Reverse Engineering of Dependency Graphs via Dynamic Analysis

[Invited talk]

Wilhelm Hasselbring
University of Kiel
Institute of Computer Science
Software Engineering Group
D-24118 Kiel, Germany
wha@informatik.uni-kiel.de

ABSTRACT

Reverse engineering of software systems often employs static analysis of a program's source code. In this invited talk, I will present our approach to reverse engineering of software systems via analyzing monitoring data of a programs operational use; thus, via dynamic analysis. Our Kieker monitoring framework generates dependency graphs from observed monitoring data. It is used in several industrial cooperations on which I'll report.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Distributed/Internet based software engineering tools and techniques*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*; D.2.8 [**Software Engineering**]: Metrics/Measurement—*Performance measures*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Frameworks*; D.4.8 [**Operating Systems**]: Performance—*Measurements Monitors*; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Software Engineering

Keywords

Reverse engineering, Monitoring, Dynamic analysis, Dependency Graphs

1. DYNAMIC ANALYSIS WITH THE KIEKER MONITORING FRAMEWORK

The object-oriented Kieker monitoring framework has been designed for continuous monitoring of software systems. The framework components for software instrumentation, logging, and analysis/visualization are extensible and may easily be replaced to fulfill the requirements of specific project contexts. For instance, as a non-intrusive instrumentation technique, we employ aspect-oriented programming (AOP). Kieker uses a common data structure for monitoring records in all components that produce or consume monitoring data. For analysis of monitoring data, Kieker provides several visualizations of a system's runtime behavior, such as UML sequence diagrams, and dependency graphs. These models are extracted from recorded application-internal traces originating from system-provided services. The analysis may be performed online or offline. Kieker supports distributed request tracing, since the service-providing components of large-scale software systems are usually distributed across several execution containers on physical or virtual server nodes.

In our research, we employ Kieker for various purposes, e.g., fault localization based on timing behavior anomaly detection [3], architecture-based runtime adaptation / reconfiguration [8, 4], visualization of software runtime behavior [7], application-level intrusion detection [2], and trace-based performance analysis [5, 6].

Kieker is structured into the two main components Kieker.Monitoring and Kieker.Analysis with the Monitoring Log in between, as illustrated in Figure 1. Kieker.Monitoring provides a reusable infrastructure for collecting application-level monitoring data in Monitoring Probes and writing this monitoring data to the Monitoring Log, e.g., the local file system, a database, or a messaging queue, using a Monitoring Log Writer. The Monitoring Controller is responsible for initializing and controlling a Kieker.Monitoring instance. The Monitoring Log contains Monitoring Records, each holding the monitoring data of a single measurement created by the Monitoring Probes. Kieker.Analysis provides the infrastructure for analyzing the Monitoring Log: a Monitoring Log Reader (Figure 1) reads Monitoring Records from the Monitoring Log and delivers these to registered Monitoring Record Consumers, according to the observer design pattern [1]. Monitoring Record Consumers perform the actual analysis or visualization functionality. A Kieker.Analysis instance is initialized and controlled by an Analysis Controller instance (Figure 1).

Figure 2 shows an example dependency graph generated by Kieker.TraceAnalysis, visualizing calling dependencies among classes and their operations. This figure provides an ag-

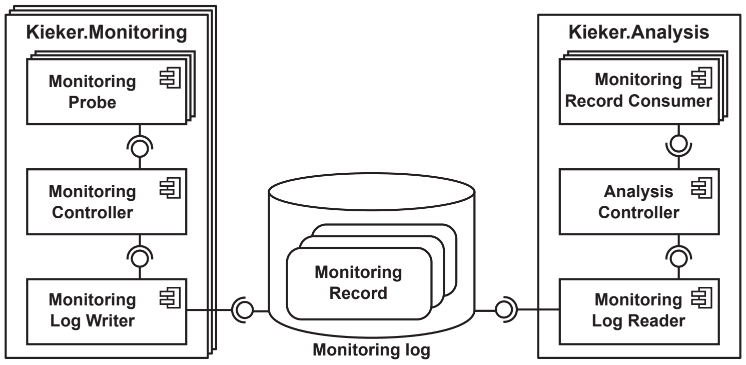

Figure 1: Top-level view on Kieker's architecture

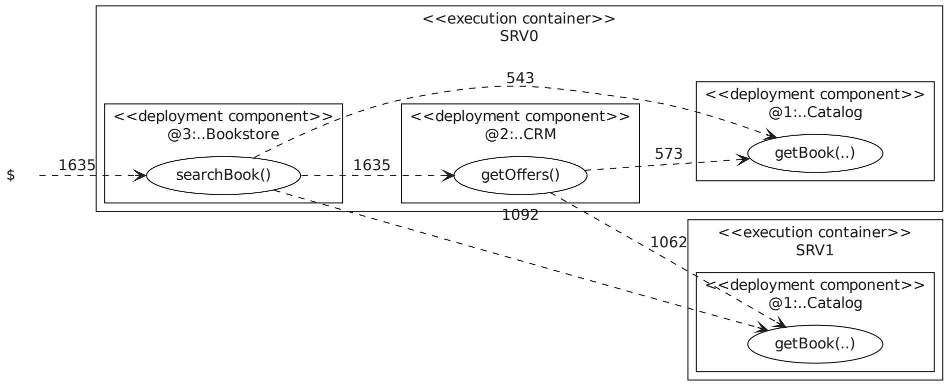

Figure 2: Generated deployment-level operation dependency graph

gregated view of the runtime dependencies as observed in 1635 traces.

Kieker is open-source software. For more information on Kieker, please refer to

`http://kieker.sourceforge.net/`

2. REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[2] I. A. Gul, N. Sommer, M. Rohr, A. van Hoorn, and W. Hasselbring. Evaluation of control flow traces in software applications for intrusion detection. In *Proceedings of the 12th IEEE International Multitopic Conference (IEEE INMIC 2008)*, pages 373–378. IEEE, 2008.

[3] N. S. 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. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009)*, pages 47–57. IEEE Computer Society, Mar. 2009.

[4] J. Matevska and W. Hasselbring. A scenario-based approach to increasing service availability at runtime reconfiguration of component-based systems. In *Proceedings of 33rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 137–144. IEEE Computer Society, Aug. 2007.

[5] M. Rohr, A. van Hoorn, S. Giesecke, J. Matevska, W. Hasselbring, and S. Alekseev. Trace-context sensitive performance profiling for enterprise software applications. In *Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08)*, volume 5119 of *Lecture Notes in Computer Science*, pages 283–302. Springer, June 2008.

[6] 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 *1st Joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW 2010)*. ACM, Jan. 2010. To appear.

[7] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stöver, S. Giesecke, and W. Hasselbring. Kieker: Continuous monitoring and on demand visualization of Java software behavior. In *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE 2008)*, pages 80–85. ACTA Press, Feb. 2008.

[8] A. van Hoorn, M. Rohr, A. Gul, and W. Hasselbring. An adaptation framework enabling resource-efficient operation of software systems. In *Proceedings of the 2nd Warm-Up Workshop for ACM/IEEE ICSE 2010 (WUP '09)*, pages 41–44. ACM, Apr. 2009.