# Self-Adaptive Software System Monitoring for Performance Anomaly Localization

Jens Ehlers, André van Hoorn, Jan Waller, Wilhelm Hasselbring
Software Engineering Group
Christian-Albrechts-University Kiel
24098 Kiel, Germany
{jeh, avh, jwa, wha}@informatik.uni-kiel.de

## ABSTRACT

Autonomic computing components and services require continuous monitoring capabilities for collecting and analyzing data of runtime behavior. Particularly for software systems, a trade-off between monitoring coverage and performance overhead is necessary.

In this paper, we propose an approach for localizing performance anomalies in software systems employing self-adaptive monitoring. Time series analysis of operation response times, incorporating architectural information about the diagnosed software system, is employed for anomaly localization. Comprising quality of service data, such as response times, resource utilization, and anomaly scores, OCL-based monitoring rules specify the adaptive monitoring coverage. This enables to zoom into a system's or component's internal realization in order to locate root causes of software failures and to prevent failures by early fault determination and correction.

The approach has been implemented as part of the Kieker monitoring and analysis framework. The evaluation presented in this paper focuses on monitoring overhead, response time forecasts, and the anomaly detection process.

## Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Monitors, Tracing*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, Reverse Engineering, and Reengineering*

## General Terms

Measurement, Performance

## 1. INTRODUCTION

Though runtime performance is a critical characteristic of software systems, monitoring their operation is often neglected in practice. Typically, monitoring probes are only instrumented in reaction to prior performance degradations or even system failures.

In contrast to profiling at construction time, continuous monitoring of operational systems has to consider keeping the monitoring overhead deliberately small. A trade-off between information quality and monitoring overhead has to be accepted. The overhead is not caused by the injection of probes at numerous join points in the control flow, but by the complexity of the probe implementations. A finding of our evaluation is that it is feasible to instrument probes at a variety of possibly relevant join points, as long as not all of them are active at the same time during operation. Hence, it is desirable to introduce an adaptable setting that prescribes which probes and join points are activated at a distinct point in time.

It is well-known that a major part of the failure recovery time is required to locate the root cause of a failure [6]. In this paper, we present a self-adaptive, rule-based monitoring approach that will reduce the potentially business-critical wait time that delays a failure or anomaly diagnosis, as it allows autonomic on-demand changes of the monitoring coverage at runtime. The monitoring rules should reflect previously specified goals. For many goals, it is sufficient to monitor performance at component level, e.g. to fulfill evidence of SLA compliance, to support capacity planning decisions, or to extract usage patterns for interface design. In the following, we concentrate on the goal to localize the causes of performance anomalies that effect a change in a system's normal behavior as perceived by its users. The proposed monitoring adaptation feature is implemented as an extension plugin of our Kieker framework[1] [8], which facilitates to monitor and to analyze the runtime behavior of software systems. We employ the Object Constraint Language (OCL)[2] to specify the monitoring rules. The rules refer to performance attributes, particularly responsiveness anomaly scores, that change their values during runtime. Self-adaptation is based on the continuous evaluation of the monitoring rules. For that purpose, we propose and evaluate an online anomaly rating procedure for the timing behavior of system-inherent operations. Our implementation is based on EMF (Eclipse Modeling Framework)[3] meta-models which allow to evaluate OCL query expressions on object-oriented instance models at runtime.

The remainder of this paper presents our approach for self-adaptive performance monitoring and the underlying


*ICAC'11,* June 14–18, 2011, Karlsruhe, Germany.


[1] `http://kieker.sourceforge.net/`
[2] `http://www.omg.org/spec/OCL/2.2/`
[3] `http://www.eclipse.org/modeling/emf/`

anomaly rating procedure in Section 2, its evaluation in Section 3, and a conclusion and future work in Section 4.

# 2. SELF-ADAPTIVE MONITORING FOR ANOMALY LOCALIZATION

The monitoring adaptation feature is integrated as a plugin into our Kieker monitoring framework being explicated in detail in [8]. Figure 1 illustrates the major components of the Kieker framework. As presented in our previous work [5], the underlying monitoring process is structured into the following activities: probe injection, probe activation, data collection, data provision, data processing, visualization, and (self-)adaption. The proposed Monitoring Adaptation Plugin is implemented as one of several plugins for the Kieker Analysis component.

