problem of too much data being produced by the servers altogether.

Additionally, every server employs certain measures to monitor each instance's performance. Important measures are response time and counters for the count of certain calls such as database queries, memcache calls and template renderings. Furthermore, some system data like memory allocation and heap size is gathered and published to the queue. This information is encoded in JSON and also propagated to the importer. A sample message is given in Listing 2.6.

Logjam is the web front end written by Stefan Kaes and used by XING to displaying the log aggregates from the MongoDB historically and continuously every second from the importer queue. The screenshot in Figure 2.21 shows Logjam's web-based dashboard. The front end comes with a live view displaying the incoming messages from the queue directly and the possibility of browsing to older dates. The graphs are built with D3 [Bostock et al. 2011] and the server that produces the HTML output is a Ruby on Rails application.

```
{
    "count": 5204.903527993169,
    "memcache_time": 6505.196318140181,
    "api_time": 2207.0271495891297,
    "db_time": 5004.8727338680155,
  ...
    "view_time": 3936.1623304929153,
    "total_time": 1586.8188192888886,
    "api_calls": 5546.250545491678
}
```

Listing 2.6: AMQP message produced by the logjam importer

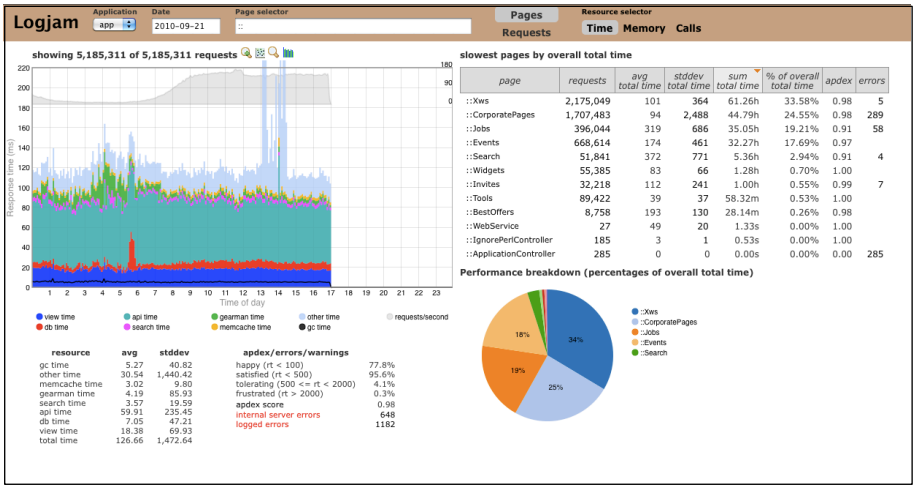

Figure 2.21: The Logjam web front end showing the measurements of one day till 5pm.

In 2011, XING's server produced several hundreds of millions log lines per day. Logjam is used by all development teams for monitoring and post-mortem analyses in case of software faults.

MongoDB

MongoDB is a highly scalable document store database (see Section 2.4.1). The storage and querying format is BSON, which makes it compatible to Javascript interpreters. One use case is for instance using it as a backend for simple web applications, without the need of an intermediate web server translating request from and to SQL. Queries can be sent via Ajax [Holdener 2008] to the MongoDB and directly interpret the response data as Javascript objects.

XING uses MongoDB to store the aggregates of logs. The main reason for that decision was the adaptability to new log formats while still offering complex data queries. Thus, new sources can be added without migrating old logging data into a new format.

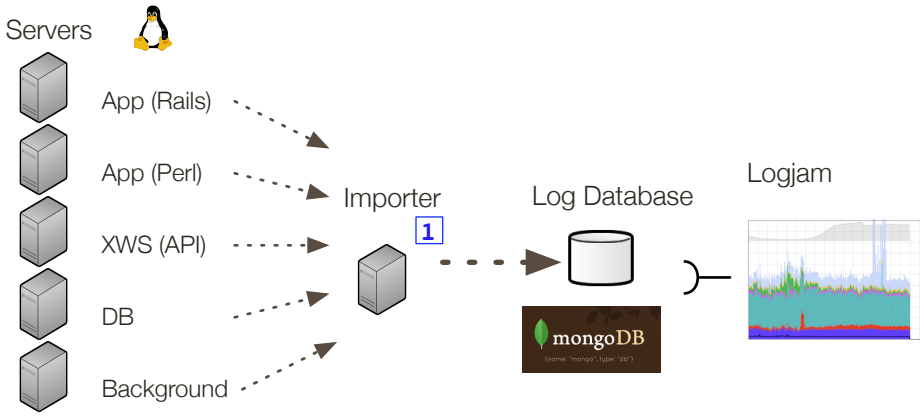

Figure 2.22: All application, database and supporting servers log into the importer at 1 that aggregates all logs and writes them into the log database every second. From there, Logjam accesses and displays the data in a web front end.

Furthermore, the large-scale software system of XING's produces a huge amount of logging data which requires upscaling and high availability of the logging servers. Since MongoDB was designed as a high-performance database it fits the demands of the XING's system architecture.

Chapter 3

The ΘPAD Approach

This chapter refers to the the previously introduced foundations and explains the approach taken by Θ**PAD** in an abstract way. It shows activity diagrams from the *Unified Modelling Language (UML)* as they are the means to specify and document complex processes [Rupp et al. 2007, p. 267]. A mathematical model of Θ**PAD** is defined and will be used for the upcoming implementation in Chapter 4.

Two types of activities are performed in the Θ**PAD** workflow. An offline activity for configuring the measures and three offline activities that are taken by Θ**PAD** to measure performance characteristics and process the values online. Section 3.2 gives an overview of the activities, which are described in detail in the subsequent Sections 3.3 to 3.6.

3.1 Naming

Nissanke [1999] declares Θ as a symbol defining *real time*, hence the theta in the name of Θ**PAD**. This term is often used for programs directly interfaced with some physical equipment [Burns and Wellings 2009]. Other examples are video rendering or streaming applications that require the processing to be completed in limited time defined by the environment. In context of monitoring and anomaly detection, the term *online* provides a greater description. As Shasha and Zhu [2004, p. 119] put it, “even if the data come in forever, I can compute the statistics of the data with a fixed delay from their occurrence”, the focus lies more on the continuity of the data. Since anomaly detection calculates on an infinite stream of measurements of a running system, online defines the name of this approach.

3.2 Activities

As the title *Online Performance Anomaly Detection* indicates, Θ**PAD** addresses the main concerns of detecting unusual performance of software systems online, i.e., while they are running in their production environments. The activities performed

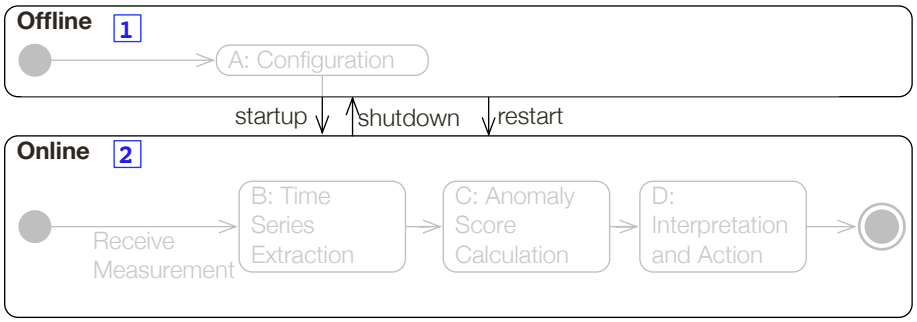

Figure 3.1: 1 In the offline activity A, aspects get configured according to the performance measures. After startup, 2 three online activities are processes on every received raw measurement.

through configuration, startup and runtime of an Θ**PAD** instance are depicted in Figure 3.1 and sketched in the following Sections 3.2.1 - 3.2.4. A detailed description of these activities is provided in Sections 3.3 - 3.6.

3.2.1 A: Configuration

As learned from related research on time series analysis [Chan et al. 2003, p. 2], anomaly detection often depends on the specific environment it is used in. Θ**PAD** therefore introduces the concept of *aspects*, which are separated units of measure and analysis.

For instance, one aspect can measure the response time of a web application. The aspect holds all parameters required to perform anomaly detection. After startup, an Θ**PAD** instance gathers measurements for all configured aspects. Performance anomaly detection based on multiple characteristics and parameters can therefore be performed simultaneously.

3.2.2 B: Time Series Extraction

Θ**PAD** uses forecasting as one step of its anomaly detection and gathers data from a continuous stream of measurements. Since most forecasting algorithms calculate upon time series, this temporal sequence has to be discretized (see Section 2.2.2). The discretization is done by first *dispatching* measurements to their according aspect and subsequently aggregating them on discrete time points.

For instance, measurements of users logging in can be send to Θ**PAD** every time a login happens. For an aspect that expects hourly logins as input, this steps aggregates all these raw measurements (see Section 2.1) and delivers a time series to the aspect.

3.2.3 C: Anomaly Score Calculation

The anomaly detection is broken down into two simpler steps: i) *forecasting* the next probable value based on historic values and ii) *comparing* it with the measured values. The comparison is defined by anomaly detection algorithms that calculate the anomaly score. This score is passed to the next activity for interpretation.

ΘPAD's set of forecasting and anomaly detection algorithms is extensible. Benefits of research in this field can be ported to existing deployments.

3.2.4 D: Interpretation and Action

In this activity, ΘPAD takes action based on every new anomaly score calculated by the previous step. First, each score is stored into a time series database for later post-mortem analyses. Second, the anomaly score is compared with the aspect's threshold, which was configured in activity A. If the calculated anomaly score exceeds the threshold, alerts are reported.

Additionally, time series data can be retrieved from the time series database in parallel to ΘPAD's runtime. This helps refining the aspects and enables post-mortem analyses.

3.3 A: Configuration

In order to apply ΘPAD generically to software systems, the concept has to be configurable. This configuration step is done with knowledge of the system prior to the runtime. The configuration is influenced by the system's characteristics that can be grouped into the following:

1. Measures of the system such as total response time, as described in Section 2.1.
2. The timeliness and magnitude of the measurements gathered from the measures

Different monitored systems can have the same measures and metrics but deviate in the time of measuring. For instance, it is likely that the hourly page visits of an advertisement website produce different time series than the usage of a billing web application system, which is used during normal office hours. Thus, anomaly detection should be optimized to address the characteristics of the system under monitoring.

The ΘPAD approach bundles configuration for these characteristics in entities called aspects. Every aspect is identified with a unique key to be used as a reference for later post-mortem analysis and as an index for the persistent time series storage (see Section 3.6). Table 3.1 shows the symbols used for the aspect configuration entity and all its *configuration properties*.

In the following, the underlying mathematical model is defined. Further on, the *dispatching* process is described, which assigns incoming measurements to the responsible aspect.

Basic terms

Name	Symbol	Type	Example
Key	$\mathcal{K}$	$\mathcal{P}(s^*)$	`string`
Measurand	$\mathcal{M}$	$\mathcal{K}$	response time sum
Measurement	M	$\mathcal{M} \times \mathbb{R}$	(`ttime`, 2.3)
Measure	$\mathbb{M}$	$\mathcal{M} \times \mathbb{N} \rightarrow M$	response time
Algorithm	Λ		Statistical mean
Aspect	A	$\mathcal{A}$	'AD on `ttime`'

ΘPAD instance $\Theta\textbf{PAD} := (\mathcal{A}, \Lambda_{fc}, \Lambda_{ad})$

Name	Symbol	Type	Example
Configured aspects	$\mathcal{A}$		`aspects.yaml`
Available forecasters	Λ_{fc}	Λ	{MEAN, SES,... }
Avail. anomaly detectors	Λ_{ad}	Λ	$\{D_{norm}$, ... }

Aspect configuration $A := (id, a, \Delta_t, \lambda_{fc}, D_{fc}, \lambda_{ad}, \theta)$

Name	Symbol	Type	Example
Identifier	id	$\mathcal{K}$	`mean.D1h.L1d`
Measure	a	$\mathbb{M}$	`all_pages`
Step size	Δ_t	$\mathbb{N}$	$60 * 60$
Forecasting algorithm	λ_{fc}	Λ_{ad}	MEAN
Forecasting window size	D_{fc}	$\mathbb{N}$	$60 * 60 * 24 * 7$
Anomaly detection alg.	λ_{ad}	Λ_{ad}	D_{norm}
Detection threshold	θ	$\mathbb{R}$	0.28

Aspect at runtime $\alpha := (asp, hist, fc, \psi_c)$

Name	Symbol	Type	Example
Aspect configuration	asp	A	`mean.D1h.L1d`
Measurement time series	$hist$	$\mathbb{TS}$	$\{3, 4, 3.3, 4.4, 9.9\}$
Forecasting time series	fc	$\mathbb{TS}$	$\{3.3, 4.4, 9.9\}$
Current anomaly score	ψ_c	$\mathbb{R}$	0.9

Measurement record $R := (t, m_1, ..., m_c) \in \mathbb{N} \times M^c$

Name	Symbol	Type	Example
Measurement	$m = (k, v)$	M	(`vtime`, 2.3)
Measure	k	$\mathcal{K}$	`ttime`
Measurement Value	v	$\mathbb{R}$	2.3

Table 3.1: Symbols used for the mathematical models of **ΘPAD**

3.3.1 Mathematical Model

In the following, the mathematical model is developed. Table 3.1 gives an overview of the term definitions. With the same ordering, details are given in the following sections.

Basic Terms

- $K \in \mathcal{P}(s^*)$, with s^* being a sequence of alphanumeric characters s. K is the set of *keys* or 'strings with a meaning' defined by the environment. Keys are used to identify aspects, measurands, or time series stored in the database.
- Λ as any form of algorithm.
- Point in time are natural numbers ($\mathbb{N}$), also called *timestamps*, that are the number of seconds since a particular date.[1]
- $\Delta \in \mathbb{N}$ defines a *time span* by the milliseconds between two time points.
- SMM's Measurand, measurement and measure as defined in Section 2.1.

ΘPAD Instance

To give a notion of how Θ**PAD** is configured, the subsequent definitions follow a *top-down approach* starting with the coarse-grained definitions as follows. Additionally given is a set of aspects $\mathcal{A}$, a set of forecasting algorithms Λ_{fc}, and anomaly detection algorithms Λ_{ad}. With these types an installation of the system can be defined as following:

$$\Theta\textbf{PAD} = (\mathcal{A}, \Lambda_{fc}, \Lambda_{ad}) \quad . \tag{3.1}$$

Aspect Configuration

From these definitions the aspect type $\mathcal{A}$ can be deduced. Let $a \in K$ be an *attribute* measured by the system. The threshold $0 < \theta < 1$ separates normal from abnormal behavior. Reference in the time series storage is made by setting the identifier $id \in \mathcal{K}$:

$$A = (id, a, \Delta_t, \lambda_{fc}, D_{fc}, \lambda_{ad}, \theta) \quad . \tag{3.2}$$

$\lambda_{fc} \in \Lambda_{fc}$ and $\lambda_{ad} \in \Lambda_{ac}$ are the forecasting and anomaly detection algorithms defined in the aspect's configuration. They will be used at runtime of the aspect as ecplained in the following section.

[1] In UNIX operating systems, the 1 January 1970 at 0:00 is used as a reference point [Leach et al. 2005, p. 28]

Aspect at runtime

At runtime, an aspect loads its assigned configuration and instantiates the following objects:

- asp, the reference to the aspect configuration.
- The time series $hist$, which holds all values previously received by the aspect asp. These values were discretized by using the aggregation function (see Figure 2.5) with the step size Δ_t configured in asp.
- The forecasting *time series* fc is sliding window of $hist$. This time series window moves to the end of $hist$ every time a value gets appended to $hist$. fc is used as the input for λ_{fc}.
- ψ_c, the current anomaly score calculated by activity C. $\psi = 0$ if the anomaly detection algorithm was not executed yet.

As defined in Table 3.1, the aspect at runtime is defined as following:

$$\alpha := (asp, hist, fc, \psi_c) \quad . \tag{3.3}$$

Measurement Record

Measurements put into Θ**PAD** are called measurement records:

$$R = (t, m_1, m_2, ..., m_c) \in \mathbb{N} \times M^c \quad . \tag{3.4}$$

R has c raw measurement which each having a key denoting a measurand of the system under monitoring, for instance $(256220100, (\texttt{vtime}, 2.3), (\texttt{db}, 1.5), ...)$. These records are continuously received from a temporal sequences as introduced in Section 2.2.2.

For all $m_i = (k_i, v_i)$, $m_j = (k_j, v_j)$ with $i \neq j$ we additionally require $k_i \neq k_j$, that is: components $m_1, m_2, ..., m_c$ of R have distinct keys. Hence, R defines a function

$$f_R : \{k_1, ..., k_c\} \to \mathbb{R}, \quad k_i \mapsto v_i, \tag{3.5}$$

mapping the keys to the values.

3.3.2 Aspect Lifecycle

Configured

Before Θ**PAD** accepts input data, aspects have to be configured and instantiated in order to collect data according to their attribute. In the instantiation process, time series $hist$ and fc get created with the according configuration properties.

Running

In this state, an aspect accepts measurements with the key according to the attribute it was configured for. Every time Θ**PAD** dispatches a new measurement

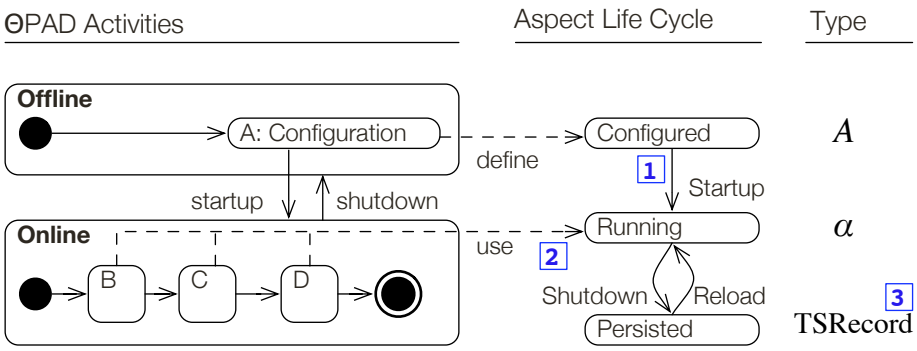

Figure 3.2: The aspect lifecycle corresponds with the activities of **ΘPAD**. On startup (1), the aspect configuration gets instantiated into *running* aspects. 2 indicate ways in which the **ΘPAD** instance accesses the aspects in the process of anomaly detection. The type *TSRecord* 3 is defined in Section 3.6.

to the aspect, it gets appended to the time series $hist$, the forecasting window fc gets moved and subsequently the anomaly detection is executed. This process is described in detail in Section 3.4.

Persisted

When shutting down, the time series database holds all time series points. The aspect only resides as a configuration entity in **ΘPAD**. When starting up again, the time series $hist$ gets filled from possibly previous data and the time series fc gets created as a sliding window with the configured length D_W.

3.4 B: Time Series Extraction

The steps taken to extract time series are described as *discretization* (see Section 2.2.2) of a temporal sequence of measurements. As an overview of this chapter, Figure 3.3 shows the steps of this activity.

B.1 Dispatching

The activity of processing measure records is called *dispatching*. The approach defines one point which receives the data from the continuous monitoring. This point has to inspect those incoming measure records and sort them to the according aspects configured in the system.

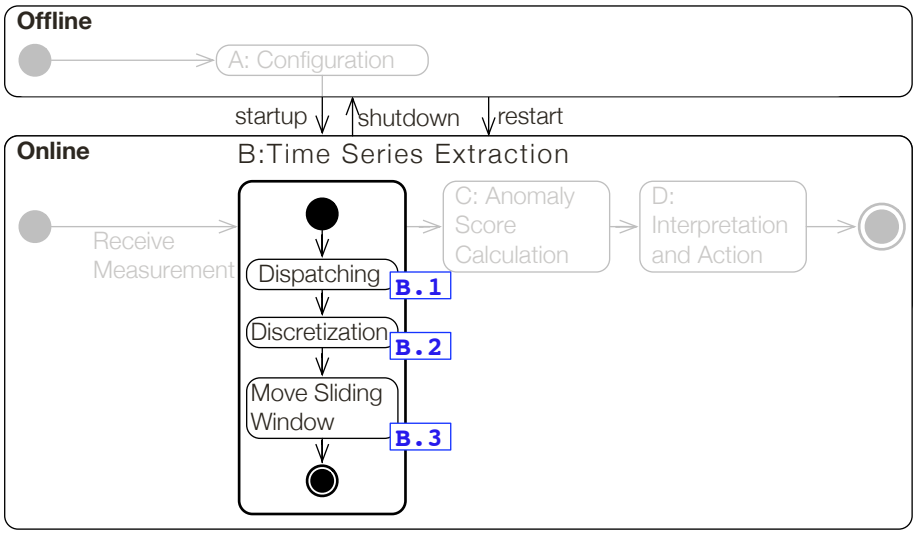

Figure 3.3: Steps taken to extract time series from the received temporal data

This step extracts the measurements $m = (k, v)$ out of each measure record R and assigns them to the aspects matching the measurement's measurand k to the aspect's measure a.

$$f_{dispatch} : \mathcal{A} \times R \rightarrow \alpha \quad , \tag{3.6}$$

$$(A, R) \mapsto f_{dispatch}(A, R) \tag{3.7}$$

with

$$f_{dispatch}(A,R) = \begin{cases} A & : \ f_R(a) \text{ undefined} \\ (a,\ hist \oplus f_R(a),\ fc \oplus f_R(a),\ \lambda_{fc}, \lambda_{ad}) & : \ f_R(a) \text{ defined.} \end{cases}$$

We assume $hist \oplus f_R(a)$ to append the measure value $v = f_R(a)$ to the time series $hist$. For every incoming measure record $r \in R$, the dispatching process applies the according $f_{dispatch}$ method to all configured aspects, as depicted in Figure 3.4.

Measurements received from **ΘPAD** can be of arbitrary size and information. They can be gathered continuously and are not required to appear at discrete time points. As introduced in Section 2.2.1, the data model of continuous series of events is called a *temporal sequence*.

Yet forecasting algorithms rely on time series in order to compute the next values based on past data. To address this requirement, **ΘPAD** uses aggregation as a special form of discretization as described in Section 2.2.2.

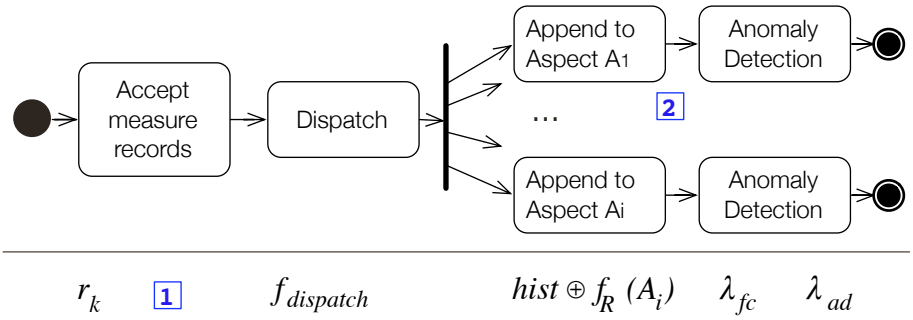

Figure 3.4: The dispatching process assigns incoming measurements $r_k \in R$ 1 to the aspect $A_i \in \mathcal{A}$ 2, which are configured for the measurements.

B.2 Discretization

3.4.1 Amount of Collected Data

As defined in Equation 3.5 the function f_R maps its keys to the measured values. The previously defined dispatcher has to go through the whole set but discards all those measurements that have no corresponding aspect.

The amount of data being collected depends on the number of measurements, their according measures and the configuration of the Θ**PAD** service (see Section 3.1). Assuming every measurement yields values for every aspect's attribute, the aggregated number of discrete time series points $SIZE$ can be calculated for a given time span $\Delta_{runtime}$ as following:

$$SIZE(\Theta, \Delta_{runtime}) = \sum_{a \in \mathcal{A}} \frac{\Delta_{runtime}}{\Delta_{t_a}} \quad . \tag{3.8}$$

Every time a value is appended to the aspect's time series $hist$, Θ**PAD** performs the anomaly detection. The time spans between these processing steps depend on the step size Δ_t. Hence, the computation needed for an Θ**PAD** instance is influenced by all step sizes defined in the aspects's configuration properties.

B.3 Move Sliding Window

Forecasting algorithms used by anomaly detection as described in the foundations (Section 2.2), give different weights on input data. For example, the SES weighs the values of the *forecasting window* exponentially (see Equation 2.6). Thus, modifications of the aspect's *window size* $\Delta_W \in \mathbb{N}$ affect the output of the forecasting algorithm. Since Δ_W can be set for every aspect, the anomaly detection can be refined further.

