van Hoorn, A., Waller, J. and Hasselbring, W. (2012) Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis [Paper] In: 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE'12), April 22-25, 2012, Boston, Massachusetts, USA.

Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis

— Invited Demo Paper —

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

ABSTRACT

Kieker is an extensible framework for monitoring and analyzing the runtime behavior of concurrent or distributed software systems. It provides measurement probes for application performance monitoring and control-flow tracing. Analysis plugins extract and visualize architectural models, augmented by quantitative observations. Configurable readers and writers allow Kieker to be used for online and offline analysis. This paper reviews the Kieker framework focusing on its features, its provided extension points for custom components, as well the imposed monitoring overhead.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Monitors, Tracing*; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*

1. INTRODUCTION

Application-level monitoring and dynamic analysis of software systems are a basis for various quality-of-service evaluation and reverse-engineering tasks. Example use cases include the diagnosis of SLO violations, online capacity management, as well as performance model extraction and calibration. The Kieker framework provides monitoring, analysis, and visualization support for these purposes.

Kieker development started in 2006 as a small tool for monitoring response times of Java software operations. Since then, Kieker has evolved into a powerful and extensible dynamic analysis framework for Java-based systems, including, e.g., model extraction and visualization support for dependency graphs, sequence diagrams etc. [1]. Kieker has been been used for dynamic analysis of production systems in industry [1, 2]. Recently, monitoring adapters for additional platforms, such as .NET and COM, have been added. In 2011, Kieker was reviewed, accepted, and published as a recommended tool by the SPEC Research Group. Since then, the tool is also distributed as part of SPEC RG's tool repository at `http://research.spec.org/projects/tools.html`.

ICPE'12, April 22–25, 2012, Boston, Massachusetts, USA

2. FRAMEWORK AND CORE FEATURES

The Kieker framework is structured into a monitoring and an analysis part [1]. On the monitoring side, *monitoring probes* collect measurements represented as *monitoring records*, which a *monitoring writer* passes to a configured *monitoring log or stream.* On the analysis side, *monitoring readers* import monitoring records of interest from the monitoring log/stream and pass them to a configurable pipe-and-filter architecture of *analysis plugins.* For the mentioned components, Kieker already includes a number of implementations, summarized below. Given the framework's extensibility, custom components can be developed easily, if required.

Focusing on application-level monitoring, Kieker includes monitoring probes for collecting timing and trace information from distributed executions of software operations. Additionally, probes for sampling system-level measures, e.g., CPU utilization and memory usage, are included. Kieker supports monitoring logs and streams utilizing file systems, databases, as well as JMS and JMX queues. A number of plugins for reconstructing, analyzing, and visualizing software architectural models such as calling-dependency graphs, call trees, and sequence diagrams are included.

3. DYNAMIC ANALYSIS WORKFLOW

Running a dynamic analysis with Kieker requires the instrumentation of the software system, as well as the specification of the monitoring and analysis configuration. Fig. 1 illustrates a typical dynamic analysis workflow with Kieker.

Components of the software system need to be instrumented with *monitoring probes.* Usually, existing monitoring probes and record types are sufficient, but depending on the required measurement data and involved technologies, custom monitoring probes and monitoring record types may be implemented.

In most cases, it is not necessary to implement custom monitoring writers and readers. File system (with comma-separated files) or database writers enable direct access to the *monitoring log/stream* and collected *monitoring records* can be analyzed using standard spread-sheet or statistics tools. Additionally, collected or queued records can be transferred to an *analysis configuration.*

On the analysis side, a configuration of monitoring readers and analysis plugins needs to be defined, using the included pipe-and-filter framework. Once defined, these configurations can be executed with Kieker to analyze previously collected (offline) or incoming (online) monitoring records and to produce textual output or graphical visualizations.