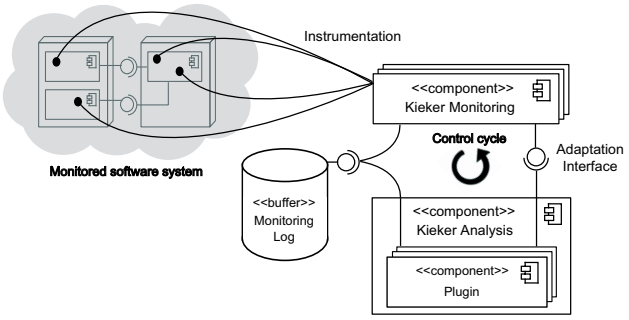


**Figure 1: Kieker monitoring architecture, cf. [5]**

Probes are spread across various join points in the control flow of a monitored system so that all operation executions can be monitored. For a system in productive operation, it is not possible to record monitoring data each time a probe is actuated as one of its join points is reached. The restriction not to activate all measuring points at the same time is caused not only by broad instrumentation, but particularly by extensive system workload (number of requests/s). Considering component-based systems, an appropriate initial coverage is to activate join points at operations that are part of a component's provided interface.

## 2.1 Rule-based Monitoring Adaptation

The Monitoring Adaptation Plugin allows the specification of monitoring rules which are evaluated continuously and may effect changes of the current monitoring coverage. The adaptation process works like a simple rule-based expert system. The performance engineer acts as the expert who defines the rules of inference. The premise of a rule specifies an expression that is evaluated for all measuring points. We employ the OCL to specify the rule premises:

$P_1$: **context** CallingContextTree: **self**.callingContexts→*select*(
(parent.op.monitoringActivated **and** parent.anomalyScore $> t$)
**or** level $= 1$)→*collect*(op)

The example rule premise $P_1$ is based on a calling context tree (CCT) model, as defined in [1]. $P_1$ selects all operations that are called from a caller operation that is already monitored (parent.op.monitoringActivated) and behaves anomalous in a particular calling context, i.e. the context's anomaly score exceeds a specified threshold $t$ (parent.anomalyScore $> t$). Additionally, all operations are added to the result set that are at the topmost level of the CCT (level $= 1$), i.e. system-level interface operations for incoming client requests.

The set of measuring points for which the premise evaluates to true is handled by the rule's conclusion. The conclusion is either to activate or to deactivate probes at the previously selected set of measuring points. As all Kieker Analysis plugins are based on EMF meta-models, we are able to utilize the EMF Model Query sub-project, which allows constructing and running queries on EMF models by means of OCL. Goal-oriented self-adaptation is based on the possibility to refer to attributes in the OCL expressions that change their values during runtime, e.g. performance metrics and derived anomaly scores. The rule specified above increases the monitoring coverage of a component's interior control flow if it does not behave as expected. In this way, our approach affords automatic on-demand adaptation of the effective software system monitoring.

## 2.2 Rating of Anomalous Responsiveness

Anomalies are patterns in the monitored data that do not conform to the expected behavior. Finding these nonconforming patterns, also called outliers, is referred to as anomaly detection [4]. We consider an anomaly to arise as a significant deviation between a measured observation and a previously expected forecast value, whereas forecasts are based on historic time series.

Factors impacting the timing behavior of a software service are particularly the usage profile of its provided interface, its deployment environment, its internal implementation, and the behavior of external services it depends on [2]. The concrete values of these factors determine the context in which a service is called. We try to capture and to separate different contextual values from each other as far as possible, as it mainly depends on the context if an observation has to be considered anomalous or not [4]. Given a specific implementation (including external services) and a deployment environment, a service has an expected response time distribution that usually depends on the input parametrization and the current system workload. Both are characteristics of the usage profile. Considering software services, the actual distribution model of response times is a priori unknown. Though we collect response times at a fine-grained contextual level at which distinct operations and their stack contexts are distinguished, this is not necessarily sufficient to expect deterministic response times as long as variations in parametrization and workload are not distinguished as well.