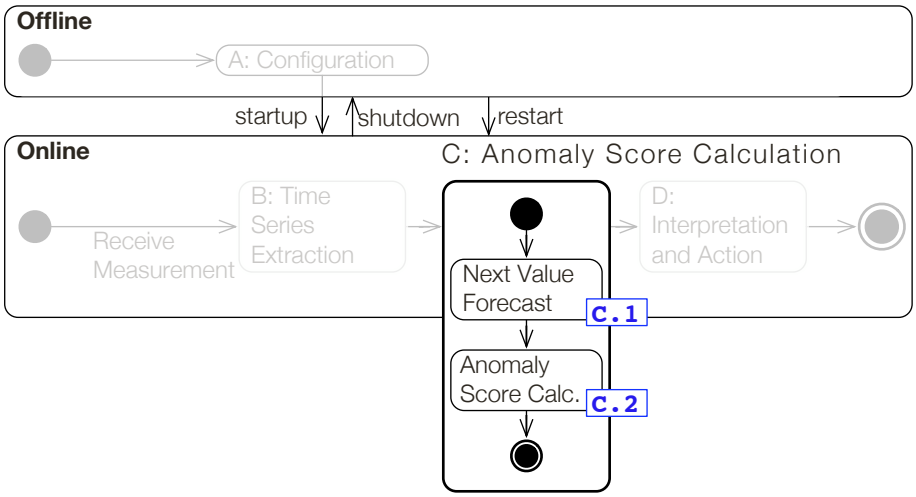

Figure 3.5: C.1 Anomaly score calculation is achieved by forecasting the next value with the configured forecasting algorithm. C.2 compares and normalizes the values and builds the anomaly score $0 < \psi_c < 1$ to indicate the *level of abnormality*.

3.5 C: Anomaly Score Calculation

The basic idea of Θ**PAD**'s anomaly detection is based on the intermediate step of calculating a level of 'abnormality'. This so-called anomaly score is a metric indicating the deviation of the current measure from the reference model. We assume the forecast fitting the reference model, this is the normal behavior of the performance characteristic. The input of the actual course is given every time activity B delivers a new value, as described in Section 3.4. In combination, both values can be used to calculate the anomaly score.

Reasons to split the anomaly score calculation are to keep the algorithms simple and exchangeable. For the forecasting algorithms, simplicity means only needing one forecasting value. Thereafter, the succeeding score calculation can be achieved by distance calculation based on only two values. The resulting algorithm signatures follow the *divide and conquer* principle and allow the configuration and evaluation of other algorithms.

In the foundations, the mathematics of forecasting (see Section 2.2.3) and anomaly score calculation (see Section 2.3) were defined. The following sections put these formulas into an algorithmic context.

C.1 Next Value Forecast

The anomaly detection of Θ**PAD** is online and relies on the current measure as an input parameter. Since there is only one current value to compare to, the lead time

of any applied forecasting algorithm is 1. This leads to a better algorithm run time and only one value ($F_1 \in \mathbb{R}$) to calculate:

$$F_1 = \lambda_{fc}(fc) \quad . \tag{3.9}$$

C.1 Anomaly Score Calculation

In this approach of online anomaly score calculation, an incrementally sliding forecasting window is used as shown in Section 2.3.4. Every time a value is appended, one old time point gets discarded. Thus, the window always contains the newest values.

Accordingly, anomaly detection can be based on the newest value as well. This gives the advantage of updating the reference model on every update and simplifies the anomaly score calculation.

Due to this simplification, the anomaly score is a normalized distance measure with only two inputs. At every current time point c, the anomaly score can be calculated by an algorithm λ_{ad}. It calculates the current anomaly score ψ based on the time series input $X = \{x_1, ..., x_c\}$, the forecasting algorithm λ_{fc} and a forecasting window $W = \{x_{i-D_W}, .., x_{c-1}\}$ with $\Delta_W = \Delta_X$:

$$\psi_c \quad = \quad \lambda_{ad}(\lambda_{fc}(W),\ x_c) \quad . \tag{3.10}$$

This activity puts out anomaly scores consistent throughout the time the system is monitored. These scores are given to the last activity to enable the interpretation of behavior.

3.6 D: Interpretation and Action

This activity receives the current data points of all three relevant time series as input. These hold the measurements, forecasts and anomaly scores at discrete time points. The previous activities held this data in a volatile runtime context. In this activity the time series get persisted (D.1), anomaly scores are interpreted and in case of abnormal behavior, reported.

Furthermore, this step interprets the data by comparing the anomaly score with the configured threshold (D.2) and alerts the surrounding system in case this threshold is exceeded (D.3). Figure 3.6 puts these steps into context of the overall Θ**PAD** process.

D.1 Persistent Score Storage

This step receives all three time series as input. Combined with the aspect, the following data get inserted in the time series storage on every new input.

- Measurements from the measure defined in the aspect
- Forecasts for every time point

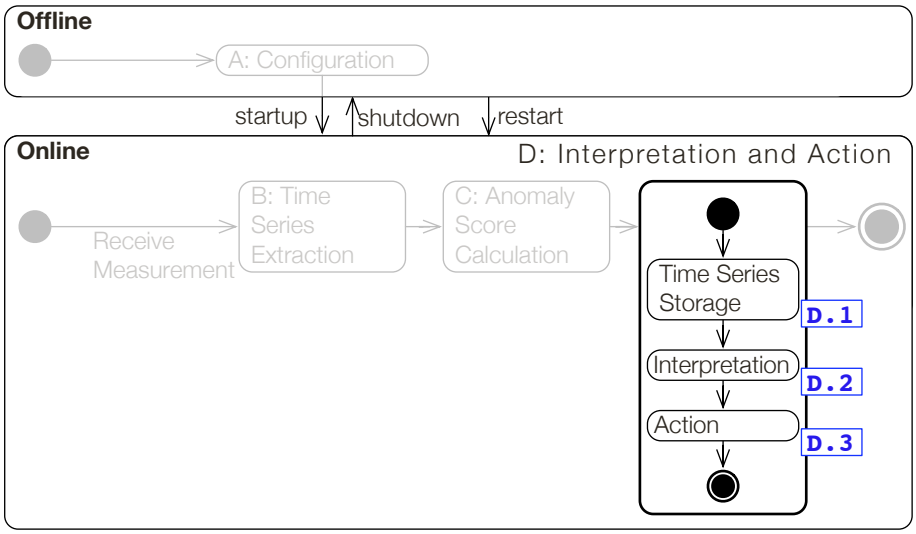

Figure 3.6: Activity details of step D.3. Calculated anomaly scores get stored into the database and alerts are propagated whenever they exceed a certain threshold.

- Anomaly scores as calculated by the anomaly detection algorithm
- The aspect's identifier to refer to the data origin

Since the time series all have the current timestamp in common, one record stored into the database can be defined as a tuple:

$$\text{TSRecord} := \mathbb{N} \times \mathcal{K} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \quad . \tag{3.11}$$

To clarify, Equation 3.12 gives an example for a record at time t_{42} for an aspect with $id = \texttt{aBest}$ as following:

$$\begin{aligned} \text{TSRecord}_{42} &= (t_{42}, id, X_{42}, F_{42}, \Psi_{42}) && (3.12) \\ &= (459819300,\ \texttt{aBest},\ 300042.24,\ 270042,\ 0.4). && (3.13) \end{aligned}$$

This storage format is compatible with common formats and databases as described in Section 2.4.1. Apart from the algorithms, the persistent data storage can be changed as well.

As an output, this activity propagates the anomaly scores to the score interpretation (D.2) as depicted in Figure 3.7.

D.2 Score Interpretation

In this step, the threshold θ, as defined in the aspect (see Section 3.3), is compared to the anomaly score $0 < \psi < 1$. If the score exceeds ($\psi > \theta$) the threshold, the behavior is assumed to be abnormal, hence ψ is delivered to the next step D.3. If not, the activity is terminated after this point.

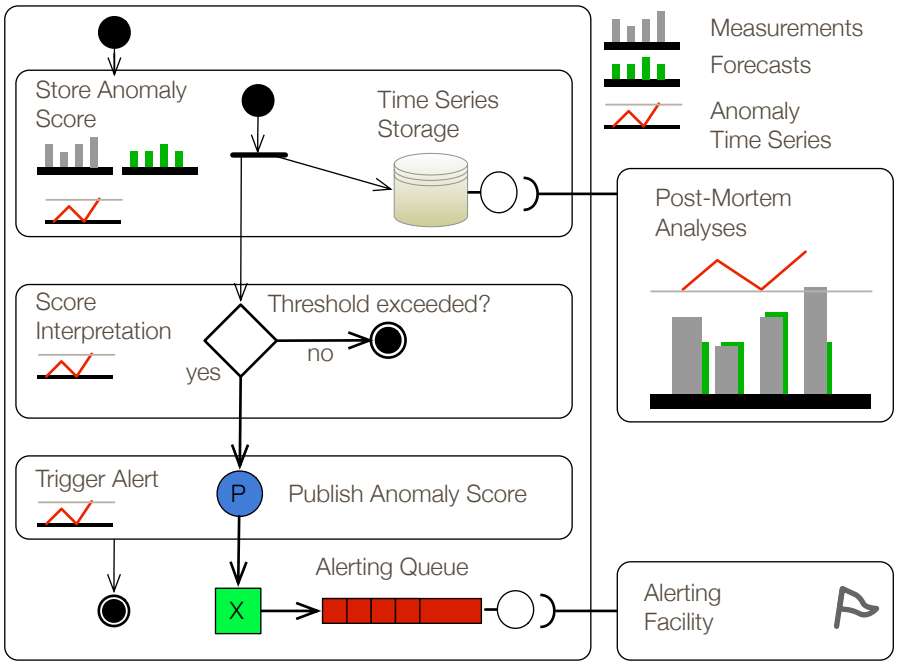

Figure 3.7: ΘPAD interpreting the anomaly score and offering post-mortem analyses based on the origin of the calculations. The AMQP actors (see Figure 2.19) are used for the alerting queue.

D.3 Action

This step receives anomaly scores that are higher than the threshold. Whenever an input is received, the score gets published onto a queue that follows the FIFO principle: Subscribers get the first published value ahead of the later propagated scores.

An alerting facility can therefore subscribe to the queue and trigger alerts whenever a message is published. As additional information, the transferred value is sent along, thus serving as a degree of abnormality to counteract more effectively.

Furthermore, additional characteristics of the monitored system can be configured outside of ΘPAD. For instance, a collective anomaly that gets alerted could be directly followed by a large contextual anomaly as described in Section 2.3.4. An alerting facility could address that by keeping the alert switched on until appropriate action is taken.

Post-Mortem Analyses

Since every calculated anomaly score gets stored in the persistent database alongside with the data origin, the algorithm input of every score is known. In combination with the aspect configuration, the origin of alerts are stored for future inspection.

This enables post-mortem analyses in order to gather information on the reasons of occurring anomalies and correlations amongst the aspects. Additionally, scenarios can be replayed in order to test other aspect configuration properties.

Chapter 4

Design and Implementation of the ΘPAD System

This chapter takes the Θ**PAD** approach of the previous chapter and concretizes it in software modeling and design. To define the design decisions, structural UML diagrams, such as *component*, *deployment* and *class* diagrams are used. For details and definition of the used standard we refer to Rupp et al. [2007].

The following Section 4.1 uses the Θ**PAD** activities and the previously described technology stack (see Section 2.4) to form the requirements of the software implementation. Section 4.2 lists the supporting software packages and Section 4.5 goes into detail on how Θ**PAD** controls $\mathcal{R}$ to do the calculation. Since the software runs as a Kieker plugin, Section 4.4 explains how the interfaces are used and how Θ**PAD** is started up in this context. Finally, the anomaly score output and interpretation and alerting technologies are described in Section 4.6.

4.1 Requirements

In order to employ a solution for the formal specification of the approach in Chapter 3, a software solution has to be implemented, which fulfills that approach. Throughout the following sections, the requirements classification of Schwinn [2011, p. 22] is borrowed.

Functional requirements assure that Θ**PAD** addresses the problems, which were motivated in Section 1.1. All necessities are listed, which are mission-critical for providing a solution for the given problem field.

To apply a solution on software systems, the surrounding *framework conditions* have to be met. These requirements serve to assure that Θ**PAD** works on the technology stack of the case-study environment. Furthermore, *non-functional requirements* describe how the implementation addresses certain quality needs.

4.1.1 Functional Requirements

The functional requirements (FRs) correspond directly to the activities depicted in Figure 3.1.

FR1: Environment Configuration

ΘPAD has to be deployed in varying environments, which measure performance by different attributes. The approach introduces the aspect model, which provides this configurability. In order to have multiple aspects in one **ΘPAD** installation, a configuration register is needed, which can be equipped with the aspects fitting the environment.

FR2: Time Series Extraction

Forecasting and consequently anomaly detection algorithms require equidistant time series data as input. The process described in Figure 2.5 extracts discrete time series points from a temporal sequence using a discretization function. **ΘPAD** uses the aggregation function and requires the parameter step size to define the discrete time series points. Hence, every aspect has to be configurable in order to build its internal time series.

FR3: Anomaly Score Calculation

The two-step anomaly detection described in Section 3.5 is crucial to the functionality of **ΘPAD**. One instance of the system, as defined in Equation 3.1, is called *installation* in this context. The **ΘPAD** called type defines sets of attributes as input parameters, which define the available algorithms for forecasting and anomaly detection. To address system-specific characteristics, these algorithms have to be configurable and exchangeable.

FR4: Interpretation and Action

One possibility **ΘPAD** offers is post-mortem analysis as described in Section 3.5. The complete history of anomaly scores has to be stored in a database. To gain knowledge from this, data has to be accessible and displayed in an appropriate way. For complete post-mortem analysis, we additionally require **ΘPAD** to store all previously measured data and the forecasted valued calculated in between.

In a final step (see Figure 3.7), anomalies have to be propagated to alerting facilities. As of Section 3.6, the approach is to use queueing as a connection to alerting facilities defined by the surrounding system.

4.1.2 Framework Conditions

FC1: Case-Study Environment Compliance

The case study for ΘPAD will be the large-scale software system at XING AG, Hamburg (see Section 1.1). The functional requirements FR1 to FC4 already address the approach that was designed to fit the problem space in general. This requirement set the necessity of the implementation to be adaptable to the case study environment previously described in Section 2.4.4.

4.1.3 Non-Functional Requirements

NFR1: Quality

Software is never free of errors and every additional line of code introduces a certain probability of programmer's errors, so-called *bugs*. The development process needs to address this with the use of certain paradigms and tools. Furthermore, the implementation has to be evaluated on production data of the case-study environment, thus requiring it to be stable in terms of robustness.

NFR2: Graceful Restarting

Since ΘPAD is expected to measure performance online, it may never get shut down. However, restarts can be required when reconfiguring the setup or migrating to different hardware. Another motivation, as described by Huang et al. [1995, p. 2], is software rejuvenation: restarting is used to clear stale run time data, which clogged up the server. Thus, ΘPAD is required to store the work data persistently and be able to regain the old status when restarting.

NFR3: Big Data Processing

Neither the number of aspects configured in ΘPAD nor the detail of the aggregated data (denoted as Δ in Section 2.2.1) should be limited. Multiple aspects can be tried out in order to find the parameters that fit the surrounding domain best. This adds the requirement of storing big amounts of time series data and have them accessible fast.

NFR4: Reusability

Historically grown software systems are often customized to a high degree. Hence, the environment configuration (FR1), has to be supported by an architecture that can be be modified and easily extended. Additionally, a modular structure gives the chance of porting and reusing parts of ΘPAD in other software.

4.2 Supporting Software and Libraries

The Θ**PAD** approach is implemented with the support of third-party and own libraries that comply with its open-source license. Benefits are reduced development time, code quality and encapsulation. In the following, the included libraries and its purposes are listed:

- Section 4.2.1: Esper Complex Event Processing for time series extraction
- Section 4.2.2: MongoDB, storing the time series
- Section 4.2.3: Rserve to connect Java to an $\mathcal{R}$ server
- Section 4.3: `TSLib`, Θ**PAD**'s own time series library outsourced into an own open source library

4.2.1 Esper Complex Event Processing Platform

Esper by Espertech is an open source event processing library licensed under the *General Public License (GPL)*. It can be packaged with other open source software, but also offers an enterprise license for commercial redistribution. Gualtieri et al. [2009, p. 6] compared nine *Complex Event Processing (CEP)* engines and refers to EsperTech as the leading open source provider for CEP. It offers processing for temporal sequences, which can be accumulated, correlated, and queried. The Θ**PAD** system uses Esper to perform time series extraction and aggregation of temporal sequences of measurements as described in the approach, Section 3.4.

Esper has its own SQL-like syntax the Event Processing Language (EPL) that includes aggregation of time series. Listing 4.1 states the syntax for an aspect with aggregation time series X and a step size $\Delta_X = 1000$ milliseconds.

```
1  select sum(value) as aggregation
2  from MeasureEvent.win:time_batch( 1000 msec )
```

Listing 4.1: Esper syntax (EPL) describing the aggregation to time points every 1000 milliseconds

Since Esper provides a strong feature set and is embeddable in Java applications [Gualtieri et al. 2009, p. 8], it matches the programming language of Θ**PAD** and can run in the same context as Kieker.

4.2.2 MongoDB

As of requirement NFR2, the implementation has to store the data persistently, To achieve that, Θ**PAD** uses a document store database as introduced in the foundations, Chapter 2.4.4. The architecture decision to use this kind of database came from the already existing knowledge gathered with Logjam. An additional advantage of MongoDB is the ability to process large amount of data (NFR3). Due to the simple data model, concepts like *relations* or sophisticated SQL queries are not needed.

The case-study environment already offers MongoDB, as described in Section 2.4.4. This open source database comes with standalone server and many

driver implementations. Since Java and Ruby are supported, MongoDB can be used for Θ**PAD** without huge development effort.

Figure 4.7 shows the class `MeasureSeries`, which implements the listener interface `IDataBeat` and hence listens on `AggregationDataBeat`. Thus, it receives the aggregated values from Esper at every discrete time point according to the aspect. Every time a new value is appended, forecast and anomaly score is calculated and added to the record in `MeasureSeries`. Subsequently, `MongoDataBeat` uses the same approach to chain itself to `MeasureSeries`. It receives complete aggregates combined with the output of the aspect's algorithms.

4.2.3 Rserve

The default behavior of $\mathcal{R}$ is providing a command line interface. With certain functions, libraries and code can be loaded and executed from the file system. Since Θ**PAD** is required to process large amounts of data (NFR3), a dedicated $\mathcal{R}$ server can provide scalability.

Rserve is a library that is loaded into $\mathcal{R}$ and opens a port for remote control. Authorized clients can connect and thereafter execute commands on the running $\mathcal{R}$ instance. JRclient is a Java library complying to the protocol offered by Rserve. It provides classes to translate the communication between Java and $\mathcal{R}$.

4.3 TSLib Component

The Java library `TSLib` is a part of Θ**PAD** that got outsourced in order to be used in other projects dealing with time series analysis and forecasting (see Section 5.5). As the date of this thesis, there was no third-party library available that provided the desired feature set. This component makes Θ**PAD** reusable and hence addresses NFR4. The main attributes of `TSLib` are as follows:

- **Common definition:** The foundations state the definition of time series used throughout literature (Section 2.2). The Java code reflects that definition.
- **Forecasting algorithm interfaces:** The algorithms used by the Θ**PAD** approach (Section 3.5) implement a common interface. Thus, later extension is possible without breaking code that uses `TSLib`.
- **Outsourcing calculation to $\mathcal{R}$:** In order to move computation-heavy processes to other servers and to utilize existing algorithm definitions, a connection to $\mathcal{R}$ is provided.

The internal structure of `TSLib` is depicted as an UML class diagram in Figure 4.1. Details of the classes and a runtime example are given in the following sections.

TimeSpan

The `TimeSpan` class defines the step size Δ (see Section 3.3.1), which is used to determine the distance between time points of time series. This class does not exist

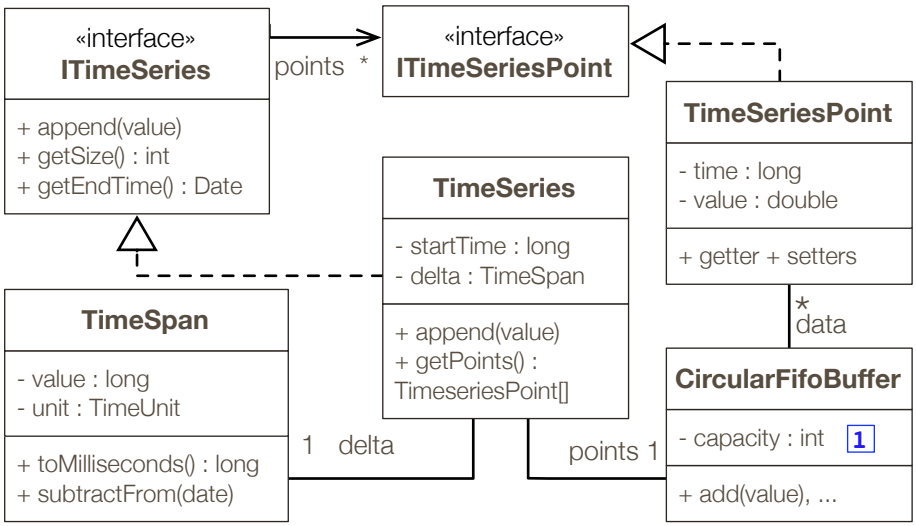

Figure 4.1: The classes provided by `TSLib` are arranged around the `TimeSeries` class, which uses a circular buffer. This data structure discards old time series points when exceeding a certain capacity 1.

in this form in the Java standard library but is a meaningful wrapper for calculations amongst time distances in time series.

For instance, the aspect approach of Section 3.3 defines this in the configuration. The `TimeSpan` class is this representation in Java code provided by the `TSLib`. It uses Java's `TimeUnit` enum for defining the the unit of the specified time. Since this also reflects to the attributes specified in the aspect configuration, the YAML configuration is easy to maintain. The following example is one aspect taken from ΘPAD's configuration file:

```
1  - !aspect
2      delta: 15
3      deltaUnit: MINUTES
4    ...
5      lengthValue: 1
6      lengthUnit: HOURS
```

Listing 4.2: The aspect configuration uses YAML syntax and makes defining timespans understandable by accepting time points as tuples of integers and `TimeUnit`s.

Time Series

Throughout this work, time series with a certain length D_X (for a time series X) are used. Section 2.6 shows an example using a sliding window, which is used for forecasting. The Java class `TimeSeries` is the implementation of this model and can be instantiated with a capacity parameter that restricts the instance to a particular count of `TimeSeriesPoint`: When appending new time points the buffer discards old values that are above its capacity.

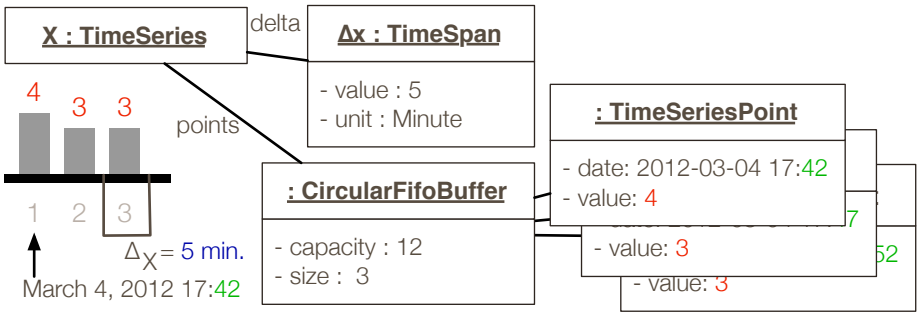

Figure 4.2: Example of a time series with three points instantiated in `TSLib`. The `CircularFifoBuffer` will discard value 4 of $t = 1$ after inserting the 13th value at March 4 2012 18:47, which exceeds the capacity of 20.

Figure 4.2 shows an object diagram of a time series instance with three time points in a `CircularFifoBuffer` attached to an instance of `TimeSeries`. At run time, the buffer is used to encapsulate the memory management for time series restricted to a certain step size.

4.4 ΘPAD Kieker Plugin

The context, in which **ΘPAD** runs, is provided by Kieker, the performance monitoring and dynamic analysis framework described in Section 2.4.2. To benefit from Kieker's existing implementation base, **ΘPAD** is implemented as a plugin and follows the interfaces required by Kieker's plugin container.

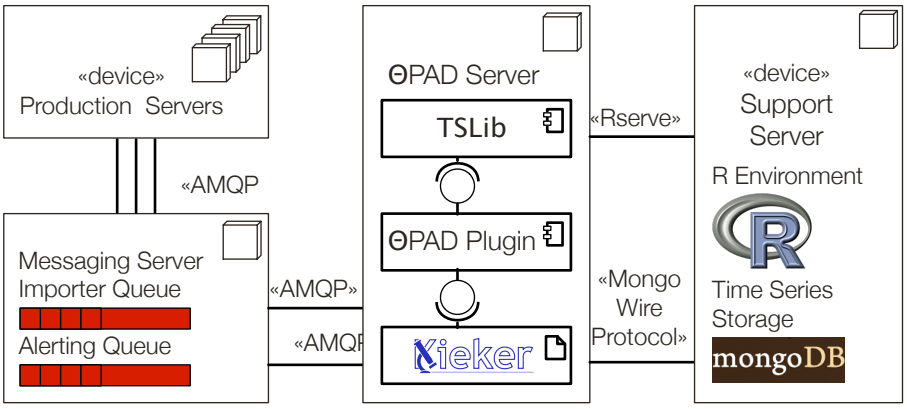

Figure 4.3: Deployment of **ΘPAD** in a Kieker server with two optional servers for infrastructure and support.