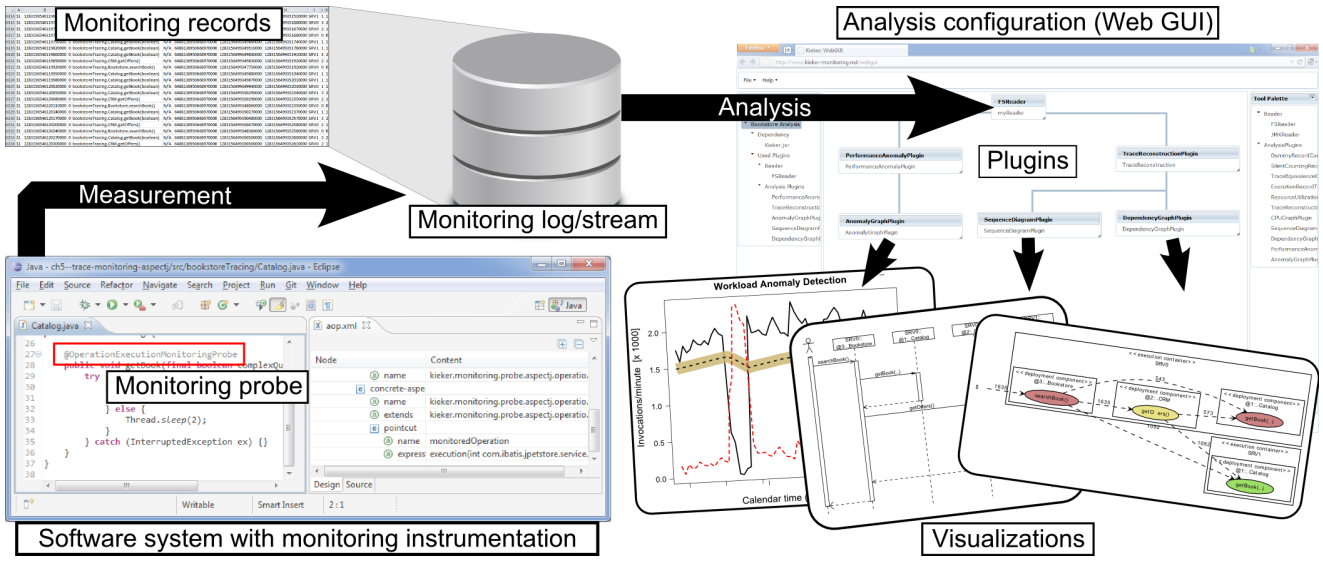
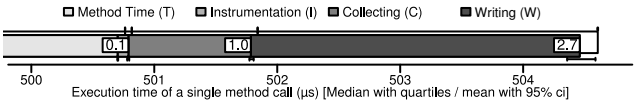


Figure 1: Illustration of a typical dynamic analysis workflow utilizing Kieker

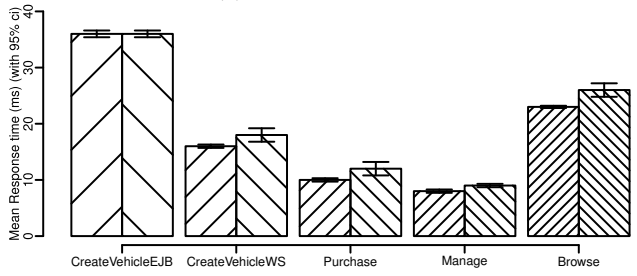
4. MONITORING OVERHEAD

Monitoring of software systems imposes a performance overhead. We performed extensive micro- and macro-benchmarks to quantify this impact on Java applications. In Fig. 2, we present the results of such a micro-benchmark to determine the three portions of the monitoring overhead [1] and of such a macro-benchmark to quantify the overhead of monitoring a typical enterprise system, represented by the SPECjEnterprise™ 2010 industry benchmark.[1]

On a typical enterprise server machine, our micro-benchmarks for a single method call reveal a median overhead of 0.1 µs for instrumentation, 1.0 µs to collect performance data, and 2.7 µs to write the collected data to a file system (Fig. 2 (a)). These results scale linearly with additional monitored method calls. The macro-benchmark simulates a typical Java EE application with 40 instrumented classes and 138 instrumented methods, accessed by approximately 260 concurrent threads. A comparison of the average response times without and with active Kieker monitoring reveals the average overhead at below 10% (Fig. 2 (b)).



(a) micro-benchmark



(b) macro-benchmark

Figure 2: Summary of monitoring overhead results

5. CONCLUSION & FUTURE WORK

Kieker is developed and employed for various purposes in research, teaching, and practice. Application areas include: performance evaluation, self-adaptation control (e.g., online capacity management), problem localization, simulation (replaying workload traces for driving simulations; measurement and logging of simulation data; analysis of simulation results), and software reverse engineering (e.g., extraction of architectural and usage models). In these contexts, our future work includes the development of additional analysis plugins, e.g., improved support for analyzing concurrent behavior. In addition to Java, .NET, and COM, we are working on monitoring support for other platforms, such as COBOL. Currently, we are developing a Web-based user interface for configuring and running dynamic analyses with Kieker. A road map is provided on the Kieker home page [3]: `http://www.kieker-monitoring.net`

References

[1] 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. TR-0921, Dept. of Computer Science, Univ. of Kiel, Germany, 2009.

[2] M. Rohr, A. van Hoorn, W. Hasselbring, M. Lübcke, and S. Alekseev. Workload-intensity-sensitive timing behavior analysis for distributed multi-user software systems. In *Proc. Joint WOSP/SIPEW Int. Conf. on Perf. Eng. (WOSP/SIPEW '10)*, pages 87–92. ACM, 2010.

[3] Kieker home page. `http://www.kieker-monitoring.net`.

This work is partly funded by the German Federal Ministry of Education and Research (BMBF) under grant number 01IS10051.

[1] SPECjEnterprise is a trademark of the Standard Performance Evaluation Corp. (SPEC). The SPECjEnterprise2010 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjEnterprise2010 is located at `http://www.spec.org/jEnterprise2010/`.