The number of concurrent sessions or threads and the CPUs' utilization can be taken as workload indicators. However, to measure an operation's input parametrization completely is not feasible in practice. It may not only be the input arguments of an operation signature that effect its response time, but also global component-internal parameters. The component-internal state is not transparent for a client initiating a service request. Nevertheless, it can have an impact on the control and data flow, and thus on the resulting resource demand. A compromise can be requiring a performance engineer to mark the performance-effecting service parameters in a design-oriented performance model as in [2]. As often the performance-relevant parameters are unknown in advance or a design-oriented performance model does not yet

exist, our anomaly rating procedure approaches a different solution. We assume that it is not possible to separate all context-determinant impact factors; thus, even from a fine-grained contextual viewpoint, response times can be arbitrarily distributed and do not necessarily converge to a parametric distribution model. Our anomaly rating procedure consists of four steps: (1) response time forecast, (2) anomalous behavior hypothesis test, (3) anomaly score calculation, and (4) anomaly score correlation and aggregation.

**(1) Response time forecast**: Response time observations being ordered in time form a univariate time series. A time series is a realization of a stochastic process, which is a sequence of time-indexed random variables. A basic assumption in time series analysis is that some patterns observed in the past will remain in the future. Hence, time series analysis can be used for forecast purposes based on historical data: $x_1, x_2, ..., x_t \rightarrow \hat{x}_{t+1}, \hat{x}_{t+2}, ...$ (observations $\rightarrow$ predictions). Forecasting based on a time series abstracts from any technical or economical interrelations. It is assumed that characteristic features of the underlying stochastic process can be recovered from the historic data.

We implemented and evaluated different common forecasting models for stochastic processes with respect to their ability to predict the response time of software services. Each forecasting model assumes the time series to be generated from a different underlying stochastic process. The forecast models being evaluated in Section 3 include single exponential smoothing (SES), double exponential smoothing (DES) according to Holt-Winters, and first-order autoregressive integrated moving average (ARIMA) processes, namely ARIMA(1,0,1) and ARIMA(1,1,1), according to Box-Jenkins. For further details concerning the foundations of time series analysis and these forecast models refer to [3].

**(2) Anomalous behavior hypothesis test**: For anomaly testing, it is leveraged that not every individual operation execution has to be classified and reported if it is considered to be an outlier. Instead, it is intended that a cohesive collection of suspicious operation executions is observed before reporting an anomaly. Our procedure benefits from the central limit theorem: Whenever a random experiment is replicated many times, the new variable, which equals the average result over the replicates, is likely to follow the normal distribution. A random sample is taken from all response time observations within a specified time interval. With a sample size of 10, the average response time is fairly normalized. In our evaluation, we used different sample sizes to bundle operation executions and test whether they form a collective anomaly.

To decide whether an anomaly is observed or not, a hypothesis test is conducted. It is tested whether the forecasted response time $\hat{x}_{t+1}$ can be accepted as the mean of the population $\mu$ from which the sample has been taken. The validity of the test rests on the assumption that the sample mean $\overline{x}$ is approximately normally distributed. The null hypothesis is $H_0 : \mu = \mu_0$ with $\mu_0$ being the forecasted response time $\hat{x}_{t+1}$. So, only if $H_0$ is rejected, the observed collection of response times is rated as anomalous. $H_0$ will be rejected if $\mu_0$ lies in a critical region outside a confidence interval. The range of the confidence interval depends on a specified significance level $\alpha$, which controls the rate of false positives (probability of type I errors). A susceptibility to false alarm due to a multitude of false positives reduces the efficacy of an automated approach. Frequent false alarms cause unnecessary effort and emotionally blunt an analyst, who has to investigate and discard each false positive. $H_0$ will be rejected if the value of the test statistic $t_0$ falls in the critical region defined by the lower and upper $\alpha/2$-percentage points of the corresponding t-distribution. If $t_0$ is in the confidence interval, the recent sample of response time observations $\vec{x}$ is associated with an anomaly value $a_{\vec{x}} = 0$. Otherwise, the sample is rated as anomalous, i.e. $a_{\vec{x}} = 1$.