Figure 4.3 shows an example deployment setup with distributed systems for infrastructure, monitoring, and support. For a large-scale software system, this high degree of distribution is common. However, all systems could be run on one

machine and communicate locally as well. For this implementation, Kieker is used as a base only without additional analysis means. However, it is possible to deploy **ΘPAD** alongside other performance analysis plugins.

Following are the previously introduced libraries and technologies woven together in Kieker (Section 4.4.1). The plugin's internal structure is described in Section 4.4.2. Ultimately, Section 4.4.3 explains how good code quality was achieved in the development process.

4.4.1 Integration into Kieker

ΘPAD uses the Kieker monitoring framework written in Java as a base for its implementation. The connection to the anomaly detection code is made by a plugin container Kieker offers. The plugin pattern is used when certain behavior is based on different implementations that have to be configured in a central runtime environment [Fowler 2003, p. 500]. To start **ΘPAD** in a Kieker environment, all necessary depending libraries, the `kieker.jar` and the plugin code have to be accessible in the same class path.

ΘPAD gets instantiated at startup of the Kieker server. Both main architectural components of Kieker, *Monitoring* and *Analysis* are used to route the measurements to the plugin. The data flow from input to output is illustrated in Figure 4.4.

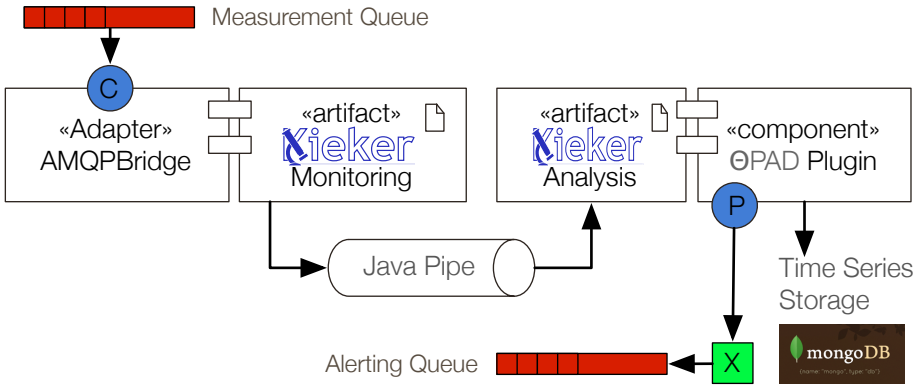

Figure 4.4: The coarse-grained architecture follows the linear data flow of the approach (see Chapter 3). The `AMQPBridge` adapter translates the monitored system's measurements to Kieker records and therefore makes **ΘPAD** reusable in other environments (NFR4). This graphic uses the AMQP notation of Figure 2.19.

Raw measurements, encoded as JSON strings, are sent to the measurement queue by the system under monitoring in a temporal sequence fashion as described in Section 3.4. The Kieker Monitoring component, then deserializes these strings and transforms them into measurement records as defined in Equation 3.4. Subsequently, these records are put into a Java Pipe forwards data in memory according to the FIFO principle. Figure 4.5 shows the corresponding class instantiation.

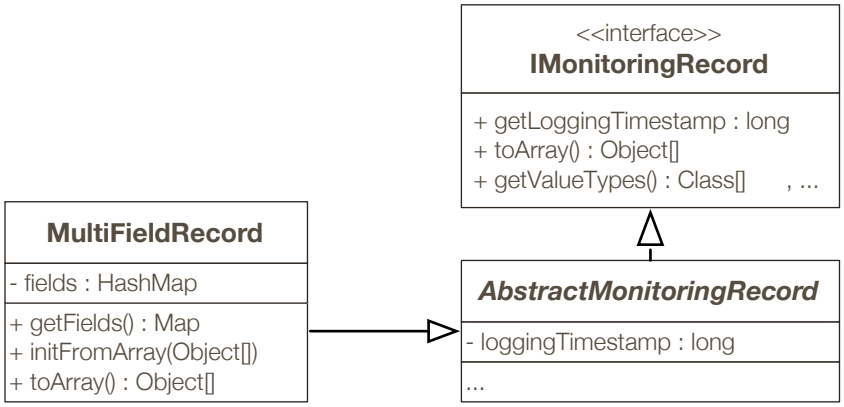

Figure 4.5: The `MeasurementRecord` class is used as a wrapper for the incoming JSON data corresponding to the type measurement record defined in Table 3.1

The Kieker Analysis component consumes these deserialized Java objects and forwards them to the Θ**PAD** plugin, which takes care of extracting time series, performing the anomaly detection and publishing anomaly scores to the according queue in case of a detection. The plugin internally loads aspects configuration files holding n aspects with the mathematical type $\{A_1, ..., A_n\}, A_i \in \mathcal{A}$ as defined in Section 3.3.1 and instantiates them (see Section 3.3.1). The resulting runtime aspects $\{\alpha_1, ..., \alpha_n\}$ each hold time series $hist$ and fc in memory. In periods configured by the step size, updates to these time series are stored in the MongoDB for post-mortem analyses.

In order to implement Θ**PAD** as a Kieker plugin, it has to offer a class implementing the `IAnalysisPlugin`. This class gets instantiated and called with the `init()` method when Kieker loads its plugins. Accordingly, the `terminate()` method unloads the plugins. This has the advantage of giving the startup and shutdown responsibility to the surrounding framework. Additionally, a plugin in Kieker offers itself for the analysis of certain measures. The resulting pattern is close to the chain of responsibility [Gamma et al. 1995, p. 223] and gives the possibility of using multiple plugins in one Kieker server.

4.4.2 Architecture

Architecture describes the structure, interfaces, and the cooperation of components of a system [Starke and Hruschka 2011, p. 24]. The Architecture of Θ**PAD** is derived from the linear data flow of the system. Internally, this linearity is proceeded by a custom class structure which developed in Θ**PAD** and was called named 'Data Beat' pattern. Figure 4.4 shows the coarse-grained chain of processing.

The observed system puts measures on the importer queue. The Kieker Monitoring component acts as an AMQP consumer, denoted with the blue circle. The bridge between Monitoring and Analysis components is achieved by config-

uring Kieker to work with a Java Pipe[1]. This in-memory pipe is enabled in the `kieker.properties` by setting the following configuration:

```
1  kieker.monitoring.writer=
2    kieker.monitoring.writer.namedRecordPipe.PipeWriter
```

Listing 4.3: Kieker can be configured to use a Java pipe to transfer measurements from the monitoring to the analysis component

Measurements then are routed to the **ΘPAD** plugin as shown in Figure 4.4. Thereafter, the anomaly detection algorithms are executed. Periodically, as determined by the configuration property Δ (see Section 3.3.1), aggregated measures and anomaly scores are written into the time series storage. In case anomalies are detected, they get published via AMQP. Therefore, **ΘPAD** also acts as a publisher, which puts anomalies on the anomaly queue. The surrounding system can subscribe to that queue in order to trigger alerts. Details of this output are described in Section 4.6.1.

Figure 4.6 shows the libraries described previously and the coarse-grained internal structure as a UML component diagram. The dashed arrows are `use` relationships. Their direction corresponds to the data flow of Figure 4.4.

Data Beat Pattern

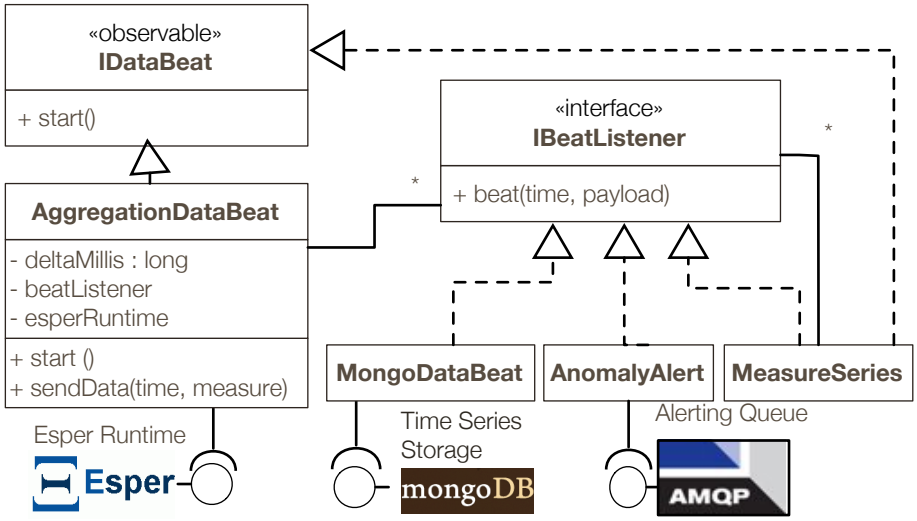

Figure 4.7: Class diagram of the DataBeat Pattern

In the normal program flow, time series objects undergo many different operations such as aggregation, forecasting, anomaly score calculation, and storage. To deal with this complexity, we created an own pattern that uses separation of concerns as suggested by Starke and Hruschka [2011]. This pattern is called `DataBeat`

[1] A concept similar to UNIX pipes: `http://www.linuxjournal.com/article/2156`

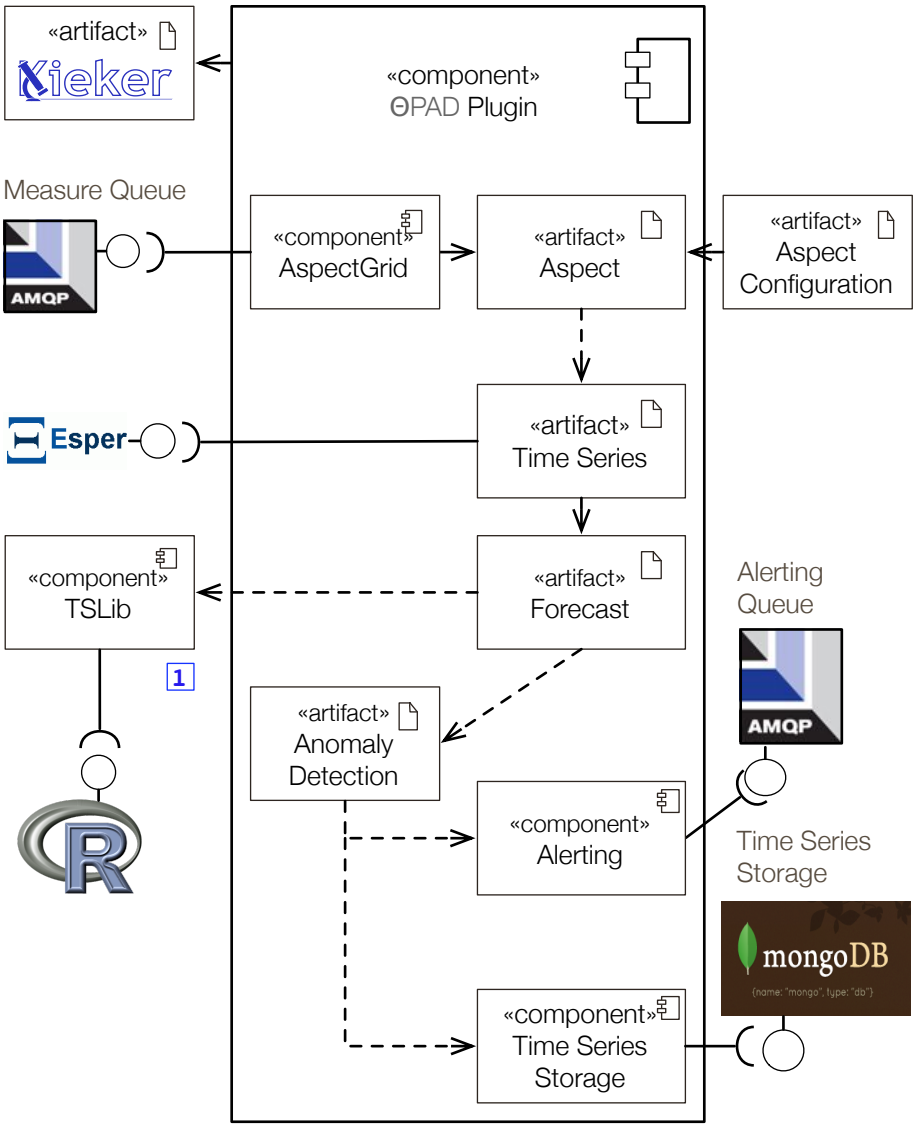

Figure 4.6: The components of the **ΘPAD** plugin architecture use several supporting software and libraries outside its own code base. This diagram reflects the data flow described in the approach in Chapter 3. `TSLib` (1), a part of **ΘPAD**, was outsourced to an external library for reusability in other work.

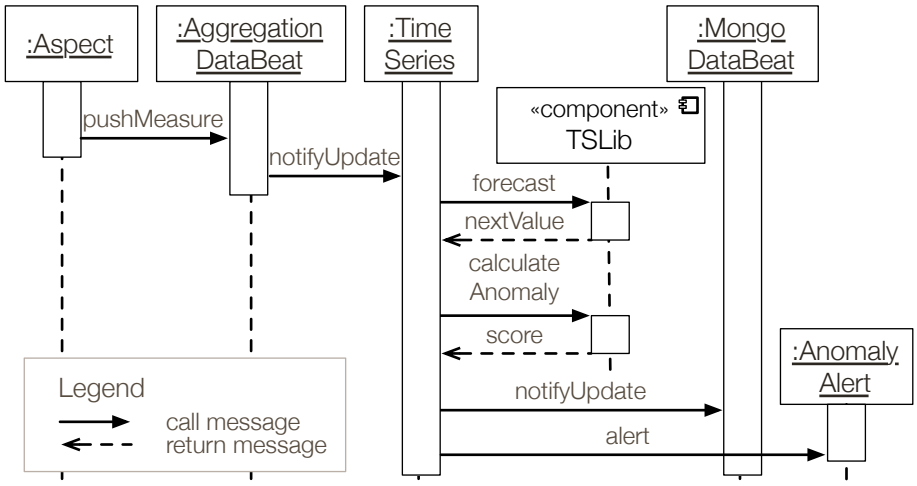

Figure 4.8: This call sequence that gets processed every time a new measurement record is received. It follows the steps of activity B of the approach as depicted in Figure 3.7.

since it involves different objects reacting to periodic changes in time series data. As depicted in Figure 4.7, it includes the *observer pattern* as proposed by Gamma et al. [1995, p. 293] which notifies objects when the `Observable` class is changed. `AggregationDataBeat` is the key class, which converts a temporal sequence to time series with equidistant time points as described in Section 3.4. Hence, all declared listeners can operate on time series data with equidistant time points.

At runtime, the classes following this pattern communicate sequentially as illustrated in Figure 4.8. The concept of the data beat pattern is influenced by the `Decorator` introduced by Gamma et al. [1995, p. 175].

4.4.3 Unit and Integration Tests

In order to achieve a high software quality as required by NFR1, the Java code was written with *test-first programming*. 'Code Complete', the programmer's standard literature by McConnell [2004, p. 504] refers to this practice as being the most beneficial of the recent years. The importance is also proven by its inclusion in new project management principles like *Extreme Programming (XP)* [Beck and Andres 2004, p. 51].

This methodology is particularly known for good quality due to extensive use of tests and also called *Test-Driven Development (TDD)*. A strict TDD approach becomes even more important since the core of Θ**PAD** was written without a second programmer. Thus, it helped writing high quality code without daily supervision.

In the resulting implementation, every important class has a corresponding test class conforming to the structure of the test framework JUnit. Periodically, all dependent tests are run to ensure the change did not compromise other functionality.

Additionally, the continuous integration server Jenkins was set up to run the test suite after every `push` to the git repository.

4.5 $\mathcal{R}$ Algorithms

The functional requirement FR3 defines the core functionality of ΘPAD being anomaly detection. As of the approach, detection is achieved by forecast and comparison. Additionally, ΘPAD is required to be configurable for the environment's characteristics (FR1) and be deployed in the case study system (FC1). Therefore, this implementation in software has to include configurable and extendable algorithms for both steps in the detection process.

Additionally, it is required, that ΘPAD provides a basic set of algorithms, which are used to prove the functioning of its approach. To support a fast development process and to be extensible in future, ΘPAD has to rely on existing implementations of algorithms.

Java libraries for forecasting exist such as OpenForecast. However, $\mathcal{R}$, as introduced in Section 2.4.3, offers an abundance of mathematical methods of all fields with a large expert community. $\mathcal{R}$ is said to be the prominent open source statistic language and environment. Section 4.2 will introduce the package Rserve, which is used to move the computation of algorithms in $\mathcal{R}$ to a dedicated server in order to deal with big data as demanded by NFR3.

This distributed approach has to deal with the drawback of depending on a remote server. Although Fowler [2003, p. 91] states separation of vendor differences as one reason when to distribute logic to outside processes, it is often less stable than direct method calls. ΘPAD aims at addressing this concern by using the remote server for calculation only and not storing session data on it.

Connection to $\mathcal{R}$

As described in Section 4.2, Rserve is used to connect to $\mathcal{R}$. The Java library JRclient wraps this connection and lets ΘPAD deal with Java classes. It transforms methods to direct calls on $\mathcal{R}$ through the offered interfaces. However, since forecasting algorithms have different parameters, there is an additional layer of abstraction needed.

To address that, `TSLib` employs a *strategy pattern* [Gamma et al. 1995, p. 315] that includes an interface for forecasting: `IForecaster`. The main purpose is the method `forecast` that returns an object of interface type `IForecastResult`. Figure 4.9 shows how the inheritance in UML.

With this structure, forecasting algorithms only have to implement the interface `IForecaster` and be declared in the enum `ForecastMethod` as shown in Figure 4.12. Hence, they are configurable in the aspect's YAML file and can be used for anomaly detection inside ΘPAD. Figure 4.10 visualizes the calls made to use forecasting algorithms in an UML sequence diagram, Figure 4.11 shows the process in more detail.

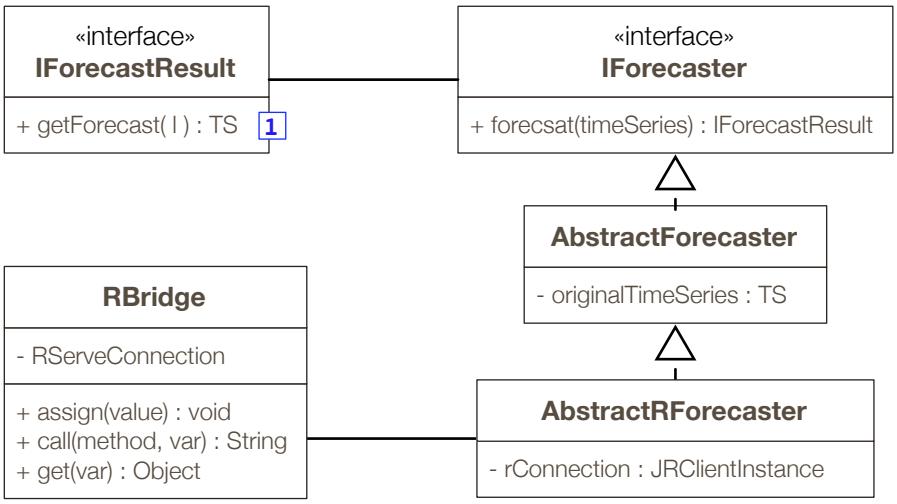

Figure 4.9: IForecaster and IForecastResult are interfaces used for applications using `TSLib`. The `AbstractRForecaster` is a wrapper with convenience methods to call forecasting packages in $\mathcal{R}$. 1 is the abbreviation of the type `TimeSeries`.

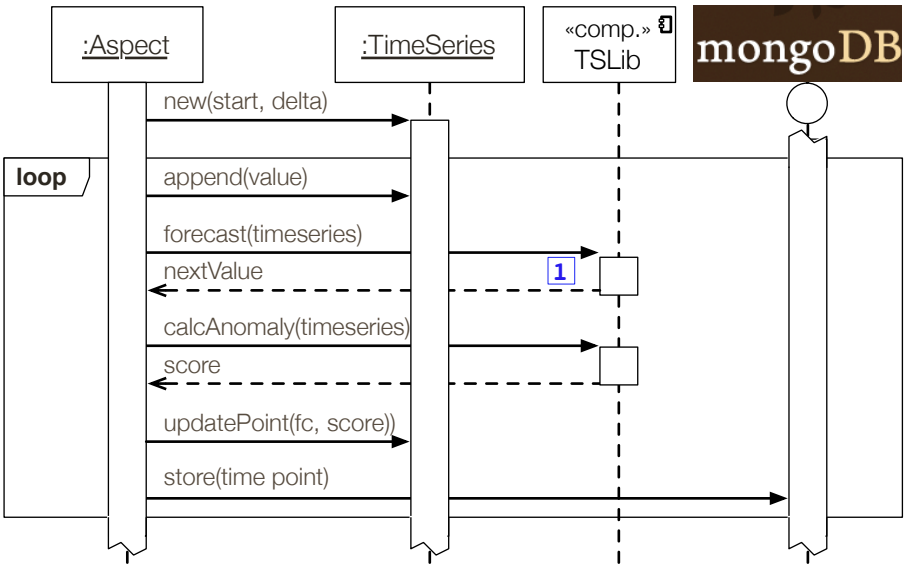

Figure 4.10: A coarse-grained view of the `Aspect` class using time series objects and the forecasting provided by `TSLib`. After every anomaly score calculation, the result gets persisted. The marker at 1 refers to a more detailed view in Figure 4.11

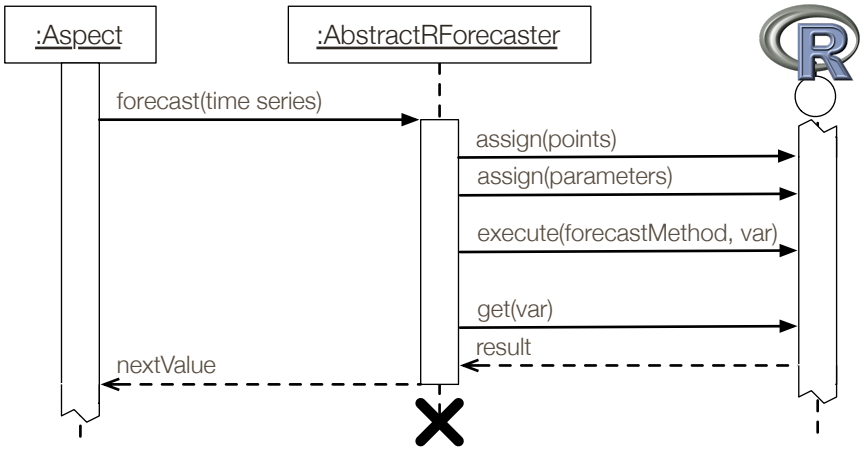

Figure 4.11: The abstract class `AbstractRForecaster` wraps the call to the JRclient components and thus performs algorithm calls on the Rserve instance.

Algorithms used

Θ**PAD** needs a selection of forecasting algorithms predefined to address the anomaly detection of the case study FC1. If this selection fits that particular environment, it can be assumed that Θ**PAD** serves for other software as well having related performance characteristics.

Frotscher [2011, p. 7] evaluated several forecasting algorithms and compared them in the field of performance monitoring. Θ**PAD** uses these recommendations to serve prepackaged forecasting algorithms that are most likely to be needed. Most algorithms used in Θ**PAD** work toward detecting contextual anomalies, as classified in Section 2.3.1. The mathematic foundations were described previously in Section 2.2.3.

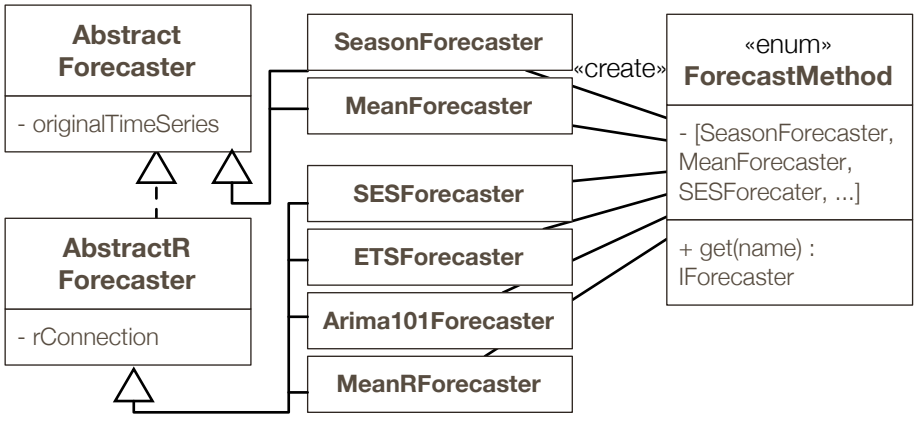

Figure 4.12: The available forecast methods of Θ**PAD** inherit from different abstract forecasters depending on whether they use $\mathcal{R}$.