**(3) Anomaly score calculation**: It is not sufficient to save the observation time of each collective anomaly. Instead, it is desired to calculate a single numeric figure that represents the recent degree of an operation's timing behavior to be anomalous. Therefore, an anomaly scoring function $a$ is constructed that condenses the frequency and the trend of anomaly observations over time. For each operation, the function maintains an anomaly score between 0 and 1 where 0 indicates that response times have been as expected, and 1 indicates a completely anomalous timing behavior. Each sample of response times $\vec{x}$ that is tested for anomaly slightly impacts the overall anomaly score of an operation $a_{op}$, which is calculated recursively by exponential smoothing as follows: $a_{op,t+1} = \beta\ a_{\vec{x}} + (1 - \beta)\ a_{op,t}$. The smoothing parameter $\beta$ determines how sensible the scoring function reacts. If anomalies are detected frequently, the score increases. Otherwise, it will slowly decrease.

**(4) Anomaly score correlation and aggregation**: The operation-level anomaly scores are aggregated to higher levels of abstraction such as component-level anomaly scores. Aggregation is done via weighted averages based on operation call frequencies. Further, we provide the option to adjust the anomaly scores by applying a correlation algorithm. On each level of abstraction, the system entities such as operations, classes, or components are represented as nodes of a call graph. The graph's directed edges represent the calling actions or dependencies among the system entities. It is commonly assumed that anomalies are propagated backwards through the call graph [7], i.e. if a node indicates anomalous behavior, then the anomaly is partially propagated to the node's callers. Correlation algorithms perform a negation of the propagation effects to identify the root cause of an anomaly.

# 3. EVALUATION

In our previous work, we reported on how we employed the Kieker Monitoring component in the productive systems of a telecommunication company and a digital photo service provider [8]. These case studies confirmed the applicability and the robustness of our approach. In lab experiments, we obtained detailed evaluations concerning the monitoring overhead, the response time forecasts, and the anomaly detection process.

**Monitoring overhead**: Figure 2 presents our results for a specific platform containing measured cost rates caused by instrumentation ($\Delta_I$), data collection ($\Delta_C$),

and data logging ($\Delta_L$). The boxplots show that (1) instrumentation, i.e. stepping into previously woven, but inactive dummy probes, causes very little overhead ($\Delta_I$ is less than 1 $\mu$s) compared to (2) data collection and (3) logging, i.e. creating and persisting the monitoring records to a Monitoring Log ($\Delta_C$ and $\Delta_L$ are each about 4 $\mu$s). The overhead evaluation allows the conclusion that instrumentation of probes at multiple measuring points is not critical as long as data collection and logging can be (de)activated systemically.

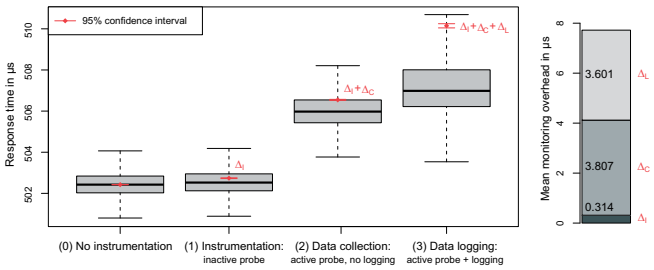


**Figure 2: Evaluation of the monitoring cost**

**Response time forecast models**: Besides, we evaluated the forecast models described in Section 2.2. We set up two experimental lab scenarios using the JPetStore[4] sample application and the SPECjEnterprise2010[5] industry standard benchmark as systems under test. In both scenarios, the number of concurrent users was constructed to provoke a close-to-reality workload with seasonal variation and trend. We monitored response times of selected services provided by the two test systems and applied the different forecast models. The time series of measured response times are forwarded to the statistic tool R, in order to utilize the functions HoltWinters and arima in the R stats package. Given a time series as input, these functions allow to calculate the best-fitting parameter values for the forecast models to be evaluated. The parameter values are used to determine the expected response times. Periodically, the parameter values are updated based on more recently observed time series. Table 1 summarizes our results concerning the forecast errors. The shown measures are mean average percentage errors being aggregated for all monitored services and different combinations of sample sizes and update intervals of the forecast model parameters. Compared to pragmatic forecast approaches $x_t$ (forecast is latest observed value) and $\bar{x}_t$ (forecast is average of preceding observations), the time series-based models provide more precise forecasts. The forecasts in scenario $E_1$ are better as than in scenario $E_2$, because the scenarios differ in the scheduling discipline of their major bottleneck ($E_1$: CPU with approx. processor sharing scheduling, $E_2$: database I/O with FIFO scheduling).

| Scenario | | SES | DES | ARIMA (1,0,1) | ARIMA (1,1,1) | $x_t$ | $\bar{x}_t$ |
|---|---|---|---|---|---|---|---|
| $E_1$ | JPetStore | 13,3% | 14,6% | 13,1% | 14,3% | 15,9% | 19,7% |
| $E_2$ | SPECjEnterprise2010 | 23,9% | 22,9% | 23,8% | 18,0% | 22,8% | 50,3% |

**Table 1: Evaluation of forecast model errors**

[4] http://sourceforge.net/projects/ibatisjpetstore/
[5] http://www.spec.org/jEnterprise2010/

**Anomaly detection process**: In the next evaluation step, we demonstrate that consecutive divergences of forecasts and observations are an indication of anomalous timing behavior. We injected faults into the test systems and observed how our self-adaptive monitoring approach zooms into a component by activating monitoring for those operations that are affected by a fault and thus are rated as anomalous. Detailed evaluation results are left out due to space restrictions. A manual exploration of the cause-and-effect chains is much more time-consuming and error-prone than an automated processing. A major contribution of the self-adaptive monitoring approach is to save this time and effort.

# 4. CONCLUSIONS AND FUTURE WORK

Responsiveness and scalability of productive software systems have to be observed and analyzed continuously. Our self-adaptive monitoring approach allows zooming into a component on demand if it behaves anomalous. In this case, zooming means to activate more (or less) measuring points in the application-level control flow aiming at increasing (or decreasing) insight, e.g. into the operation call stack, effective loop iterations, or conditional branches taken. For self-adaptive control of the monitoring coverage a set of OCL-based monitoring rules is proposed.

Moreover, we proposed an approach for self-adaptive software system monitoring based on the continuous evaluation of OCL-based monitoring rules at runtime. Further, we explicated an underlying anomaly rating procedure for the timing behavior of software systems. In our evaluation, we quantified the monitoring overhead and studied the practicability of the presented forecast models.

In future work, we will evaluate the adaptive monitoring approach in industrial case studies. Besides, we intend to integrate and to evaluate alternative anomaly detection procedures not being based on time series analysis.

# 5. REFERENCES


[1] G. Ammons, T. Ball, J. R. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. *ACM SIGPLAN Notices*, 32(5):85–96, 1997.

[2] S. Becker, H. Koziolek, R. Reussner. The Palladio component model for model-driven performance prediction. *JSS*, 82(1):3–22, 2009.

[3] G. E. P. Box, G. M. Jenkins, G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, 4th ed., 2008.

[4] V. Chandola, A. Banerjee, V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

[5] J. Ehlers W. Hasselbring. Self-adaptive software performance monitoring. In *Software Engineering 2011*, LNI, pages 51–62. GI, 2011.

[6] E. Kiciman A. Fox. Detecting application-level failures in component-based internet services. *IEEE Trans. on Neural Networks*, 16(5):1027 –1041, 2005.

[7] M. Steinder A. S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004.

[8] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, D. Kieselhorst. Continuous monitoring of software services: Design and application of the Kieker framework. Technical Report TR-0921, Dept. of Computer Science, University of Kiel, 2009.