The name of every algorithm is declared in the enumeration `ForecastMethod`. Thus, the instantiation of the configured algorithm is done automatically according to the aspect's YAML configuration. Figure 4.12 lists the available algorithms and shows the class inheritance.

MeanForecaster

The mean of a sliding window is calculated on every update of the time series. The green line chart denotes the forecasted values as an overlay on the real measurements denoted with the gray bars (Figure 4.13).

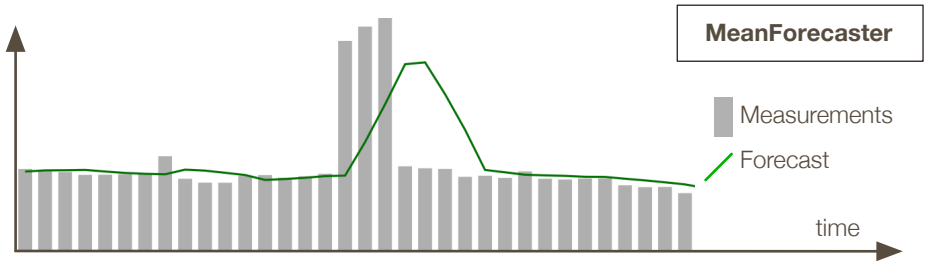

Figure 4.13: MeanForecaster example output for time series X with $\Delta_X = 1$h and a forecasting window W with $D_W = 3$h in an observed time span of 5h.

SeasonForecaster

Figure 4.14 illustrates how the *SeasonForecaster* needs a complete forecasting window of $D_W = 24$h to produce any output. As the next forecasted value, it takes the value that was measured 24h ago at the same hour. Due to this simplicity, this algorithm is directly implemented in Java.

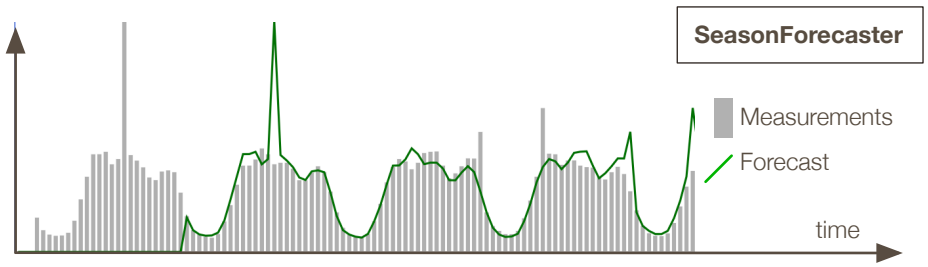

Figure 4.14: SeasonForecaster example output for time series X with $\Delta_X = 1$h and a forecasting window W with $D_W = 24$h in a course of 4.5 days.

ETSForecaster

Figure 4.15 shows the output of the `ETSForecaster` with the corresponding call of the `forecast` function in $\mathcal{R}$ in Listing 4.4.

```
1  forecast(ets(input, alpha=0.9), h=1)
```

Listing 4.4: SES forecast method in $\mathcal{R}$

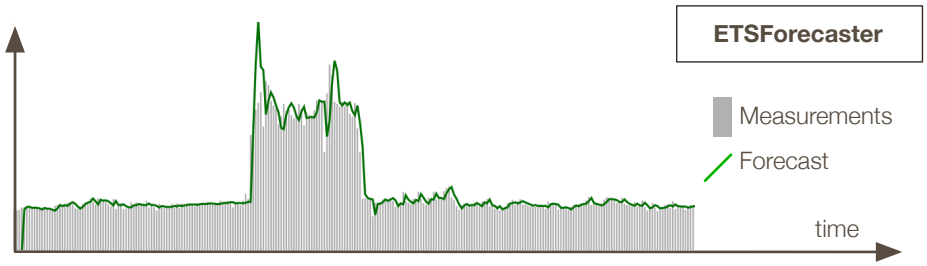

Figure 4.15: `ETSForecaster` example output for time series X with $\Delta_X = 1$h and a forecasting window W with $D_W = 3$h in an observed time span of 2h.

SESForecaster

Figure 4.16 shows the output of the SES forecaster with the according $\mathcal{R}$ code given in Listing 4.5.

```
1  ses <- HoltWinters(points, beta = FALSE, gamma = FALSE)
```

Listing 4.5: SES forecast method in $\mathcal{R}$

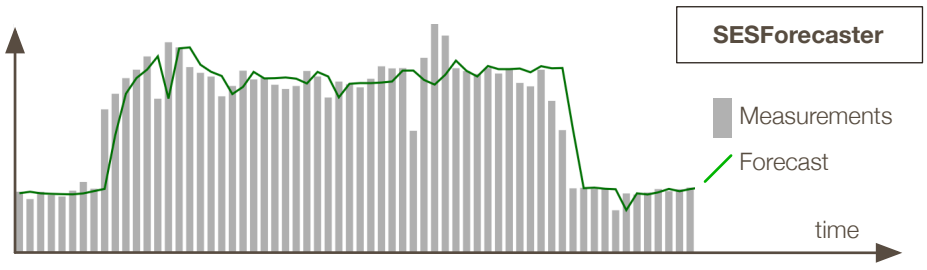

Figure 4.16: SESForecaster example output for time series X with $\Delta_X = 1$min and a forecasting window W with $D_W = 15$min in an observed time span of 10min.

ARIMAForecaster

Figure 4.17 shows the output of the `ARIMA(1,0,1)` algorithm as defined in Section 2.2.3.

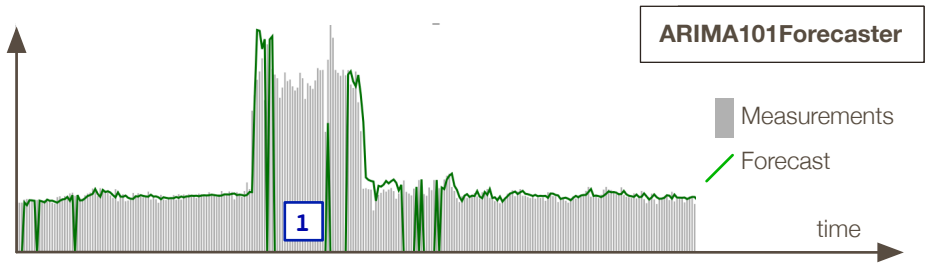

Figure 4.17: ARIMAForecaster example output for time series X with $\Delta_X = 1\text{min}$. It uses a forecasting window W with $D_W = 1\text{h}$ in an observed time span of 2h and applies the automatic model detection of the `forecast` package. At some points, for instance at 1, a model could not be detected automatically, thus yielding null values.

Anomaly Score Calculation Methods

The anomaly detection is based on the euclidean distance function described in Section 2.3.4. To keep it even simpler, the difference is divided by the sum of actual value and forecast. A snippet of the Java code is given in Listing 4.6.

```
1  double difference = nextpredicted - measuredValue;
2  double sum = nextpredicted + measuredValue;
3
4  double score = Math.abs( difference / sum );
```

Listing 4.6: Java code to calculate anomaly scores

4.6 Anomaly Score Interpretation

This section addresses the functional requirements FR4 'Interpretation and Action' and corresponds to activity D of the Θ**PAD** approach (see Section 3.6) that defines how anomaly scores are processed. Section 4.6.1 explains the usage of message queues for this purpose.

Further on, Section 4.6.2 gives details on the persistent storage of the extracted time series and how they are accessed and evaluated graphically (Section 4.6.3).

4.6.1 Anomaly Score Queue

Requirement FR4 defines the necessity of dealing with anomaly scores, which exceed the defined threshold. At the time of detection, immediate action has to be

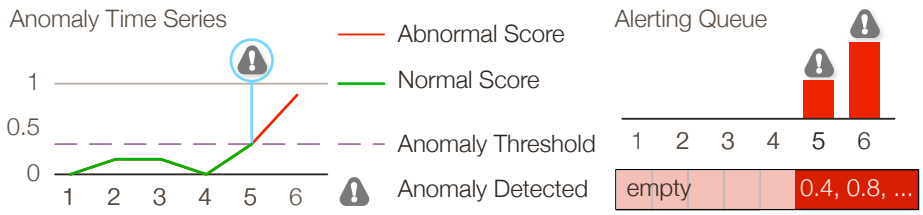

Figure 4.18: From time point 1 to 4 the alerting queue is empty. The first value was produced when the anomaly score exceeded the threshold of 0.4. From this point on, anomaly scores are propagated to the anomaly queue depicted in AMQP notation.

taken in order to fix the problem that has caused the anomaly to occur. Numerous alerting means exist such as *Short Message Service (SMS)* gateways, automatic emails, or phone calls. As of requirement FR4, ΘPAD has to be configurable, with the first setup being in the case-study environment (FC1).

To address all these demands, the implementation does not include a special kind of alerting, it rather uses AMQP as a standard to report alerts. So, the environment can employ its own alerting by connecting a consumer to the anomaly queue.

For normal behavior, nothing is published. Whenever the behavior is classified as abnormal, the anomaly score is put on to the queue. For instance, a connected SMS gateway could wake up administrators at night and provide the anomaly score in the alerting text message. Figure 4.18 shows an anomaly time series with the according messages on the anomaly queue.

4.6.2 Time Series Storage

To enable startup and shutdown processes (NFR2), ΘPAD persists its internal state to preserve it after shutdown. When starting the server, it loads the previous data and builds up the time series objects in memory.

As of Section 4.2, MongoDB is required by the server running ΘPAD. This document store database is used to hold values for each time point. In addition to the anomaly score, the calculation's basis, the measurement and the forecasted value are stored. This helps discussions about the origin of anomalies and is a requirement for the anomaly visualization described in the next section.

Since the storage format of MongoDB is related to the human-readable JSON (see Section 2.4.1), an example time point can be simplified listed in plain text in Listing 4.7. The key `anomaly` denotes the anomaly score calculated based on the forecasted value and the current measurement in a time series. Additionally, the aspect's `identifier` is included, specifying the aspect it was calculated from. This string was specified offline, at configuration time (see Section 3.3), for instance, by an administrator. It concatenates human-readable information such as the controller (page for the case study), the aggregation step size and the length of the forecasting window.

```
{
  "_id" : ObjectId( "4f09a7ee0364755651e4d9ba" ),
  "time" : Date( 459819300000 ),
  "measure" : 1245.434,
  "forecast" : 1249.112,
  "anomaly" : 0.00195,
  "identifier" : "root_page.D1min.L1h"
}
```

Listing 4.7: Time point stored as BSON in MongoDB

4.6.3 Graphical Evaluation

Post-mortem analysis and adjustments of the configuration are one requirement of the **ΘPAD** implementation. Good configuration of the aspects is crucial in order to define the best threshold at which anomalies are detected as such. Plots of measurement time series can support discussions about the reasons of anomalies. This method was classified as visualization based anomaly detection Section 2.3.1. Accordingly, visualized anomaly scores can help finding the best threshold and forecasting algorithm.

Plotting time series can be achieved by a variety of spreadsheet applications, statistical software and libraries including $\mathcal{R}$'s PDF output. However, navigating through time is rarely offered and there is[2] no software plotting the data of the time series storage (see Section 4.6.2) in an acceptable way.

To address this need, **ΘPAD** includes a web front end that accesses the time series storage, lets users navigate through time and displays the measurements alongside the forecasted values and calculated anomaly scores. Figure 4.19 shows a screenshot of how the Safari rowser displays the graph with HTML5 technologies. On the server side, the software can be installed directly on the monitoring system (see the UML deployment diagram in Figure 4.3) and administrators with access can inspect the visualization via web browsers.

The front end is built with the lightweight Sinatra web-framework [Günther 2010, p. 2]. The D3 library for graphical grammar written by Bostock et al. [2011, p. 8] helps rendering the custom time series data into an image using the portable [Eisenberg 2002, p. 6] SVG format. With this library, even a long time span with many measurements can be navigated performant. The user can choose the display time frame and the configured aspects freely, which makes navigation and evaluation of different aspects is possible. At every change, new data is loaded from MongoDB and rendered in the browser via Ajax without reloading the page.

[2] As of the implementation period in 2011

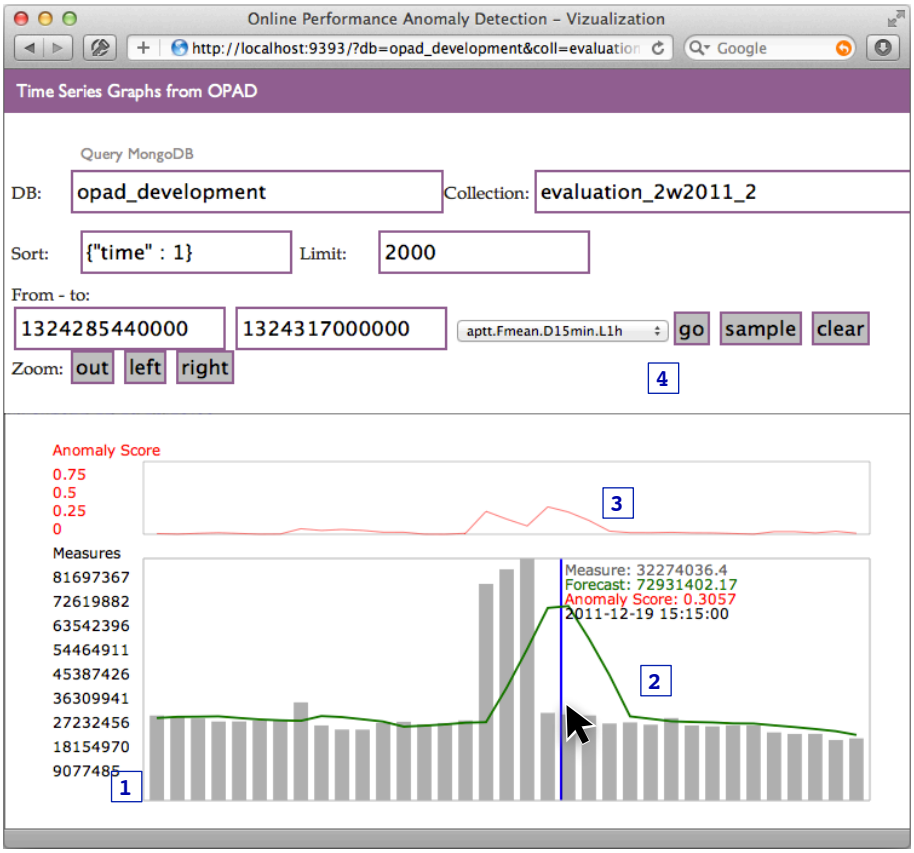

Figure 4.19: Screenshot of the web frontend provided for post-mortem analyses. The HTML5-capable Safari rowser is used to access the interface and draw the time series on a canvas element holding an SVG [Hickson 2012, Section 4.8.11]. The bars at 1 visualize aggregated measurements, 2 the forecast, and 3 depicts the calculated anomaly score. A control panel at 4 offers navigation through time and selection of the database and aspect.

Chapter 5

Evaluation

The thesis proposal of 2011 stated four goals that defined the design of 'Online Performance Anomaly Detection' and its setup for a case study. **T4**, the thesis' evaluation goal stated in Section 1.2, informally lists the research questions that should be covered in an evaluation phase as follows:

- **RQ1**: Can anomalies be detected automatically?
- **RQ2**: How good does automatic detection compare against manual detection?
- **RQ3**: Is the software implementation usable in a long-term test in practice?
- **RQ4**: Which factors can cause anomalies?

These questions can be refined and be directed to goals. Subsequently, metrics are needed to answer the goals reliably and to determine the way, how the measures of an experiment have to be quantified. The *Goal Question Metric Paradigm (GQM)* is a framework for "defining measurable goals" suggested by the inventors [Basili et al. 1994, p. 528]. Its approach defines the contents of the following Section 5.1 and is as follows:

1. Define **g**oals on the objects being researched.
2. Pose **q**uestions that support reaching the goals.
3. Set **m**etrics to answer the questions subjectively or objectively.

In order to answer the research questions, a long-term test is set up in the case-study environment (Section 5.2). The gathered data and observations are described in Section 5.3. Finally, Section 5.4 analyzes the observations with respect to the metrics. According to the GQM, the results are hence used to quantify how far the goals were reached.

5.1 Evaluation Goals

As the acronym indicates, GQM defines three essential entities that have to be specified in order to conduct and interpret engineering phenomena: **G**oal, **Q**uestion and **M**etric. Figure 5.1 depicts the steps of this evaluation phase following the GQM hierarchy.

For goals, Basili et al. [1994, p. 528] state three objects ob measurement: *process*, *product* or *resource*. The goals **G1**, **G2** and **G3** adhere to this definition and are also defined on distinguished objects. A goal is based on four *dimensions*: *purpose*, *issue*, *object* and *viewpoint*. The following Tables (5.1 to 5.3) adhere to the GQM standard. As the hierarchy of Figure 5.1 illustrates, some questions share the same metrics and are purposely combined.

G1: Assess Practicality of Approach

The process of online performance anomaly detection is the central use case of the Θ**PAD** implementation. This goal addresses the evaluation of how the anomaly detection can be achieved in general.

Goal	**G1**	
	Purpose	Assess
	Issue	the practicability of
	Object (process)	performance anomaly detection
	Viewpoint	on an online, automatic basis
Question	**Q1.1**	Is Θ**PAD**'s server stable?
Metrics	**M1**	server uptime
	M2	CPU utilization
Questions	**Q1.2**	How precise is the detection?
	Q1.3	How accurate is the detection?
Metrics	**M3**	Number of true positives
	M4	Number of false negatives
	M5	Number of actual anomalies

Table 5.1: Definition of **G1**: "Assess the practicability of performance anomaly detection on an online, automatic basis"

G2: Find ΘPAD Configuration for XING

This goal is defined on Θ**PAD** as a `product` and directly influences **T3** "Θ**PAD** Integration with Case-Study System". If this goal is reached, a fitting configuration is found that collects the performance data from the case-study system's architecture. However, this does not tell if the anomaly detection itself can compete against manual detection.

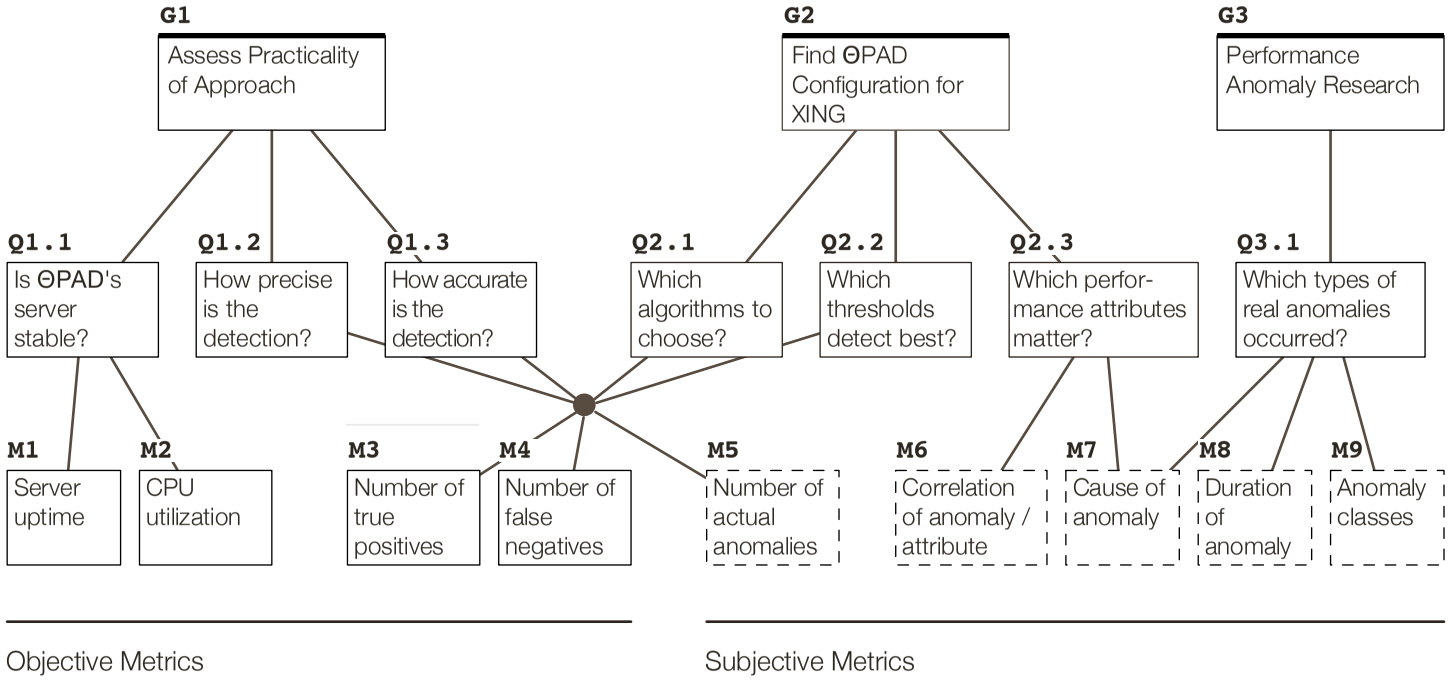

Figure 5.1: Evaluation goals with according questions and metrics. The hierarchy follows the GQM paradigm of Basili et al. [1994, p. 529]. Subjective Metrics do not only depend on a qualitative measure but also on the viewpoint. **M3** to **M5** are shared amongst multiple questions as denoted by the dot.

Goal	G2	
	Purpose	Find the best
	Issue	configuration setup of
	Object (product)	ΘPAD
	Viewpoint	for the case study
Questions	Q2.1	Which algorithms to choose?
	Q2.2	Which thresholds detect best?
Metrics	M3	Number of true positives
	M4	Number of false negatives
	M5	Number of actual anomalies
Question	Q2.3	Which performance attributes matter?
Metrics	M6	Correlation between anomaly and attribute
	M7	Cause of anomaly
	M8	Duration of anomaly
	M9	Anomaly classes

Table 5.2: Definition of G2: "Find the best configuration setup of ΘPAD for the case study"

G3: Performance Anomaly Research

Since this thesis addresses the detection of performance anomalies, the collected data can be used to research the performance behavior of large-scale software systems such as in the case study. Thus, this goal is defined on the resource *software system* and poses questions about the occurred anomalies, their duration and the reasons as such.

Goal	G3	
	Purpose	Learn which
	Issue	anomalies occur with
	Object (product)	software system
	Viewpoint	in large-scale architectures
Questions	Q3.1	Which types of real anomalies occurred?
Metrics	M7	Causes of anomalies
	M8	Duration of anomalies
	M9	Anomaly classes

Table 5.3: Definition of G3: "Learn which anomalies occur with software system in large-scale architectures"

Metrics

Hayden [2010, p. 38] suggests, that research should only consider metrics supporting goals. The goal definitions in the previous section referred to certain metrics,

which will be defined in this section. According to GQM, objective and subjective metrics are distinguished as following:

Objective Metrics

This type includes all metrics that yields results when looking at the object under research only.

M1: **Server uptime**. For the specified runtime of the experiment, the *uptime* of the ΘPAD server is determined by the number of gap entries in the time series storage.

M2: **CPU utilization**. When all aspects are loaded, the CPU utilization is determined by using Unix's `top` command. This is done in a random test after all aspects were loaded.

M3: **Number of true positives**. According to Salfner et al. [2010, p. 8], every algorithm has to be checked against real observations. Subsequently, the classification as depicted in Figure 2.15 can be applied to count the true positives (TP).

M4: **Number of false negatives**. Similar to **M1**, the number false negatives (FN) is part of the output of a particular anomaly detection algorithm that got configured with a threshold and run on a time series with knowledge of the actual anomalies.

Subjective Metrics

This type is influenced by the measured object and the viewpoint. In order to gather data on these metrics, qualified interviews are held and manual detection of anomalies was performed in a post-mortem way:

1. The post-mortem analysis graph of Section 4.6.3 serves as a means to find anomalies in a certain time period.
2. For every anomaly found the duration is read by using the time picker provided by the tool. In case an anomaly grows steadily, the first point of increase is taken.
3. The reasons for the specific anomaly is discussed in a qualified interview with an software architect and an administrator responsible for the IT infrastructure.

For every anomaly the start and end times are captured along with the reasons and any other system information that can be gathered such as the Munin graphs. Following this schema, these metrics can be extracted from the human observation of performance graphs:

- **M5**: Number of actual anomalies
- **M7**: Duration of anomalies
- **M8**: Causes of anomalies

Based on the findings, two more metrics could be defined by further processing and correlating the results as follows:

M6: **Correlation between anomaly and attribute**. After finding the anomalies, multiple attributes can be taken into account for finding possible correlations of the anomalies.

M9: **Anomaly Classes**. Distinguishing properties amongst anomaly observations are searched in order to perform clustering.

5.2 Experiment Setup

Before the evaluation phase, the software implementation of Chapter 4 had to be tested in order to ensure the detection phase was stable and uninterrupted. Subsequently, the aspects configuration of the case study's environment (see Section 2.4.4) had to defined. Following the GQM approach used for the evaluation, the setup has to be done focused on specific goals [Basili et al. 1994, p. 1]. These decisions are explained in Section 5.2.1.

Running the detection phase itself required a time frame as long as possible after having finished and tested the Θ**PAD** implementation in the preceding step. Shortly after the software tests, the evaluation phase was planned and production data was gathered in the following time frame:

Began: December 18, 2011 10:00pm **Ended:** December 30, 2011 10:55pm

These 12 day were good fit since they included the Christmas holidays 2011 as well as 10 business days. We expected a high number of offices closing between Christmas and New Years, leading users to access XING for both private and business purposes. Thus it was likely that we would observe the platform to be under lower but varying *workload intensity* than in normal months.

For the *server* running Θ**PAD**, XING contributed a dedicated MacbookPro (2GHz Intel Core i7, 4GB 1333 MHz DDR3 RAM) and gave authorization to the importer queue. In order to set the right attributes as measures, an interview with Stefan Kaes was done prior to the evaluation phase. This step set up the basic configuration of Θ**PAD** in order to focus on the evaluation's goals. The following section gives insight on the decisions made in this interview.

After this time frame a simulated post-mortem analysis was planned in order to conduct the manual anomaly detection. This interview required a person with deeper knowledge of the architecture, preferably an administrator or architect maintaining the case study's system. Ergo, mainly data of the subjective data would be gathered by manual anomaly detection. Anticipated was the finding of significant anomalies (**M5**), their duration (**M7**) and possibly the causes (**M8**).

5.2.1 Setup Interview

The Aspects, Θ**PAD**'s basic setup entities, have to be declared in the software. XING's importer delivers a continuous stream of measurements aggregated from different sources of the architecture described in Section 2.4.4. Since every queued *event* gives a JSON object of various measured attributes (Figure 2.6) this phase

had to decide on which attribute to focus and how to configure the aspects accordingly.

Together with Stefan Kaes the performance data of Logjam in November and December 2011 was analyzed. Based on the discovered anomalies the following decisions about the configuration for the evaluation phase were made:

- **Page:** As described in Section 2.4.4, pages separate XING's architecture. In order to gather data of all parts of the application Logjam's *catch-all* identifier `all_pages` was selected to use.
- **Attribute:** Variations in performance could be seen best in the aggregation of all measures. Additionally, the focus on only one attribute promised a better comparison after the evaluation. So, for every aspect the attribute `total_time` was chosen.
- **Step size (Δ):** 5 minutes to 1 hour, depending on the forecasting algorithm.
- **Forecasting algorithms:** The Θ**PAD** software provided 5 different algorithms that all had to be tested: `SeasonForecaster`, `MeanForecaster`, `SESForecaster`, `ETSForecaster`, `Arima101Forecaster`.
- **Forecasting windows:** 1 hour to 7 days, depending on the forecasting algorithm.
- **Anomaly score calculation:** The `SimpleAnomalyCalculator` following Equation 2.13 was chosen in order to provide comparable anomaly scores.

5.2.2 Configuration

These decisions predefined the principles of the precise configuration. However, for step size, forecasting algorithm and window, only ranges were defined. Therefore, the aspect configuration was set as a meaningful combination of these variables in order to evaluate a significant number of possible anomaly detection settings. The threshold was left open and evaluated later. This was possible since this number could be changed iteratively while replaying the data (see the concept of time series storage in Section 3.6) to conduct qualitative analysis to find the values that fit best.

Table 5.4 shows the configuration with abbreviated names. Let `all::tt` be the measure `total_time` of page `all_pages`. Time spans are denoted by `min` (minute), `h` (hour), and `d` (day). The identifier in the last column is used to store the aspect's reference in the time series storage according to Section 4.6.2.

5.2.3 Evaluation Procedure

As stated in the previous section, the threshold was left undefined since there was no experience to choose any particular value beforehand. So, the time series data was gathered in the MongoDB to be replayed by an evaluation script in order to test out the best threshold. Since some metrics depend on the threshold, they had to be evaluated later. In order to build a coherent evaluation procedure, the metric quantification and data refinement was planned in steps as shown in Figure 5.2.

Measure	Δ	λ_{fc}	D_W	Identifier
all::tt	5min	Mean	1h	aptt.Fmean.D5min.L1h
all::tt	15min	Mean	1h	aptt.Fmean.D15min.L1h
all::tt	1min	Mean	1h	aptt.Fmean.D1min.L1h
all::tt	30min	Mean	3h	aptt.Fmean.D30min.L3h
all::tt	1h	Mean	3h	aptt.Fmean.D1h.L3h
all::tt	20min	Mean	2h	aptt.Fmean.D20min.L2h
all::tt	1min	–	1h	aptt.Fnone.D1min.L1h 1
all::tt	1min	ETS	1h	aptt.Fets.D1min.L1h
all::tt	1min	Arima101	1h	aptt.Farima101.D1min.L1h
all::tt	1min	SES	15min	aptt.Fses.D1min.L15min
all::tt	1min	SES	1h	aptt.Fses.D1min.L1h
all::tt	1h	WS	24h	aptt.Fses.D1h.L24h 2
all::tt	1h	WS	24h	aptt.Fws.D1h.L24h
all::tt	15min	WS	24h	aptt.Fws.D15min.L24h
all::tt	15min	WS	7d	aptt.Fws.D15min.L7d

Table 5.4: This configuration table lists all 13 aspects used in the experiment. The aspect at 1 does not use a forecaster and is used to replay the data only. 'WS', at 2 is an abbreviation of the `SeasonForecaster` as in Figure 4.14.

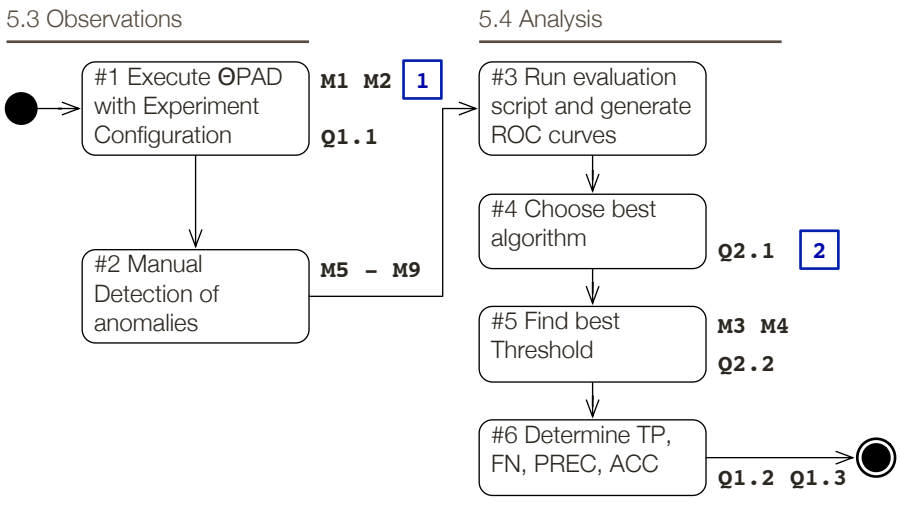

Figure 5.2: The activities performed throughout Section 5.3 and 5.4. 1 denotes the metrics **M1** and **M2** being quantified after step **#1**. 2 hints to the question that can be answered after the attached step.

5.3 Observations

As defined in the evaluation plan (see Figure 5.2), the observations include running the experiment (**#1**) and gathering statistics of the produced data. Subsequently, the manual detection (**#2**) is performed in an interview. Additionally, this interview is expected to reveal the causes of the anomalies that occurred in the experiment period.

#1: Execute ΘPAD with Experiment Configuration (M1, M2, Q2.1)

Section 5.2 defined the setup of the experiment, which was conducted on a dedicated OSX machine. This **ΘPAD** server gathers data from the Logjam importer. All output was stored as time series in a local MongoDB that grew up to several megabytes. In detail, the statistics of this experiment run were as following.

- **Time series storage:** Following statistic were drawn from the MongoDB:
  - `count`: 27942 entries, as described mathematically in Equation 3.8.
  - `size`: 3,460 KB (3,386 KB BSON database dump file)
  - `avgObjSize` 0.1238 KB
- **Server uptime (M1)**: **ΘPAD**'s uptime was 100% since the server never restarted and no gaps in the time series storage were found.
- **CPU utilization (M2)**: The output of the UNIX `top` command yielded nearly constant values after loading all aspects and listening on the Logjam importer queue:
  - **Rserv-bin.so**: 0.0% - 0.5%
  - **java**: 7.0% - 9.0%
  - **mongod**: 0.8% - 1.2%

These statistics lead to the answer of **Q1.1** attributing the **ΘPAD** implementation as a server stable enough for this aspect configuration gathering production data.

#2: Manual Detection of Anomalies (M5 - M9)

According to the experiment setup, the post-mortem interview was made with Stefan Kaes and Mike Adolphs, an administrator, in separate sessions. In these interviews, the causes of anomalies and their correlation to other measure attributes were discussed. The attributes taken into account for correlation were as follows.

- `db_time`: Response time of the database servers.
- `api_time`: Response time of the servers working on the databases and providing APIs for application servers.
- `count`: The number of page request. Whenever this metric is 0, it is possible for the network to be detached from Logjam.
- `view_time`: Time needed to produce HTML on the application server.

In these sessions, eight anomalies became clearly visible, thus quantifying **M5**. The time, duration and possible correlation (**M7**) were listed in Table 5.5. Based on other measures from Logjam and Munin, the possible reasons were discussed and the subjective metric **M7** quantified. Using the attribute `total_time` for anomaly

#	Begin	End	Duration	Correlated attribute	Fig.
A1	Dec. 19 14:05	Dec. 19 14:46	00:41	`count`	5.3
A2	Dec. 21 13:47	Dec. 21 15:09	01:22	`db_time`	5.4
A3	Dec. 21 23:08	Dec. 21 23:20	00:12	`count`	5.4
A4	Dec. 22 09:33	Dec. 22 09:35	00:02	`db_time`	5.5
A5	Dec. 23 13:04	Dec. 23 13:06	00:02	`view_time`	5.6
A6	Dec. 27 09:45	Dec. 27 10:15	00:30	`count`	5.7
A7	Dec. 29 15:08	Dec. 29 15:31	00:23	`api_time`	5.8
A8	Dec. 30 01:48	Dec. 30 02:09	00:21	`db+api_time`	5.9

Table 5.5: Anomalies detected manually by the administrators.

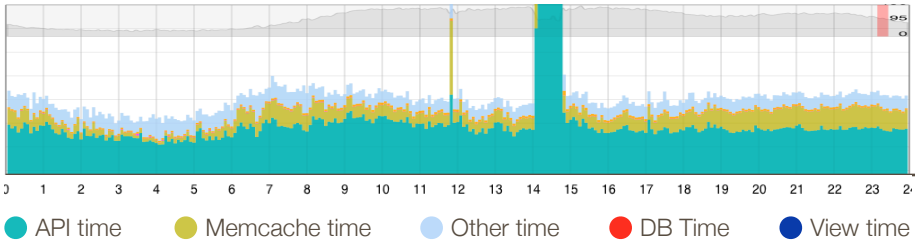

Figure 5.3: `A1`: Dec. 19 14:05 - Dec. 19 14:46
Due to a local network outtage, Logjam was disconnected from the data center.

detection is an answer to **`Q2.3`** since all causes affected the response time the users being an aggregation of all internal response times.

`M6`: **Correlation between anomaly and attribute**. After finding the anomalies, multiple attributes can be taken into account for finding possible correlations to the anomalies. Every time reasons of and, the correlation to a certain attribute could be made, as listed in Table 5.5. Figure 5.9 shows the most significant example of the database that was accessed by the application servers through the REST API. The correlation `db_time` to `api_time` could be made.

`M9`: **Anomaly classes**. Distinguishing properties amongst the findings are searched in order to perform clustering. Essentially, these types could be distinguished qualitatively.

- Complete disconnection of the live network from the network that hold the Logjam server: **`A1`**, **`A6`**, **`A3`**
- Rapid increase of the `total_time` attribute caused by a software or a server fault. The effect is users waiting for the page to load: **`A2`**, **`A4`**
- Increased page load time due to software errors or instructions with worse runtime than before: **`A7`**
- Background tasks that appeared as anomalies but were actually planned: **`A8`**
- Miscellaneous or unknown reasons: **`A5`**

This makes it possible to answer **`Q3.1`**: Five types of anomalies occurred with most of them being caused by offline monitoring. However, eight anomalies were too few to run a clustering approach like the *k*-means [Wang et al. 2008, p. 3].

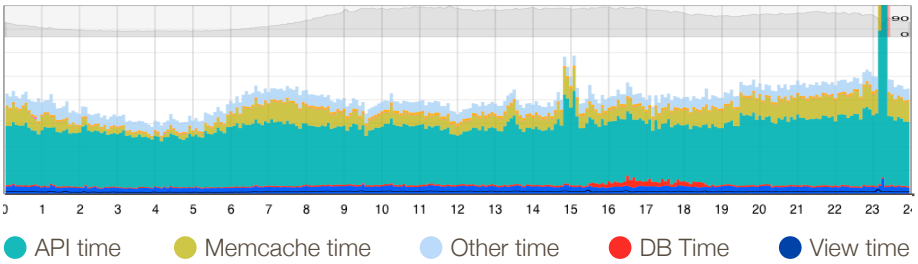

Figure 5.4: A2: Dec. 21 13:47 - Dec. 21 15:09
Erroneous code was deployed and immediately fixed by a live patch.
A3: Dec. 21 23:08 - Dec. 21 23:20
XING switched the cable company providing the bandwith between the data centers in Hamburg and Frankfurt. This caused some dependent API calls to respond slower. Correlation to attribute: `api_time`.

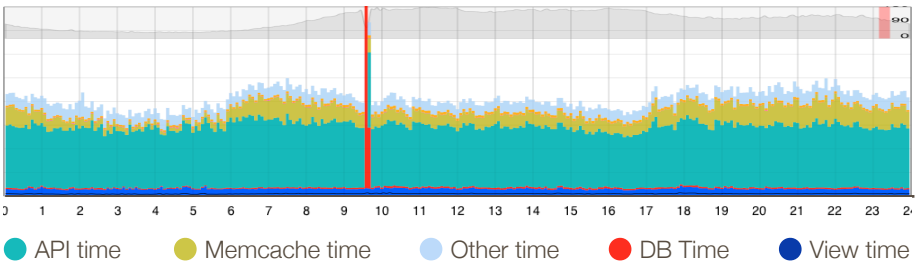

Figure 5.5: A4: Dec. 22 09:33 - Dec. 22 09:35
The hard disk of one database server crashed due to the consumption of too much memory.

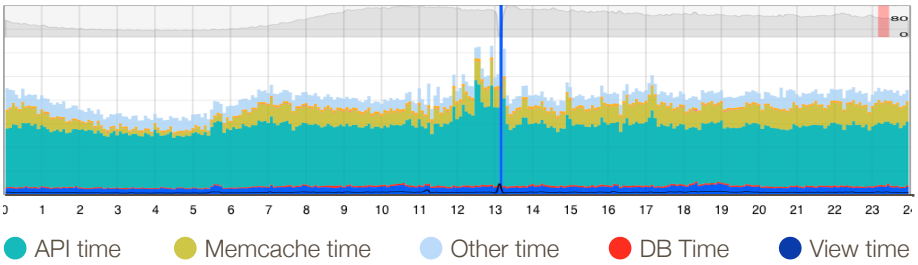

Figure 5.6: A5: Dec. 23 13:04 - Dec. 23 13:06
Reasons could not be determined.

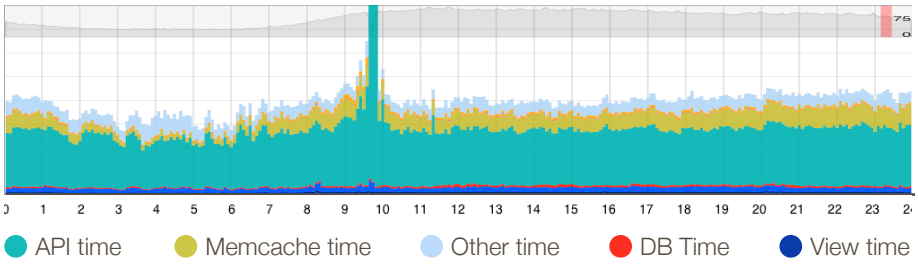

Figure 5.7: A6: Dec. 27 09:45 - Dec. 27 10:15
Logjam could not access the online system since the network of XING's office in Hamburg was offline.

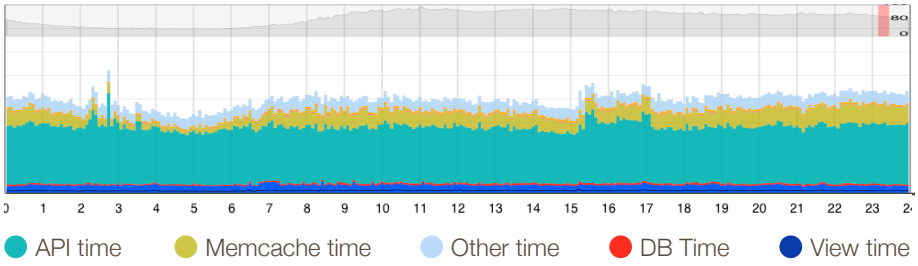

Figure 5.8: A7: Dec. 29 15:08 - Dec. 29 15:31
A Live patch in the Rails code caused the overall performance to decrease. Another live patch solved the problem.

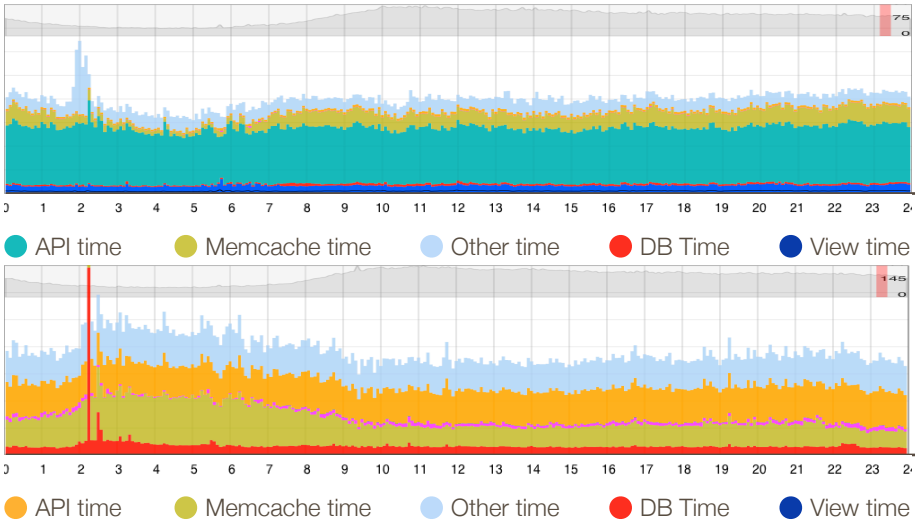

Figure 5.9: A8: Dec. 30 01:48 - Dec. 30 02:09
One backup server was down due to a scheduled backup. The lower graph shows the Perl backend slowing down due to this server down time. The db server's API response time, indicated by the red bars, rose to infinity when the server crashed.

Figure 5.10: Anomaly Score Graphs of December 19, 2011. 1 shows the sliding window of the `MeanForecaster`. 2 is the ARIMA101 algorithm that partly delivered null values. Since the forecast model is automatically determined at every discrete time point, it is sometimes unable to determine the configuration.

Aspect identifier	Data points	$max(\Psi)$
aptt.Fmean.D15min.L1h	1155	0.4
aptt.Fmean.D5min.L1h	3467	0.68
aptt.Fmean.D1min.L1h	17335	0.92
aptt.Fmean.D30min.L3h	581	0.49
aptt.Fmean.D1h.L3h	290	0.41
aptt.Fmean.D20min.L2h	869	0.45
aptt.Fets.D1min.L1h	271	0.24
aptt.Farima101.D1min.L1h	271	0.17
aptt.Fses.D1min.L15min	271	0.17
aptt.Fses.D1min.L1h	271	0.13
aptt.Fses.D1ws.L24h	290	1.0
aptt.Fws.D1h.L24h	290	1.0
aptt.Fws.D15min.L24h	1155	1.0
aptt.Fws.D15min.L7d	1155	1.0

Table 5.6: Observed statistics of the measured data mapped to every aspect. $max(\Psi)$ indicates the maximum of all anomaly scores calculated in order to preselect a fitting threshold θ.

5.4 Analysis

#3: Run Evaluation Script (M3, M4)

Since the threshold was left undefined in order to find the best value in this analysis step, an Ruby script was build that replayed several scenarios with different thresholds configured on every aspect used in the experiment. Figure 5.11 demonstrates how the algorithms starts with a threshold of 0 and increases its value up to 1 and performs the anomaly detection at every step.

This evaluation script was run on every aspect defined in the setup interview (see Section 5.2.1). Results hence got stored in spreadsheet files with Microsoft Excel[1] for comparison.

#4: Selecting the Best Aspect (Q2.1)

The data in the spreadsheets produced by the evaluation script were visualized in ROC curves. The most significant results are compared in Figure 5.12. Aspects using the SeasonForecaster were excluded in the first place due to results promised to be "misleading" [Box and Jenkins 1990, p. 301].

Aspect aptt.Fmean.D1min.L1h shows the greatest deviation from the line of random guess (see Figure 2.16 in Section 2.3.4). This aspect has the attribute total_time with a step size of $\Delta = 1min$ and uses the MeanForecaster with a sliding window of $D_W = 1h$ as shown in Table 5.4. Question **Q2.1**, "Which

[1] http://www.microsoft.com/mac/excel

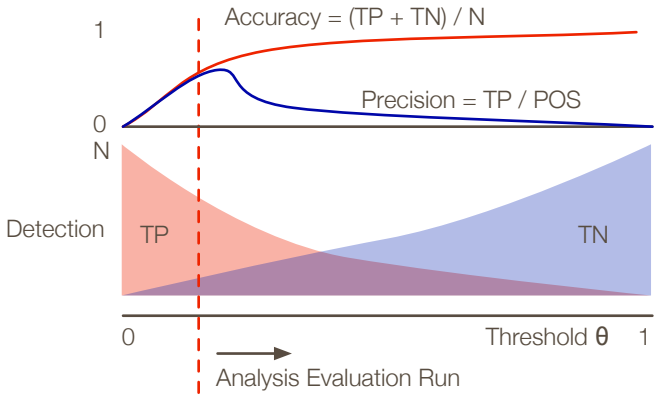

Figure 5.11: Execution of the analysis script that iterates over the range of thresholds θ in discrete steps.

θ	0.0	0.1	0.2	0.3	0.4	0.5	...	0.9	1.0
TP	221	104	70	41	14	0	...	0	0
FN	0	117	151	180	207	221	...	221	221
PREC	0.01	0.02	0.08	0.15	0.11	0.0	...	0.0	NaN
ACC	0.02	0.67	0.94	0.98	0.98	0.98	...	0.99	0.99

Table 5.7: Results of the evaluation script for identifier testing 11 values for the selected aspect `aptt.Fmean.D1min.L1h`
.

algorithms to choose?" can now be answered with `MeanForecaster`. In the next step, **#5**, possible for this algorithm are searched in order to answer **Q2.2**.

#5: Finding Thresholds for the Best Aspect (M3, M4, Q2.2)

Depending on the threshold, every aspect yields a range of true positives and false negatives. From these values, the metrics accuracy (ACC) and precision (PREC) can be calculated with the formulas defined in Section 2.3.4. These three dimensions (threshold, TPR, FNR) are listed in Table 5.7 and are plotted in Figure 5.14.

Based on this visualization, a tradeoff has to be made between following targets:

- **High precision**: A high ratio of the the actual anomalies were detected correctly.
- **High accuracy**: The algorithm often classifies correctly, either anomaly or normal behavior.

The decision on the preference of either value is influenced by various factors such as business goals, alerting infrastructure or administration guide lines. For this evaluation, the median of both peaks is taken to determine the threshold: $\theta = 0.23$ and according values of precision and accuracy.

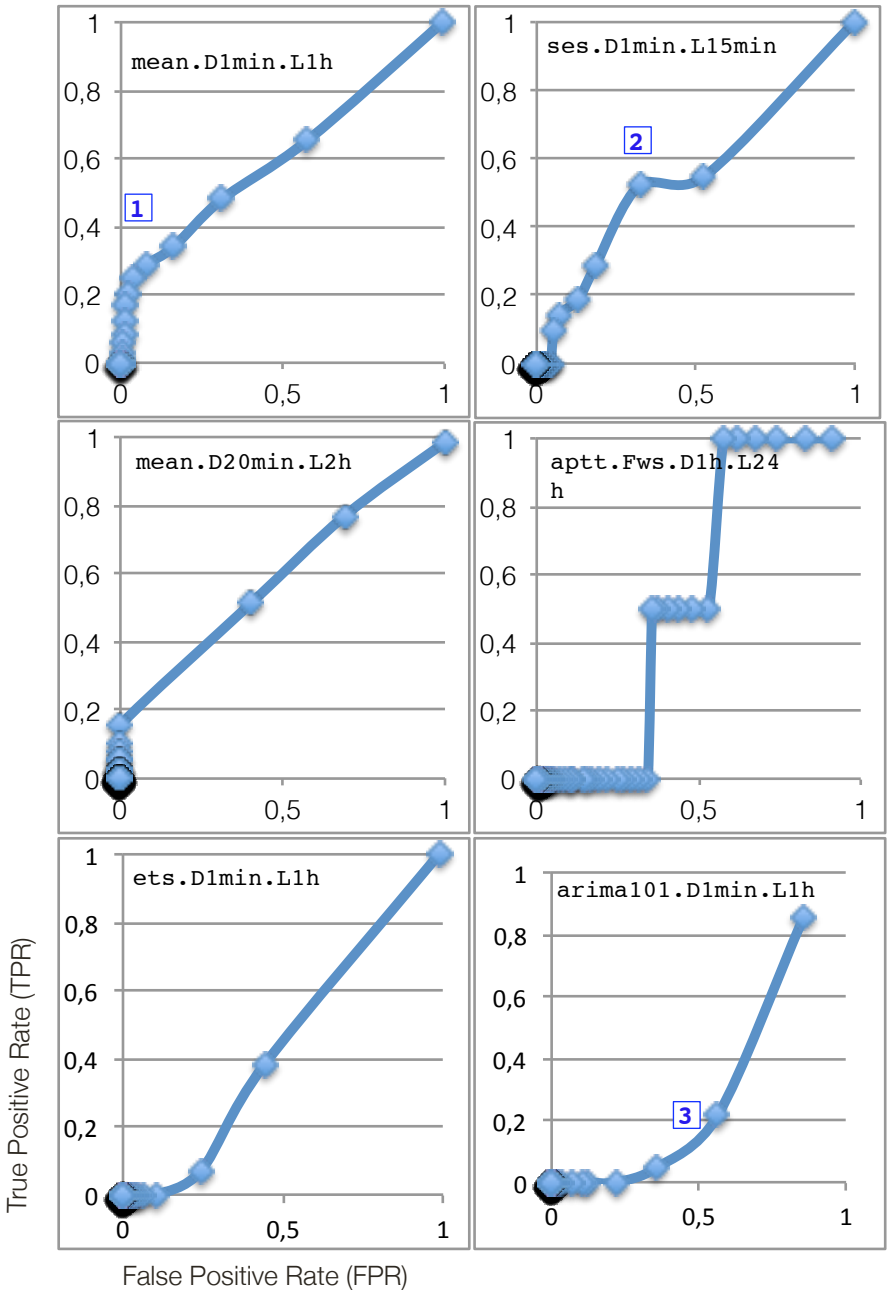

Figure 5.12: The comparison of all ROC curves show a preference on the `aptt.Fmean.D1min.L1h` aspect (1). The two examples at the bottom (e.g. 3) points to the ARIMA101 forecaster, which is unable to automatically detect the forecast model. Figure 5.10 showed that this correlates with the occurrence of anomalies, but is impractical to be evaluated with ROC curves. 2 indicates an improvement which would produce to a high false positive rate in general with one exceptional peak. That leads to the impression, that the behavior of the system under monitoring has to be well known in order to configure the aspects correctly. In this figure the keys identifying the previously defined aspects are shortened for readability by omitting the string `aptt.F`.

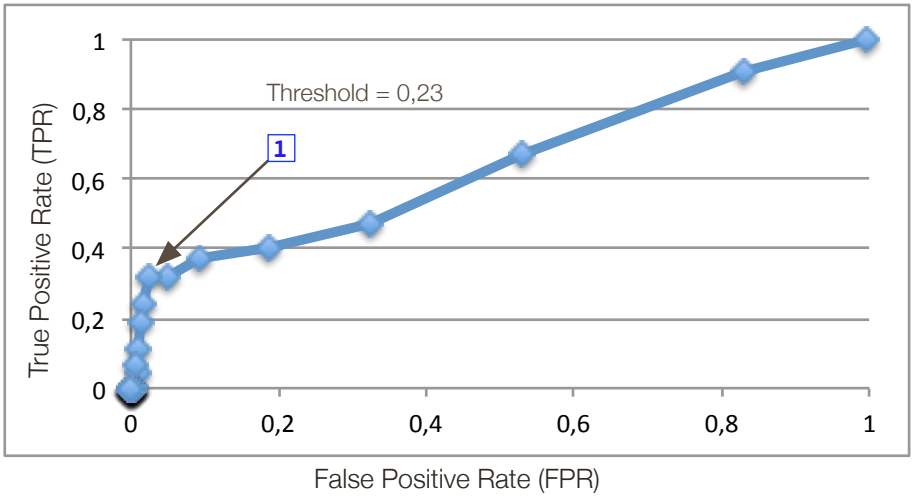

Figure 5.13: ROC curve of aspect `aptt.Fmean.D1min.L1h` tested on all eight anomalies in the experiment. 1 The threshold of 0.34 has the best tradeoff between TPR and FNR.

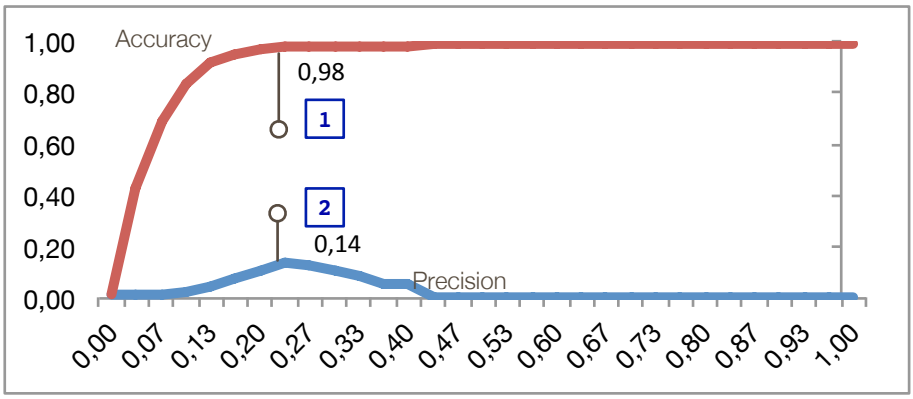

Figure 5.14: The selected aspect `aptt.Fmean.D1min.L1h` shows high accuracy (red line) but mediocre precision (blue line).

Goal	**G1**	Assess Practicality of Approach
Question	**Q1.1**	Is ΘPAD's server stable?
Answer	Yes, stable for production data. The server is proven to provide robustness over the experiment period.	
Question	**Q1.2**	How is the detection accuracy?
Answer	High, above 0.9 after a threshold of 0.1	
Question	**Q1.3**	How is the detection precision?
Answer	Low, 0.14 at best	

Table 5.8: Results of **G1**: "Assess the practicability of performance anomaly detection on an online, automatic basis"

#6: Determine TP, FN, PREC, ACC (Q1.2, Q1.3)

The choice of the best algorithm and threshold leads to answering of two questions of goal **G1** that indicate the practicability of the Θ**PAD** approach in general: Questions **Q1.2**: "How precise is the detection?" and **Q1.3** "How accurate is the detection?" are answered based on the formula of Salfner et al. [2010, p. 9] and the data of **M3**, **M4** and **M5**.

The spreadsheet data yielded finer-grained values than Table 5.7. For the given threshold of $\theta = 0.23$ 38 true positives, 231 false positives, $16,883$ true negatives and 183 false negatives were recorded. This lead to the calculation of $PREC = 0.14$ and $ACC = 0.97$ by the formula of Salfner et al. [2010, p. 9].

Based on these calculations, the goal **G1** "Assess Practicality of Approach" can be addressed by answering the two assigned questions:

- **Q1.2: "How precise is the detection?"** The low value of $PREC = 0.14$ indicates that every time an anomaly occurs, it is likely for the algorithm to ignore it.
- **Q1.3: "How accurate is the detection?"** The high accuracy ($ACC = 0.97$) gives confidence to any high anomaly score: Every time the threshold is reached, it is probably an anomaly.

GQM Result Interpretation

With the metrics being evaluated, the research questions **Q1.1** to **Q3.1** were answered with respect to the evaluation's goals. In the following, the GQM tables of Section 5.1 are filled with answers to summarize the results of the GQM approach.

Based on this analysis the next chapter will draw conclusions and give an overview of the lessons learned throughout this evaluation phase. The following section informally lists related work in research and practice and refers to software packages addressing related problems.

Goal	G2	Find Configuration for XING
Questions	Q2.1	What algorithms to choose?
Answer	The `MeanForecaster` was proven to work best	
Questions	Q2.2	Which thresholds detect best?
Answer	For the selected aspect, $\theta = 0.23$ was a good tradeoff.	
Question	Q2.3	Which performance attributes matter?
Answer	`total_time` since it aggregates all other attributes.	

Table 5.9: Results of G2: "Find the best configuration setup of Θ**PAD** for the case study"

Goal	G3	Performance Anomaly Research
Questions	Q3.1	Which types of real anomalies occurred?
Answer	Most of the times, the monitoring network was disconnected from the production system. Apart from that, other anomaly causes are code faults and server crashes.	

Table 5.10: Results of G3: "Learn which anomalies occur with software system in large-scale architectures"

5.5 Related Work

In the preceding sections, the Θ**PAD** approach was evaluated with respect to the general practicability and the case study. Additionally, anomalies found throughout the experiment gave insights on performance anomaly research. With these results, Θ**PAD** can be compared with other software of this problem field and related research can lead to further improvement.

To address this notion, related products and research have to be evaluated and the right questions asked. Figure 5.15 shows these question and the following metrics as coined by the GQM approach [Basili et al. 1994, p. 528]:

- M10 **Cost:** Setup and monthly cost
- M11 **Assessment:** Advantages and Disadvantages, without any comparison
- M12 **Classification:** The addressed problem field. If possible, the taxonomy in Figure 2.9 is used
- M13 **Prerequisites:** Input data for the algorithm and setup requirements
- M14 **Algorithms:** Any algorithms used for the approach
- M15 **Precision:** Any results of the approach on real data

5.5.1 Alternative Software Products

Following is a list of software products that also provide solutions in the field of online performance anomaly detection. Links to the according companies or projects

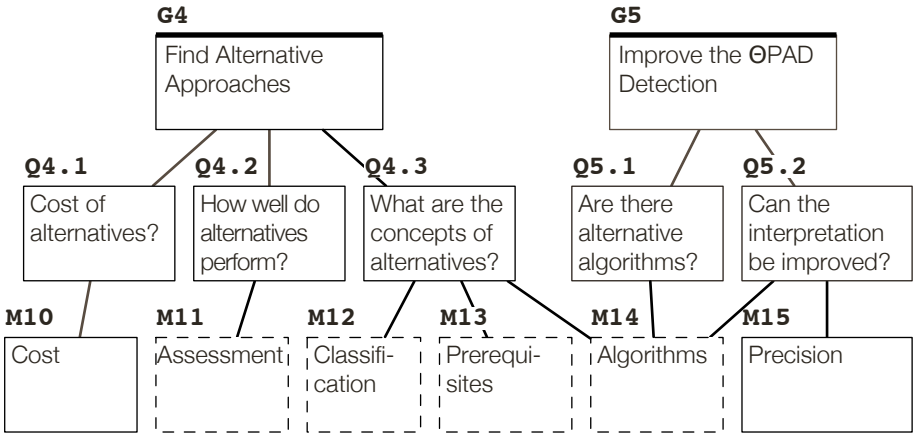

Figure 5.15: GQM hierarchy for the related work evaluation

are given in the glossary "Software Libraries and Products" on page 117. For abbreviation, most information is gathered from these according web pages and are hence not referenced in the bibliography.

Rackspace

Rackspace is a hosting company offering dedicated servers for enterprise. It advertises their services with included anomaly detection: "Our DDoS Mitigation offering analyzes your server's traffic patterns to learn about 'normal' network behavior" This normal behavior is then used as a reference model to compare against the current behavior.

- **M10 Cost:** Dedicated servers from $769 per month
- **M11 Assessment:** Monitoring is bound to their servers
- **M12 Classification:** System-Level anomaly detection
- **M13 Prerequisites:** Rackspace's anomaly detection uses metrics on network traffic as well as system-level measures

New Relic

New Relic is a performance monitoring service that performs its calculations on a centralized platform. This SaaS is able to detect abnormal behavior of large-scale systems based on application level metrics such as response time. Alerts are also triggered by comparing usage metrics against certain thresholds.

- **M10 Cost:** From $24 per month and server
- **M11 Assessment:** Many other features for applications and system-level monitoring. Since it is centrally hosted, application measurements are sent to their servers.
- **M12 Classification:** Performance and anomaly detection as an external service

Ourmon

According to its product page, RRDtool is a system for time series logging and visualization. It is widely used in industry and can be integrated in a variety of scripting languages to store data in this also-called *Round Robin Database*. Ourmon is a anomaly detection system relying in RRDtool. It captures input from system level and is mainly used for anomalies in network traffic.

- **M10 Cost:** Free, with an open source license.
- **M11 Assessment:** It is built to reduce noise in data and also measures data online.
- **M12 Classification:** Time Series, Botnet Detection
- **M13 Prerequisites:** Ethernet packets
- **M14 Algorithms:** Statistical algorithms
- **M15 Precision:** No information found

StatStream

Shasha and Zhu [2004, p. 103] used an architecture similar to ΘPAD's to aggregate raw data streams and produce time series digests in a system called StatStream.

- **M10 Cost:** Non-commercial
- **M11 Assessment:** It computes in near constant time the statistics
- **M12 Classification:** Time series analysis
- **M13 Prerequisites:** for multi-stream analysis problems, uses sliding windows like ΘPAD
- **M14 Algorithms:** Based on thresholds as well
- **M15 Precision:** No information found

Snort.AD

The *Network Instrusion Detection System (NIDS)* Snort is capable of running anomaly detection on network traffic. This software is *signature-based*, meaning that network traffic is matched against abnormal pattern. The anomaly detection of Snort also requires a proper adjustment on the domain and a learning phase as Bechtold and Heinlein [2004, p. 44] point out. A recent addition is the Snort.AD package providing an anomaly preprocessor.

- **M10 Cost:** Free, open source
- **M11 Assessment:** Widely used, - Only for network traffic. No sophisticated performance measures included
- **M12 Classification:** Network-Level
- **M13 Prerequisites:** Measures on system behavior
- **M14 Algorithms:** Pattern recognition
- **M15 Precision:** Due to the recent release date, no data was found

Resin Pro

Resin is a Java application server for enterprises. It can detect anomalies of its web traffic based on 'static analysis'. Like ΘPAD every metric can be used as input data for the detection. However, it is not an external tool and thus bound to the product.

- **M10 Cost:** One year subscription for enterprise support: $699
- **M11 Assessment:** Bound to Resin
- **M12 Classification:** Application Server
- **M13 Prerequisites:** Only for applications deployed in this server
- **M14 Algorithms:** Static analysis, like ΘPAD
- **M15 Precision:** No information found

5.5.2 Alternative Approaches in Research

Conditional Anomaly Detection (CAD)

Gathered measure data often include many attributes that can be used to perform anomaly detection on. Song et al. [2007, p. 2] found that often measurements recorded do not correlate with the actual anomalies. Their so-called 'CAD' algorithm learns these conditions from user input and excludes the anomalies that do not correlate with actual abnormal behavior.

- **M10 Cost:** - (Research)
- **M11 Assessment:** Detects relevant anomalies, filters anomalies that are irrelevant due to user input.
- **M12 Classification:** Interpretation, Data Mining
- **M13 Prerequisites:** Training with users
- **M14 Algorithms:** CAD algorithm
- **M15 Precision:** Not compared to metrics used in this work

PCAD

Rebbapragada et al. [2009, p. 1] state the fallacy in comparing two time series with each other. If this input data comes from multiple source or starting at time points that are not synchronized, a "phase-adjustment" has to be applied. Performing this step manually is costly. PCAD is their approach of using a modified *k*-means algorithm performing the preprocessing step.

- **M10 Cost:** - (Research)
- **M11 Assessment:** Works on high-dimensional and noisy data
- **M12 Classification:** Time series
- **M13 Prerequisites:** Periodic time-series data from multiple sources.
- **M14 Algorithms:** Pk-means, an improved version of *k*-means
- **M15 Precision:** Outperformed other algorithms when testing on astrophysical phenomena

Self-Adaptive Performance Monitoring for Component-Based Software Systems

In his PhD thesis, Ehlers [2012] covers fundamental questions regarding performance monitoring. Different anomaly detection approaches were developed, implemented, and evaluated directed towards self-adaption of software systems spread across components.

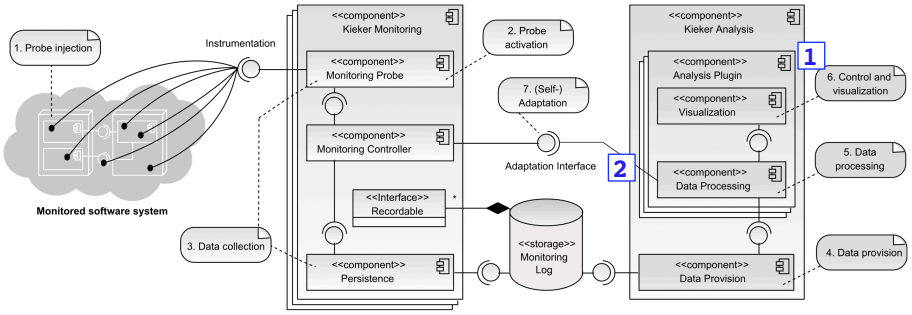

Figure 5.16: The architecture of the Self-Adaptive Performance Monitoring Approach uses Kieker as the underlying framework for performance monitoring and dynamic analysis. It is implemented as an analysis plugin (1) like ΘPAD. The monitoring is controlled via a *control feedback cycle* between the plugin and the monitoring component at 2 [Ehlers and Hasselbring 2011, p. 3].

Since large-scale application are grown and normally spread across components, subsystems and multiple machines, this work matches the problem field. Ehlers [2012] follows a rule-based approach (employing OCL) that changes the instrumentation automatically for the monitoring to adapt to performance characteristics. It uses the Kieker framework as an implementation base (see Figure 5.16) and adds a *control feedback cycle* to control the instrumentation with the learned data from previously detected anomalies.

- **M10 Cost:** - (Research)
- **M11 Assessment:** Satisfying anomaly detection in lab studies without compromising monitoring overhead
- **M12 Classification:** Anomaly Detection
- **M13 Prerequisites:** Instrumentation of a software system
- **M14 Algorithms:** Time series based anomaly scoring
- **M15 Precision:** On experiment data, a high mean precision of 0.86 was recorded.

Due to the complexity of the feedback cycle, Ehlers [2012] adopted some aspects of Kieker's architecture and developed a new independent code base. This implementation was incompatible to Kieker plugins and could therefore not be used for ΘPAD. Nevertheless, it produced related work in form of a bachelor thesis, which also helped the development of ΘPAD: Frotscher [2011] investigated, implemented, and evaluated several forecasting algorithms and gave recommendations in terms of performance forecasting. These recommendations were used as a

starting point to select the forecasting algorithms for the Θ**PAD** approach. Due to also existing algorithms and scalability of $\mathcal{R}$, the remote $\mathcal{R}$ server was favored.

Bitmap-Based Online Anomaly Detection

The approach of Wei et al. [2005, p. 2] is claimed to be testable with online data available on the web. It applies the *SAX* algorithm used for data mining and hence applies *CGR* to generate bitmaps from the discretized measurements. From these time series bitmaps, anomalies are calculated using manual or automatic anomaly classification (see Figure 5.17). This can be done without requiring domain knowledge: experts can determine abnormal behavior by selecting suspicious bitmaps only.

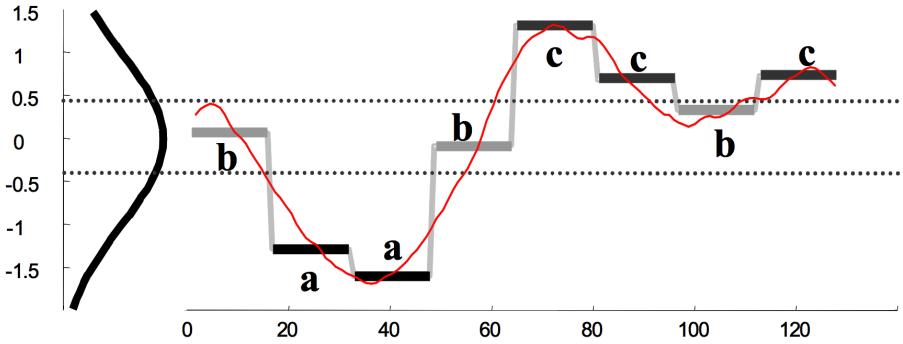

Figure 5.17: A time series get transformed to a bitmap [Wei et al. 2005, p. 2].

- **`M10` Cost:** - (Research)
- **`M11` Assessment:** Promising results with cardiologists configuring the detection with visualized behavior
- **`M12` Classification:** Anomaly Detection, based on time series
- **`M13` Prerequisites:** Classification of abnormal behavior
- **`M14` Algorithms:** Time series based anomaly scoring
- **`M15` Precision:** No absolute values are given. However, this approach was published to be tested online

RanCorr - Automatic Failure Diagnosis Support

The thesis of Marwede [2008] addresses the problem of anomaly dependencies in large-scale software systems. The approach is architecture-centric, meaning, that anomalies are assigned to the components where they occurred. Dependency graphs help determining the root cause of system failure. The steps in details are as following:

1. **Model building** to define the dependencies of the system in caller-callee relationships
2. **Aggregation** of anomaly scores. To every component, an anomaly score is calculated, which is similar to the concept used in Θ**PAD**

3. **Correlating** the architecture graph to the anomalies. Figure 5.18 shows how an anomaly in one component affects other parts of the architecture.
4. **Visualization** of the graph. Calculated anomalies are colored accordingly.

This *RanCorr* called approach uses Kieker to build the model (step 1). Step 2 applies mean calculation to build the anomaly scores. The findings in Θ**PAD** correlate to the approach of Marwede [2008] to use mean-based anomaly calculation. Since the main focus of **RanCorr** is on the correlation and visualization (steps 3 and 4), it addresses questions complementary to Θ**PAD**'s. The code used in that thesis was based on an old version of Kieker and was hence not possible to adopt due to the development of Kieker in the meantime.

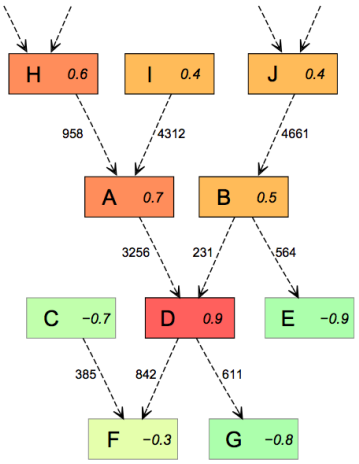

Figure 5.18: An anomaly propagating up the call tree through subsystems [Marwede et al. 2009, p. 5]

- **M10 Cost:** - (Research)
- **M11 Assessment:** A novel approach, which can lead to better examination of faults
- **M12 Classification:** Anomaly Score Calculation and Correlation, Architecture-Centric
- **M13 Prerequisites:** Long-term training data
- **M14 Algorithms:** Mean calculations
- **M15 Precision:** Results from lab tests in graphical form, not comparable to Θ**PAD**.

Spectral Clustering

Wang et al. [2008, p. 1] use feature-based spectral clustering of video sequences, which reduces computation time. This preprocessing step takes raw time series data and transforms it into sets of features. Assuming the input data has similar characteristics like performance time series, this approach could also be used for performance anomaly detection.

- **M10 Cost:** - (Open source experimental $\mathcal{R}$ code)

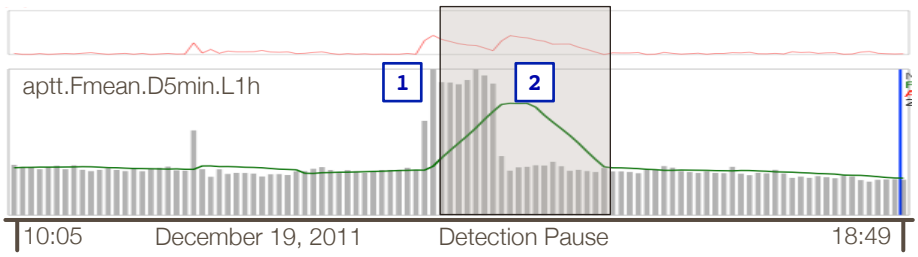

Figure 5.19: Improved TPR to FNR by pausing the algorithms in Θ**PAD** after an detection 1. This leads to ignoring the high anomaly scores at 2, which would alert false positives.

- **M11 Assessment:** Uses much preprocessing, - Approach was not proceeded further
- **M12 Classification:** Pattern recognition
- **M13 Prerequisites:** Feature preprocessing. (As fast as Θ**PAD**)
- **M14 Algorithms:** *k*-means, spectral clustering.
- **M15 Precision:** Lower than based on original data. The computational cost was lower due to the preprocessing to features.

Own Attempts to improve Detection Precision of Θ**PAD**

When the selected anomaly detection algorithm using the `MeanForecaster` detected an anomaly, the anomaly score decreases while still being in the time span of the anomaly. This is due to the detected anomalies being contextual as depicted in Figure 2.11. An example of this behavior is at point 1 of Figure 5.19.

Thus, one improvement is to disable the anomaly detector for a certain time. It discards subsequent false negatives which improves the accuracy. Additionally, the second peak when exiting the anomaly is not recognized, hence decreasing the false positives (2). Figure 5.20 shows the results as a ROC curve. In this example, the accuracy and precision were improved: $ACC = 0.202$, $PREC = 0.962$ (**M15**).

This approach can be used in practice assuming that alerts get triggered at the beginning of an anomalies. Further alerts are not necessary and can be switched off. This improvement is hence only an improvement of the evaluation and does not hamper the real accuracy of the Θ**PAD** approach.

- **M10 Cost:** - (Research)
- **M11 Assessment:** The first alert stays the same, even if nothing was detected
- **M12 Classification:** Improvement Idea
- **M13 Prerequisites:** Configuration of the pause duration
- **M14 Algorithms:** Paused anomaly detection (from Θ**PAD**)
- **M15 Precision:** Precision improved from 0.14 to 0.202 with the same forecasting algorithm.

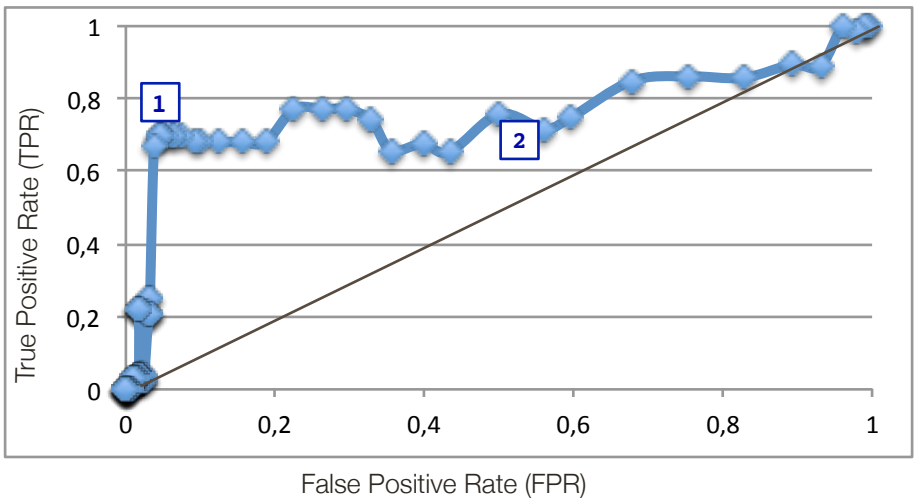

Figure 5.20: Improved TPR to FNR by pausing the detection. An appropriate threshold is $\theta = 0,33$ [1]. After a certain threshold ([2]) the precision is not significant higher compared with the random detector.

Three-Step Anomaly Detection

Xu et al. [2009, p. 3] introduce online problem detection with an additional step to identify anomalies. This could be used for ΘPAD as well. Firstly, the next value of the reference model is forecasted by means of ΘPAD. Secondly, abnormal behavior is calculated by comparing the forecast with the current observations. If the calculated anomaly score is high, the third steps applies pattern detection to ensure the exclusion of false positives.

- **M10 Cost:** - (Research)
- **M11 Assessment:** Further confirmation of the anomaly score
- **M12 Classification:** Improvement Idea
- **M13 Prerequisites:** Online measurements of log data
- **M14 Algorithms:** Additional pattern recognition step after a simple anomaly detection.
- **M15 Precision:** Not implemented in combination with ΘPAD. The approach of Xu et al. [2009, p. 3] alone resulted in a precision of 86.03%.

Dynamic Sliding Window

Shasha and Zhu [2004, p. 170] improve the anomaly detection interpretation by adjusting the sliding window size dynamically. In combination with ΘPAD, this would substitute testing out multiple window sizes at a time.

- **M10 Cost:** - (Research)
- **M11 Assessment:** Less evaluation effort needed to adjust the forecasting window size
- **M12 Classification:** Improvement Idea
- **M13 Prerequisites:** Adjust the sliding window of forecasting algorithms.

- **M14 Algorithms:** Time series analysis
- **M15 Precision:** - (No comparable metric found)

5.5.3 Questions Q4.1 - Q5.2: Answer Elaboration

In this last step of GQM the following questions are answered by using the rough data gathered for the according metrics. Most of the evaluated approaches dealt with anomaly detection of user behavior in order to conduct fraud detection research.

In consequence, a comparison between Θ**PAD** and the listed alternatives in nearly impossible since the term 'anomaly detection' is rarely used in combination with performance metrics. Hence, these questions have to be answered with a strong subjectivity.

Q4.1: "Cost of alternatives?"

- **M10 Cost:** New Relic, is the only commercial anomaly detection service found that can be used as a SaaS, priced from $24. Other solutions are open source and require a custom setup on an own server. Especially Ourmon looks promising with respect to the adaptability.

Q4.2: "How well do alternatives perform?"

- **M11 Assessment:** Each algorithm has unique characteristics, a comparison is hence difficult. The most promising results show open source software such as Snort and RRDtool that are used as a basis for further approaches. Especially Snort.AD looks promising and is close to Θ**PAD**. It works on the system-level, was published in 2012 an is not widely tested yet. Since every anomaly detection system requires specific adjustment to the system under monitoring [Bechtold and Heinlein 2004, p. 44], a product's performance can only be compared in context.

Q4.3: "What are the concepts of alternatives?"

- **M13 Prerequisites:** Most of the approaches work on online time series data like Θ**PAD**. However, the measurements that get transported are different and have to be adjusted to the system under measurement as well. New Relic offers a ruby gem that can be used with little configuration effort in existing Ruby on Rails applications. Concepts that rely on graph algorithms such as Ourmon are not alternatives to the time series based approach of Θ**PAD** but could give inspiration for future work in this field.
- **M14 Algorithms:** The software packages focus more on the pragmatic part of gathering data and calculating anomalies based on profiles adjusted by the user. Scientific approaches rather try to apply sophisticated algorithms on academic data sets.

Q5.1: "Are there alternative algorithms?"

- **M12 Classification:** Only the CAD and the spectral clustering seem related enough in this context.
- **M13 Prerequisites:** The two possible algorithms need sophisticated steps for preprocessing and thus are not directly usable in the ΘPAD server.
- **M14 Algorithms:** No details were given for the commercial products. As far as the documentations give insight, many approaches are based on statistical analysis as well. Apart from time series, pattern recognition of time series, for instance achieved by the application of spectral clustering of Wang et al. [2008, p. 1], seems to be a viable alternative. Another promising approach is to combine pattern recognition and simpler approaches in subsequent steps as done by Xu et al. [2009, p. 3].

Q5.2: "Can the interpretation be improved?"

- **M12 Classification:** Some academic approaches also work in the field for online time series data. However, there is no concept dealing with the same setup like ΘPAD and only ideas and parts of the approaches can be used to improve ΘPAD.
- **M14 Algorithms:** ΘPAD is open for new algorithms, but this set of related work did not include an algorithm that fit into ΘPAD without further modifications of the implementation.
- **M15 Precision:** Figure 5.20 shows that simple changes in the interpretation can lead to improvements in accuracy and/or precision. The work of Ehlers [2012] can also give hints on how to improve the anomaly score precision.

5.5.4 Goals G1 and G2: Alternatives and Improvement Approaches

Table 5.11 and Table 5.12 summarize the answers on goals according to the GQM paradigm.

Goal	**G4**	Find Alternative Approaches
Questions	**Q4.1**	Is Cost of alternatives?
	Q4.2	How well do alternatives perform?
	Q4.3	What are the concepts of alternatives?

Answer

This goal was not reached to satisfaction. The closest approaches are open source projects (**Q4.1**) that can work on the same input data. These concepts address nearly the same requirements as Θ**PAD** does (see Section 4.1), but much work is needed to apply them to the same field of anomaly detection of online performance data for comparison (**Q4.2**). However, **Q4.2** did not provide comparable data but can serve as a basis to start further research in this field.

Table 5.11: Results of **G4**: "Find Alternative Approaches"

Goal	**G5**	Improve the Θ**PAD** Detection
Questions	**Q5.1**	Are there alternative algorithms?
	Q5.2	Can the interpretation be improved?

Answer

For the anomaly detection step, no anomaly detection algorithms were found (**Q5.1**) that can directly be used in Θ**PAD**. Most alternatives differed in general approaches and input data. However, these insights can be used for further improvements of Θ**PAD**. Other means can be evaluated to measure system performance or to interpret the calculated anomaly scores.

Combination Possibilities

It is, for instance, possible to combine Θ**PAD** or a system-level monitoring product, such as Ourmon or Snort, with other scientific work already made by the Software Engineering Group of the University of Kiel such as of Ehlers [2012].

Architecture-based Anomaly Diagnosis and Visualization

RanCorr, the approach of Marwede et al. [2009] enhances the anomaly detection with automatic diagnosis for localizing root causes of anomalies. The dependency graphs an visualization could be adopted for Θ**PAD** to show anomaly scores at submodules to augment its post-mortem analysis graphs.

Table 5.12: Results of **G5**: "Improve the Θ**PAD** Detection"

Chapter 6

Conclusion

From October 2011 through December 2011, this research covered the planning, design and implementation of the ΘPAD approach (Chapters 3 and 4). Following was an evaluation phase including a two week long experiment. In this time, the anomaly detector was gathering data from the online XING platform. During this period, eight anomalies were discovered in the production data and subsequently used to evaluate the practicability of the approach.

This final chapter concludes the findings in Section 6.1 and brings up a discussion about the practicability in Section 6.2. Based on the results, a retrospective in Section 6.3 explains how the thesis' goals were reached. Finally, Section 6.4 presents the chances for future improvements and applications of ΘPAD.

6.1 Summary

In Chapter 1 recent trends of *Software as a Service (SaaS)* introduced the necessity of *Quality of Service (QoS)* in this field. Amongst other metrics, this term motivated the research on service performance. Issues in this respect were abnormal behavior that can cause the user to experience bad response times.

The online platform XING is a large-scale software system that has an infrastructure for performancy analyses. However, automatic online detection of anomalies was not supported by this software called Logjam (see Section 2.4.4). In order to fill this gap, the ΘPAD approach was conceptually described in Chapter 3 and thereafter implemented as a prototype in Java (Chapter 4).

ΘPAD is a configurable server implemented as a plugin for the Kieker framework for online performance monitoring and dynamic analysis. This plugin offers performance measures that can be configured for characteristics of the system under monitoring. It receives measurements in temporal sequences and extracts time series. After this discretization, time series forecasting can be used to calculate anomaly scores from current measured value. Finally, configurable thresholds are a means to classify behavior as abnormal and subsequently propagate them to alerting facilities.

In Chapter 5, an experiment on production data of the case-study system was conducted resulting in the discovery of eight anomalies in a period of two weeks. Based on these findings, the thesis' research questions were answered using the GQM paradigm. ΘPAD detected the anomalies with a high accuracy of 0.97 but a low precision of 0.14. A subsequent evaluation of related work and improvement approaches revealed several other research and industry approaches and tested an attempt to improve ΘPAD's detection precision.

The following section will discuss the discovered results and answer questions implied by the thesis' goals.

6.2 Discussion

Based on the GQM evaluation, this section discusses the results with respect to the issues of anomaly detection. Further on, a description of the lessons learned is given followed by a retrospective on how well the goals of the thesis were reached.

6.2.1 Issues

The evaluation answered questions directed to results. These answers were supported by certain metrics as suggested by the GQM paradigm. This section critically weighs the expectations against the issues.

▷ Does OPAD detect anomalies satisfyingly?

In the introduction (see Chapter 1), thesis goal **T1** aimed at detecting anomalies in large-scale software systems. Chapter 5 substantiates this with research question **RQ1**: "Can anomalies be detected automatically?"

The accuracy/precision curve in Figure 5.14 clearly indicates that ΘPAD detects anomalies with a precision of $PREC = 0.14$ using the best algorithm. However, the accuracy was high, which makes the approach more practical: few false alarms are reported unnecessarily. This is especially important for online detection since every false alarm leads to an unnecessary notification of administrators, and that cost "cannot be amortized" [Song et al. 2007, p. 1]. In order to apply a feasible anomaly detection, first the best algorithm has to be selected (see Figure 5.12) and second an appropriate threshold has to be found by making the tradeoff between accuracy and precision diligently. The necessity of taking both measures into account is also stated by Oliner et al. [2012, p. 7]: "It is important to investigate techniques that trade off efficiency, accuracy, and actionability".

Apart from that, the metrics accuracy and precision do not reason about the timeliness of the detection. Anomaly score graphs, for instance in Figure 5.19, show a high anomaly score decreasing fast if the algorithm only detects contextual anomalies. For global anomalies, the first detection was a true positive and would lead to a correct alert. However, the subsequent anomaly scores are calculated low due to the measure being perceived as normal due to a collective anomaly.

This can lead to a bad precision reflecting in the ROC curves. To address this, one improvement approach is modifying the interpretation of anomaly scores. Figure 5.20 shows that even simple modifications lead to improved presision without compromising the accuracy.

▷ Is anomaly detection based on time series analysis practical?

Goal `T1` did not predefine details of the approach taken by ΘPAD. Using forecasting on time series of measurements rather came from the characteristic of performance measures that gather records continuously. Moreover, the case study system already implemented monitoring and offered the measurements on the importer queue.

In the evaluation phase, all eight manually detected anomalies could be traced to certain performance measures. From the constructed time series, anomaly scores were calculated indicating a level of abnormality. The evaluation shows that time series of performance are a means to perform anomaly detection.

Still, the interpretation of the detection is hard to employ using the rate of true positives and false negatives: Since anomaly scores can be lagged depending on the algorithm, an evaluation based on comparing observations appears too simple. Anomaly detection can be achieved with this approach, but the interpretation has to be adjusted accordingly using ROC curves and programmatic means. Further research on the interpreation should be made in order to evaluate other interpretation means.

▷ Were the problems of the case study solved?

At best, the administrators of the XING software system should be alerted completely automatically by the ΘPAD server. This demand is still open due to the low precision and the lack of an adapter that connects ΘPAD to XING's Nagios alerting facility.

Although the work needed to connect these systems is assumingly low, ΘPAD still has to be configured better. Especially the measures can be adjusted finer in order to gather information from different attributes and subsystems. Since this requires some further setup and testing effort, this could not be done in the course of the thesis. However, it is already planned to deliver the solution in practice in one month succeeding this thesis work. How this succeeding work is planned explains Section 6.4.3.

▷ Was the technology usage appropriate?

The introduction, Section 1.1, reckoned a possible solution offering multiple measures and adjustable anomaly detection approaches. These detection algorithms had to be configurable and testable on an online production system.

The implementation in software (see Chapter 4) was designed as a plugin of the Kieker framework and relying on supporting services and libraries such as Esper,

MongoDB, `TSLib`, and $\mathcal{R}$. This induced a big setup overhead and required a sophisticated architecture. This complexity becomes visible in structures such as the `DataBeat` pattern in Section 4.4.2.

Time series forecasting is a complex field as the $\mathcal{R}$ manual [R Development Core Team 2011] displays. The combination of existing, tested software was a good approach to deliver the desired functionality in the given time frame. In order to address software errors, a large test base was written using JUnit and the TDD paradigm. Moreover, the Θ**PAD** server is written as a plugin of Kieker that was proven to introduce low overhead into the monitored system. Answering the GQM question **`Q1.1`** ("Is Θ**PAD**'s server stable?") further assures the robustness of the implementation.

6.2.2 Lessons Learned

Anomaly detection is possible with ΘPAD when configured properly

The approach taken in by this thesis addresses all requirements stated in the introduction in Section 1.1. Although metrics do not show a high precision, future improvement in configuration and interpretation is promising. Furthermore, anomaly detection always introduces the trade-off between true positives and false negatives. Θ**PAD**'s detection behavior is rather towards high accuracy, meaning a low tolerance of abnormal behavior.

Further on, we learned that a proper configuration of Θ**PAD** is crucial in order to detect anomalies with a certain precision. In the evaluation, Figure 5.12 demonstrated algorithms, which performed with a FPR being too low for practical use and only few that performed well. This leads to the conclusion that much care has to be taken at configuration time in order to find the right algorithm and configuration to fit the environment's characteristics well.

The `MeanForecaster` is the best algorithm in this setup

As shown in the evaluation, the ROC curves in Figure 5.12 clearly show that the best detection is made by using the mean of a sliding forecasting window. Interestingly, in informal conversations at XING, this result was assumed before. Employees familiar with the matter expected behavior of performance normally not vary much in the short term.

Anomaly detection demands expert knowledge

The related work (see Section 5.5) did not reveal a software product that is capable of detecting anomalies without custom setup. This necessity was experienced in the course of the implementation as well. Θ**PAD**'s Kieker plugin had to be adjusted to custom measures and communication channels in order to work in the case-study environment.

Further on, the configuration of aspects had to be made before running the experiment. Without this setup (Section 5.2) the environment's performance measures would not have been known. The related work of the previous chapter shows that even anomaly detection based on machine learning or pattern recognition does not work without human interaction.

Time series visualization supports performance reasoning

Both graphical web front ends, Logjam, and the anomaly score graph described in Section 4.6.3 helped the post-mortem analysis to find causes for anomalies. Especially Logjam is in use by all development teams at XING and helps engineers getting a view on the actual behavior of their online system.

Using open source software gave great benefits

All software packages used by Θ**PAD** are open source. For instance, without the algorithms provided by $\mathcal{R}$, Θ**PAD** could not have been developed in the given time. The Kieker framework also decreased the development time since it already provided a basic architecture with the linear data flow used by the Θ**PAD** plugin.

Alternative software such as Snort and RRDtool demonstrate the more a software is used, the more improvements are made. They both offer many additional tools and components built by the community. Θ**PAD** also has this potential due to its reusability (as of NFR4) in Section 4.1 and the affinity to the open source community of Kieker.

6.3 Retrospective: Goals Reached?

All four thesis goals listed in Section 1.2 were reached. The following list summarizes the reasons.

- ▷ `T1`: **Design of an Online Performance Anomaly Detection Concept**
  An approach was developed in in Chapter 3 based on some mathematical foundation in Chapter 2. This approach deals with performance data in an online fashion and detects anomalies based on time series.
- ▷ `T2`: Θ**PAD Implementation as a Kieker Plugin**
  The implementation (Chapter 4) was done by using the Kieker monitoring framework as a basis. Θ**PAD** was developed as a plugin that runs in it's context.
- ▷ `T3`: Θ**PAD Integration with Case-study System**
  In the case study, Θ**PAD** was evaluated in the environment over the course of two weeks. It was configured to gather selected performance measurements from the monitoring system at XING. In the course of the evaluation phase, it detected eight anomalies in production data.
- ▷ `T4`: **Evaluation**
  Chapter 5 used the GQM approach to answer questions implied by the thesis'

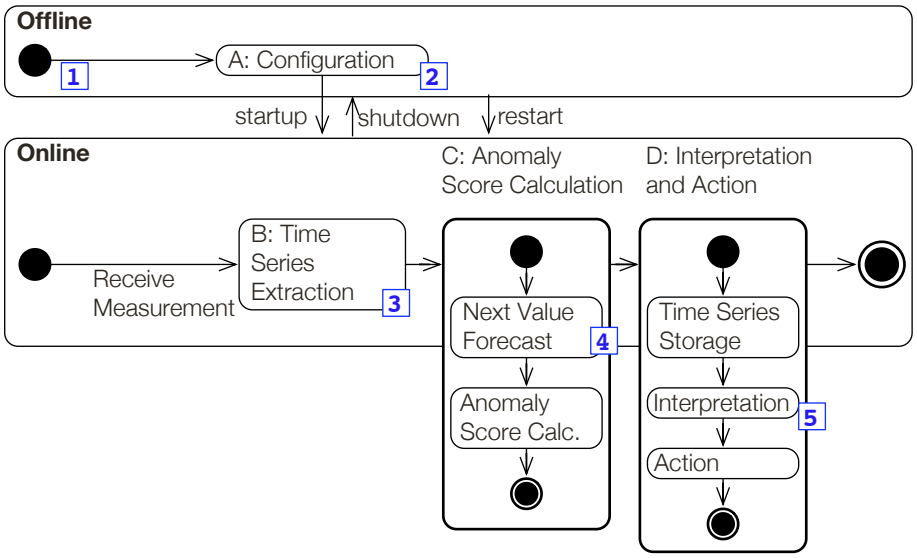

Figure 6.1: The activities of the ΘPAD approach are starting points for future improvements. The markers 1 to 5 are referred to in Section 6.4.1.

goals. All important metrics were measured directed to goals relevant for the reasearch goals.

A main focus was placed on the implementation (T2) due to the engagement of the advisory in the Kieker framework. It is anticipated that the research of the Software Engineering Group can be supported by the ΘPAD approach. Possible future work and improvements of the approach are described in the following section.

6.4 Future Work

6.4.1 ΘPAD Improvements

During implementation and evaluation several approaches came up to improve the ΘPAD server. Figure 6.1 takes the activities described in Chapter 3 and indicates the points of improvement. Subsequently, details are given in the following sections.

1 Adapt to Kieker Pipes and Filters

During the period of this thesis, the Kieker framework was enhanced with a pattern called Pipes and Filters. This makes gathering measurements and subsequent processing configurable at run time.

With this approach, the ΘPAD plugin could serve as a filter that calculates anomaly scores and propagates them inside Kieker for further processing and

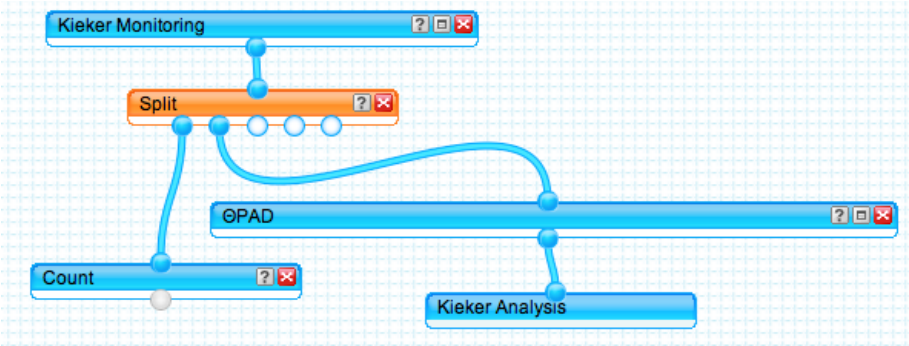

Figure 6.2: Simulation Yahoo's *Pipes and Filters* editor including Kieker and Θ**PAD**. In this demonstration, Yahoo's editor was modified with Google Chrome.

analyses. Additionally, the configuration activity can be supported with a graphical user interface. This interface is planned for 2012 and is similar to the Yahoo Pipes as demonstrated in Figure 6.2. Interesting setups could be found when coupling Kieker filters with anomaly scores or post-mortem analyses graphs from Θ**PAD**.

2 Different Source Aggregation

In the evaluation (Chapter 5), the performance measure `total_time` prove to correlate with actual anomalies (Question **Q2.3** "Which performance attributes matter?"). However, the available measures were restricted to the environment's own monitoring. Further investigation can be made in finding measures from external sources such as network provider performance or even social media activities.

Oliner et al. [2012, p. 7] even suggest using phone-call logs or customer inquiry metrics for cross correlation. These approaches would benefit from the Pipes and Filters architecture since many combinations could be tried in parallel.

3 Automatic Choice of Sliding Window

As investigated in the related work in Section 5.5, StatStream, the software written by Shasha and Zhu [2004, p. 170] uses *sliding windows* to adjust the algorithm's input data dynamically.

Future versions of Θ**PAD** could adopt this principle and change the size of the sliding window based on detection metrics such as precision.

4 Algorithm Refinement

As described in detail in the following Section 3.5, the anomaly detection method is based on forecasting as an intermediate step. Several algorithms were selected based on the suggestions by Frotscher [2011, p. 17]. Still, other algorithms can be evaluated and tested on production data. Since the architecture of Θ**PAD**

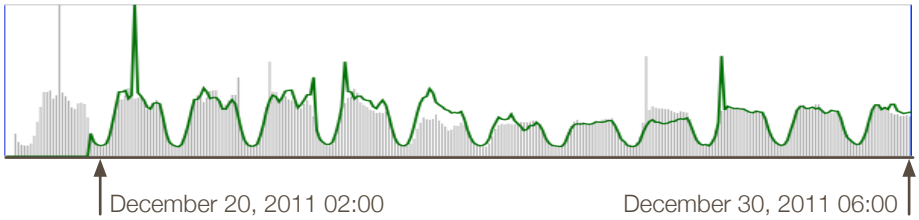

Figure 6.3: The `SeasonForecaster` plotting the forecast over a period of 12 days. The gap at $\widehat{1}$ shows that not enough data is known to perform the forecast. Improvement can be done for instance at $\widehat{2}$.

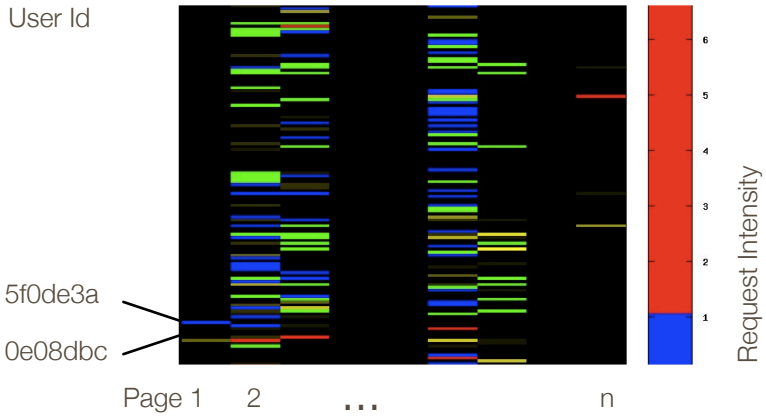

Figure 6.4: Heat Map of excessive XING Users

is designed to allow the addition of algorithms, the evaluation of new anomaly detection approaches is encouraged.

Another refinement can be made by parameterizing existing algorithms with knowledge of usage behavior For instance, the `SeasonForecaster` as shown in Figure 6.3 could not only calculate the forecast with the season but also with special events or trends.

6.4.2 Bot Activity Detection

XING had a bot detection mechanism that recorded excessive usage of the platform. The assumption was that those usage patterns come from users that are 'not normal'. This abnormal usage could come from *bots* or machines that automatically try out password etc. The software tracked all users performing more than 10 requests on `xing.com` in a certain sliding window.

Figure 6.4 shows the output of the tool with example data. The y axis shows the suspicious users, the x axis separates the different pages that were requested (see Section 2.4.4). The *heat*, as a third dimension, is the aggregation of the individual user's request.

The concept worked for an older version at XING, one, which is not in use any more because of recent changes of the platform. The usage patterns were tracked and evaluated online but not supervised outside of business hours. So it is imaginable to connect ΘPAD to this data source. Online usage anomaly detection would hence be in reach.

6.4.3 ΘPAD in the Kieker Community

In the course of this thesis, the field of performance anomaly detection was researched and much insight was gained especially due to the requirement for the detection to be *online*, on a productive system. Most of the software used to build ΘPAD and to write this thesis was available as open source. Apart from the publication of the research, the created software be a basis for future research and (hopefully) production use.

The further publication of ΘPAD is planned as following:

- `TSLib`: Available at a *github repository*[1] for easy forking and commenting
- The ΘPAD concept and code at repositories of Kieker[2]
- A guide for Setup and Configuration of ΘPAD in the Kieker *trac*[3]

After the submission, a month's employment at XING will give the possibility to to further testing and research on production data.

One future use of `TSLib` will be in a diploma thesis at the Descartes Research Group of the Karlsruhe Institute of Technology[4] authored by Herbst [2012]. A heuristic approach will be developed that selects forecasting strategies. Characteristics of time series will be analyzed for improving forecasting quality. Further on, a case study on IBM's Tivoli software for managing elastic virtualized systems is planned.

Additionally, it is planned to summarize the results of this thesis in a research paper to be submitted to a conference.

[1] `https://github.com/tielefeld/tslib`
[2] `http://kieker-monitoring.net/opad/`
[3] `https://build.se.informatik.uni-kiel.de/kieker/trac/opad/`
[4] `http://descartes.ipd.kit.edu/`

Acknowledgements

I would like to express my gratitude to the following people:

- André van Hoorn for better supervision and guidance than I ever expected. Through countless revisions and corrections, he basically taught me academic English writing.
- Prof. Dr. Wilhelm Hasselbring and Dr. Stefan Kaes for developing an interesting and challenging topic, great supervision, feedback, and consultancy.
- Dr. Jennifer Salau for helping with the math, Anna Allen, Carla Lee Hing, and Manuela Bache for polishing every English expression.
- XING AG for letting me work freely in its environment and for having the most innovative and beautiful employees ;-)
- My team at empuxa GmbH for supporting its CEO abroad.

Declaration

I certify that the work presented in this diploma thesis is to the best of my knowledge and to my belief, the original, except as acknowledged in the text. All passages, which are literally or in general matter taken out of publications or other resources, are marked as such.

The material has not been submitted for a degree at this or any other University.

Location, Date Tillmann Carlos Bielefeld

Glossary

Accuracy Ratio of correct detections to all detections. III, 22, 83, 100

AOP Programming paradigm that treats different logical aspects separately. 26

Aspect Unit of configuration in ΘPAD. 34–41, 44–46, 51, 52, 55, 59, 62, 65, 67, 74, 75, 82, 83, 103

Attribute Performance measure on the monitored system. Also used for aspects to listen on these system measures. 37, 38, 41, 73–75, 78

Availability Property of a system being able to deliver service requests at a given point in time. III, 1

Efficiency Internal and external quality. 7

Feature Characteristics of time series. 11, 93

FPR Ratio of false positive detections to all points of normal behavior. 21

Importer A process in the XING architecture which aggregates server logs. 29, 31, 55, 74, 77, 101

Measurand Object which gets values assigned in the measuring process. 7, 11, 20, 29, 37, 38, 40

Measure Entity that produces measurements in a specific scope. III, 1, 7, 11, 33–37, 40, 41, 76, 89, 99, 101, 103, 105

Measurement A single measured value. 2, 4, 7, 8, 10–12, 15, 29, 33–35, 37–41, 50, 54, 56, 65–67, 99

Online A system's state in runtime, able to serve requests. 4, 89

Page Grouping attribute of XING's web application which assigns application components to use cases such as 'Jobs', 'Events' and 'Billing'. 28, 29, 65, 75

Performance the degree to which a software system or component meets its objectives for timeliness. III, 1–3, 7, 10, 17, 29, 34, 75, 99

QoS The combination of availability, reliability, and performance. III, 1, 2, 99

Raw measurement Measurement from a continuous data stream. 7, 12, 29, 34, 38, 54

Reliability Reliability is given when a system services request according to its functionality. 1, 10

Response time Time period between user interaction and the system presenting a result. 8, 9, 17, 18, 29, 34–36, 77, 88

REST Programming paradigm defining constraints in the communication between client and server. 28

Robustness Property of a system that tolerates incorrect input [Avizienis et al. 2004, p. 23]. III, 2, 49, 86, 102

Sharding Splitting a large collection of data across several servers . 3

SMM The *Structured Metrics Metamodel* is a meta-model for representing measurement information specified by the Object Management Group, Inc.. 8, 37

Step size Distance between two succeeding point in a time series. 10–13, 38, 41, 48, 50, 51, 53, 55, 65, 75

Temporal sequence A series of events with a specific scope. 11, 12, 29, 34, 38, 40, 50, 54, 58, 99

Threshold The anomaly score level which separates abnormal from normal behavior. 20, 35, 37, 44, 45, 64–66, 73, 75, 82, 83, 86, 88, 99, 100

Software Libraries and Products

$\mathcal{R}$ Runtime environment and programming language used to execute forecasting algorithms in ΘPAD, http://www.r-project.org. 3, 13, 14, 27, 51, 59, 102

AMQP Messaging standard 'Advanced Messaging Protocol', http://www.amqp.org/. 24, 25, 28, 29, 55, 56, 65

Apache Qpid Open source messaging, http://qpid.apache.org/. 25

AppDynamics Hosted application performance monitoring service, http://www.appdynamics.com/ . 1

BSON Extension of the JSON markup with additional binary encoded content. 23, 30

CRAN 'The Comprehensive R Archive Network', an online respository for packages of the R language, http://cran.r-project.org/. 14, 27

D3 'Data-Driven Documents', Javascript library do develop and bind graphical representation to data, http://mbostock.github.com/d3/ . 29, 66

EPL Event Processing Language, the query language of the Esper engine. 50

Esper Complex Event Processing (CEP) engine, http://esper.codehaus.org/, http://www.espertech.com/. 12, 50, 51, 101

Facebook The largest and fastest growing social network with over 845 million active users, http://facebook.com/. 2

Git Decentralized source code management, http://git-scm.com . 59

GNU An open source UNIX-like operating system, http://www.gnu.org/philosophy/free-sw.en.html. 27

Graylog2 Open source log management, http://graylog2.org . 29

Java Language of Kieker and ΘPAD. Version 1.6 for 64bit OSX 10.7 was used, http://java.com/ . 51, 52, 55, 58, 59, 64, 99

Jenkins Continuous integration server, http://jenkins-ci.org . 59

JRclient Java client library to connect to a running Rserve instance via socket, http://stats.math.uni-augsburg.de/rserve/dist/JRclient/JavaDoc/ . 51, 59, 61

JSON Text-based human-readable encoding that can directly be interpreted by Javascript. 23, 24, 29, 54, 65

JUnit Unit tests for Java, http://junit.org . 58, 102

Kieker Open source performance monitoring and dynamic analysis framework, http://kieker-monitoring.net/. III, 1–5, 7, 47, 50, 53–56, 91, 93, 99, 101–105, 107

Logjam XING's logging aggregator and viewer, https://github.com/alpinegizmo/logjam. 3, 28, 29, 50, 75, 77, 78, 99, 103

Memcache Open source caching system, http://memcached.org/. 29

MongoDB "Hu**mongo**us data base", high-performance and availability document store database, http://www.mongodb.org/. 29–31, 50, 51, 55, 65, 66, 75, 77, 102

Munin System and Resource Monitoring, http://munin-monitoring.org/ . 29

Nagios Industry standard IT infrastructure monitoring system, http://www.nagios.org/. 1, 3, 29, 101

Neo4J Open source NoSQL graph database, http://neo4j.org. 28

New Relic Service for web monitoring and management, http://newrelic.com . 1, 88, 96

NoSQL 'Not only SQL', databases without the need of schema and relational integrity. 25

Omniture Web analytics and monitoring tool, http://www.omniture.com/de . 29

OpenForecast Forecasting library for Java, http://www.stevengould.org/software/openforecast/index.shtml. 59

Ourmon network monitoring and anomaly detection system, http://ourmon.sourceforge.net . 89, 96, 98

Perl The majority of XING's code base is still writtin in this open source programming language, http://www.perl.org/. 3, 29

Pipes and Filters Architecture pattern that allows the combination of computation be configured by the user, http://www.eaipatterns.com/PipesAndFilters.html. Yahoo's software is available at http://pipes.yahoo.com/ . 104, 105

RabbitMQ The message queuing server used at XING, http://www.rabbitmq.com/. 25, 28

Rackspace Hosting provider, http://www.rackspace.com/managed_hosting/services/security/ddosmitigation/ . 88

RRDtool Round Robin Database, http://oss.oetiker.ch/rrdtool/ . 89, 96, 103

Rserve Server offering connection to the R runtime, http://www.rforge.net/Rserve/files/ . 50, 51, 59, 61

Ruby on Rails Full-Stack web development framework written in the scripting language Ruby, http://rubyonrails.org, http://www.ruby-lang.org/. 3, 28, 29, 96

Safari rowser Apple's browser based on the Webkit rendering engine, http://www.apple.com/safari/ . 66, 67

Sinatra Sinatra, http://www.sinatrarb.com/ . 66

Snort Intrusion detection system, http://www.snort.org/ . 89, 96, 98, 103

Snort.AD Preprocessor for Snort to detect anomalies in network traffic, http://anomalydetection.info . 89, 96

StatStream http://cs.nyu.edu/cs/faculty/shasha/papers/statstream.html . 89, 105

StormMQ Message queuing as a service, http://stormmq.com/. 25

TSLib Time Series Library, a part of the ΘPAD approach. 50–52, 57, 59, 102, 107

XING The global social network XING is the most important platform for business contacts in Europe., http://xing.com/. 2, 3

YAML Human-readable markup language, 'YAML Ain't Markup Language', http://yaml.org/ . 24, 52, 59, 62

Bibliography

[Avizienis et al. 2004] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput., 1:11–33, January 2004.

[Banerjee et al. 2008] A. Banerjee, V. Chandola, V. Kumar, and J. Srivastava. Anomaly detection - a survey, presentation slides. Technical report, University of Minnesota, Computer Science Department, 2008.

[Basili et al. 1994] V. R. Basili, G. Caldiera, and H. D. Rombach. Goal Question Metric Paradigm. In J. J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pages 528–532. John Wiley & Sons, 1994.

[Bechtold and Heinlein 2004] T. Bechtold and P. Heinlein. Snort, Acid & Co. Open Source Press, 2004.

[Beck and Andres 2004] K. Beck and C. Andres. Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional, 2004.

[Becker et al. 2006] S. Becker, W. Hasselbring, A. Paul, M. Boskovic, H. Koziolek, J. Ploski, A. Dhama, H. Lipskoch, M. Rohr, D. Winteler, S. Giesecke, R. Meyer, M. Swaminathan, J. Happe, M. Muhle, and T. Warns. Trustworthy software systems: A discussion of basic concepts and terminology. SIGSOFT Softw. Eng. Notes, 31(6):1–18, 2006.

[Ben-Kiki et al. 2009] O. Ben-Kiki, C. Evans, and B. Ingerson. YAML ain't markup language (YAML) (tm) version 1.2. Technical report, YAML.org, 9 2009. URL `http://yaml.org/spec/1.2/spec.pdf`.

[Birman 2010] K. Birman. A history of the virtual synchrony replication model. In Replication, pages 91–120, 2010.

[Bostock et al. 2011] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. IEEE Transactions on Visualization and Computer Graphics, 17: 2301–2309, Dec. 2011.

[Box and Jenkins 1990] G. E. P. Box and G. Jenkins. Time Series Analysis, Forecasting and Control. Holden-Day, Incorporated, 1990.

[Boyd 2007] E. Boyd. Social network sites: Definition, history, and scholarship, 2007.

[Burns and Wellings 2009] A. Burns and A. Wellings. Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX. Addison-Wesley Educational Publishers Inc, USA, 4th edition, 2009.

[Chan et al. 2003] P. K. Chan, M. V. Mahoney, and M. H. Arshad. A machine learning approach to anomaly detection. Technical report, 2003.

[Chandola et al. 2007] V. Chandola, A. Banerjee, and V. Kumar. Outlier detection - a survey. Technical Report 07-017, University of Minnesota, 2007.

[Chandola et al. 2009] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. ACM Comput. Surv., 41:15:1–15:58, July 2009.

[Chodorow 2011] K. Chodorow. Scaling MongoDB. O'Reilly Series. O'Reilly Media, 2011.

[DeCandia et al. 2007] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev., 41:205–220, Oct. 2007.

[Deloitte LLP 2012] Deloitte LLP. Measuring facebook's economic impact in europe, January 2012.

[Ehlers 2012] J. Ehlers. A Self-Adaptive Monitoring Framework for Component-Based Software Systems. PhD thesis, 2012.

[Ehlers and Hasselbring 2011] J. Ehlers and W. Hasselbring. Self-adaptive software performance monitoring. In Software Engineering 2011, Lecture Notes in Informatics, pages 51–62. GI, 2011.

[Ehmke et al. 2011] N. Ehmke, A. van Hoorn, and R. Jung. Kieker 1.4 user guide. `http://kieker-monitoring.net/documentation/`, Oct. 2011.

[Eisenberg 2002] J. D. Eisenberg. SVG Essentials. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1 edition, 2002.

[Fielding et al. 1999] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999. URL `http://www.ietf.org/rfc/rfc2616.txt`.

[Fowler 2003] M. Fowler. Patterns of enterprise application architecture. The Addison-Wesley signature series. Addison-Wesley, 2003.

[Frotscher 2011] T. Frotscher. Prognoseverfahren für das Antwortzeitverhalten von Software-Komponenten. Bachelor's thesis, Christian-Albrechts-Universität zu Kiel, March 2011.

[Fulman and Wilmer 1999] J. Fulman and E. L. Wilmer. Ecmascript language specification. http://www.ecma-international.org/publications/files/ecma-st/ecma-262.pdf. Ann. Appl. Probab, 9:1–13, 1999.

[Gamma et al. 1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns. Addison Wesley, Reading, MA, 1995.

[Gualtieri et al. 2009] M. Gualtieri, J. R. Rymer, R. Heffner, and W. Yu. The forrester wave™: Complex event processing (cep) platforms. Technical report, Forrester Research, August 2009.

[Günther 2010] S. Günther. Multi-dsl applications with ruby. IEEE Software, 27: 25–30, 2010.

[Hasselbring 2008] W. Hasselbring. Entwicklung verlässlicher Software-Systeme. Zeitschrift für wirtschaftlichen Fabrikbetrieb, 103(6):413–417, June 2008.

[Hayden 2010] L. Hayden. IT Security Metrics: A Practical Framework for Measuring Security & Protecting Data. McGraw-Hill, 2010.

[Herbst 2012] N. R. Herbst. Workload Classification and Forecasting in Cloud Computing Environments. Proposal for a Diploma Thesis, February 2012.

[Hichert 2011] H. Hichert. Geschäftsdiagramme mit excel nach den SUCCESS-Regeln gestalten. Haufe-Lexware, 2011.

[Hickson 2012] I. Hickson. HTML 5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft, February 2012.

[Holdener 2008] A. T. Holdener, III. Ajax: the definitive guide. O'Reilly, first edition, 2008.

[Huang et al. 1995] Y. Huang, C. M. R. Kintala, N. Kolettis, and N. D. Fulton. Software rejuvenation: Analysis, module and applications. In FTCS, pages 381–390, 1995.

[Hyndman and Khandakar 2008] R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for r. Journal of Statistical Software, 27(3):1–22, 7 2008.

[Hyndman et al. 2012] R. J. Hyndman, S. Razbash, and D. Schmidt. Forecasting functions for time series, 02 2012. URL `http://cran.r-project.org/web/packages/forecast/forecast.pdf`.

[IEEE 1990] IEEE. Standard Glossary of Software Engineering Terminology 610.12-1990, volume 1. IEEE Pre, 1990.

[IETF 2006] IETF. JSON RFC for the Javascript Object Notation, 2006. URL `http://www.ietf.org/rfc/rfc4627.txt`.

[ISO/IEC 2001] ISO/IEC. ISO/IEC 9126 Standard. Number 9126. 2001.

[Jain 1991] R. Jain. The Art of Computer Systems Performance Analysis. John Wiley & Sons, New York, 1991.

[Kaplan 2008] J. M. Kaplan. How saas is overcoming common customer objections. cutter consortium: Sourcing and vendor relationships. Technical report, Cutter Consortium, Executive Update Vol. 8, No. 9, 2008.

[Koziolek 2008] H. Koziolek. Dependability metrics. chapter Introduction to performance metrics, pages 199–203. Springer-Verlag, Berlin, Heidelberg, 2008.

[Leach et al. 2005] P. Leach, M. Mealling, and R. Salz. Rfc 4122: A universally unique identifier (uuid) urn namespace. Technical report, IETF, 2005. URL http://www.ietf.org/rfc/rfc4122.txt.

[Makridakis et al. 1978] S. Makridakis, S. Wheelwright, and Hyndman. Forecasting: methods and applications. Wiley/Hamilton series in management and administration. Wiley, 1978.

[Marwede et al. 2009] N. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring. Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation. European Conference on Software Maintenance and Reengineering, 0:47–58, 2009.

[Marwede 2008] N. S. Marwede. Automatic failure diagnosis based on timing behavior anomaly correlation in distributed java web applications. Master's thesis, Carl von Ossietzky Universität Oldenburg, Germany, Aug. 2008.

[Maxion and Roberts 2004] R. A. Maxion and R. R. Roberts. Proper Use of ROC Curves in Intrusion/Anomaly Detection. Technical Report CS-TR-871, School of Computing Science, University of Newcastle upon Tyne, Nov. 2004.

[McConnell 2004] S. McConnell. Code Complete, Second Edition. Microsoft Press, Redmond, WA, USA, 2004.

[Membrey et al. 2010] P. Membrey, E. Plugge, W. Thielen, and T. Hawkins. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing. Definitive Guide Series. Apress, 2010.

[Mitsa 2009] T. Mitsa. Temporal Data Mining. Chapman & Hall/CRC data mining and knowledge discovery series. Chapman & Hall/CRC, 2009.

[Nissanke 1999] N. Nissanke. Formal Specification: Techniques and Applications. Springer-Verlag, London, UK, 1st edition, 1999.

[Oliner et al. 2012] A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. Commun. ACM, 55(2):55–61, Feb. 2012.

[OMG 2012] OMG. Software metrics meta-model. Technical report, Object Management Group, January 2012.

[O'Reilly 2005] T. O'Reilly. What is web 2.0, 09 2005. URL http://oreilly.com/web2/archive/what-is-web-20.html.

[Pham 2000] H. Pham. Software Reliability. Springer, 2000.

[R Development Core Team 2011] R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL http://www.R-project.org/.

[Raymond 2007] S. Raymond. Ajax on Rails. O'Reilly Media, Inc., 2007.

[Rebbapragada et al. 2009] U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. Alcock. Finding anomalous periodic time series. Mach. Learn., 74:281–313, March 2009.

[Reuters 2012] Reuters. Oracle to buy taleo. cloud war brews, February 2012. URL http://economictimes.indiatimes.com/tech/software/oracle-to-buy-taleo-for-1-9-bn-cloud-war-brews/articleshow/11825414.cms.

[Rupp et al. 2007] C. Rupp, S. Queins, and B. Zengler. UML 2 glasklar: Praxiswissen für die UML-Modellierung. Hanser, München, 3. edition, 2007.

[Salfner et al. 2010] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. ACM Comput. Surv., 42(3):10:1–10:42, Mar. 2010.

[Schwinn 2011] H. Schwinn. Requirements Engineering: Modellierung von Anwendungssystemen. Oldenbourg Wissensch.Vlg, 2011.

[Seow 2008] S. Seow. Designing and engineering time: the psychology of time perception in software. Addison-Wesley, 2008.

[Shasha and Zhu 2004] D. Shasha and Y. Zhu. High performance discovery in time series: techniques and case studies. Monographs in computer science. Springer, 2004.

[Shneiderman 1984] B. Shneiderman. Response time and display rate in human performance with computers. ACM Comput. Surv., 16:265–285, September 1984.

[Smith and Williams 2002] C. U. Smith and L. G. Williams. Performance Solutions: a practical guide to creating responsive, scalable software. Addison-Wesley, 2002.

[Sommerville 2007] I. Sommerville. Software engineering. International computer science series. Addison-Wesley, 2007.

[Song et al. 2007] X. Song, M. Wu, C. Jermaine, and S. Ranka. Conditional anomaly detection. IEEE Trans. on Knowl. and Data Eng., 19(5):631–645, May 2007.

[Starke and Hruschka 2011] G. Starke and P. Hruschka. Software-Architektur kompakt - angemessen und zielorientiert. Spektrum Akademischer Verlag, Heidelberg, 1 edition, 2011.

[van Hoorn et al. 2009] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous monitoring of software services: Design and application of the Kieker framework. Technical Report TR-0921, Department of Computer Science, University of Kiel, Germany, Nov. 2009.

[van Hoorn et al. 2012] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012). ACM, Apr. 2012. To appear.

[van Kesteren 2012] A. van Kesteren. XMLHttpRequest level 2. World Wide Web Consortium, Working Draft WD-XMLHttpRequest2-20080930, January 2012.

[Wang et al. 2008] X. Wang, L. W. 0001, and A. Wirth. Pattern discovery in motion time series via structure-based spectral clustering. In CVPR. IEEE Computer Society, 2008.

[Weber 2010] S. Weber. Nosql databases. December 2010. URL http://wiki.hsr.ch/Datenbanken/files/Weber_NoSQL_Paper.pdf.

[Wei et al. 2005] L. Wei, N. Kumar, V. N. Lolla, E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana. Assumption-free anomaly detection in time series. In J. Frew, editor, SSDBM, pages 237–240, 2005. URL http://dblp.uni-trier.de/db/conf/ssdbm/ssdbm2005.html#WeiKLKLR05.

[XING AG 2010] XING AG. Annual report of XING AG 2010, 2010. URL http://corporate.xing.com/fileadmin/image_archive/XING_AG_jahresergebnisse_2010_01.pdf.

[XING AG 2011] XING AG. Investor fact sheet of XING AG 2011, March 2011. URL http://www.equitystory.com/download/Companies/openbc/factsheet_1600_english.pdf.

[XING AG 2012] XING AG. XING corporate pages, March 2012. URL http://corporate.xing.com/xing-ag/.

[Xu et al. 2009] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Online system problem detection by mining patterns of console logs. In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, ICDM '09, pages 588–597, Washington, DC, USA, 2009. IEEE Computer Society.

[Yao et al. 2010] Y. Yao, A. Sharma, L. Golubchik, and R. Govindan. Online anomaly detection for sensor systems: A simple and efficient approach. Perform. Eval., 67(11):1059–1075, Nov. 2010.

Supporting Sources

All graphics were sketched with Omnigraffle Pro[5] with the use of some stencils provided by Graffletopia[6]. This document was compiled on May 16, 2012 at 1:30pm with LaTeX.

[5] http://www.omnigroup.com/products/omnigraffle/
[6] http://graffletopia.com/

October 2011 Till March 2012