Application Performance Monitoring with Kieker: Failure detection, capacity management, reverse engineering and more

Wilhelm Hasselbring, Reiner Jung

Software Engineering Group, Kiel University
http://se.informatik.uni-kiel.de/

with contributions by
Till Bielefeld (Voltremote), Nils Ehmke, Florian Fitkau, Sören Frey, André van Hoorn (Uni Stuttgart), Holger Knoche (b+m), Matthias Rohr (BTC), Jan Waller, Christian Wulf, and others

SPP Design for Future, Aachen, March 1st, 2013
Agenda

1. Monitoring for Dynamic Analysis
2. Kieker Monitoring & Analysis Framework
3. Failure Detection
4. Reverse Engineering
5. Capacity Management
6. Kieker Hands On
7. Summary and Outlook
Various approaches with varying objectives, some examples:

- **Code Metrics**: JavaNCSS
- **Coding rules**: Checkstyle
- **Bug patterns**: FindBugs
- **Clon detection**: Bauhaus
- **Architecture analysis**: Sotograph

Static analysis may be applied directly on the program code.
Dynamic Analysis of Runtime Behavior

Monitoring for Dynamic Analysis

- Static analysis may assess the structure of software systems,
  - but the **actual** runtime behavior may only be assessed via dynamic analysis.
- Profiling (during development) allows for dynamic analysis,
  - but usually not under real workload conditions.
- Monitoring (during operation) allows for continuous dynamic analysis
  - of the **actual** runtime behavior under **real** workload conditions.

- Appreciating the requirements of **operating** software systems is required while constructing software systems
  - Continuous monitoring is such a requirement.
Monitoring on different system layers

Managing Business Processes, Services, Application, and Middleware

Application Monitoring

System Monitoring

Key performance indicators, e.g., process throughput time, ...

SLO compliance, workload, ...

Response times, operational profile, ...

Connection pool size, ...

Thread pool size, heap size, ...

CPU/Memory utilization, ...

Availability, reliability, ...

Available Infrastructure

Business Processes

Services

Application

Middleware

Container

Operating System

Hardware Infrastructure
Scaling Facebook to 500 Million Users and Beyond

“Making lots of small changes and watching what happens only works if you’re actually able to watch what happens. At Facebook we collect an enormous amount of data - any particular server exports tens or hundreds of metrics that can be graphed. This isn’t just system level things like CPU and memory, it’s also **application level statistics** to understand why things are happening.

It’s important that the statistics are from the **real production machines** that are having the problems, when they’re having the problems – the really interesting things only show up in production. The stats also have to come from all machines, because a lot of important effects are hidden by averages and only show up in distributions, in particular 95th or 99th percentile.”

Robert Johnson, Facebook Engineering Director
Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis

www.kieker-monitoring.net

Continuous Application Performance Monitoring for Online and Offline Analysis

- Static analysis is not sufficient to study the internal behavior of software systems comprehensively
- Continuous monitoring allows to gather a system’s actual runtime behavior resulting from production usage profiles
- Performance evaluation (e.g., bottleneck detection)
- Self-adaptation control (e.g., capacity management)
- Application-level failure detection and diagnosis
- Simulation (workload, measurement, logging, and analysis)
- Software maintenance, reverse engineering, modernization
- Service-level management

Instrumentation and Monitoring Overhead

- Includes probes for collecting timing, control flow, and resource utilization
- Support for various Java instrumentation methods, e.g., AspectJ, middleware interception, Servlet filters
- Micro-benchmarks revealed low overhead
- Each activation adds constant overhead (linear scaling)

Framework Characteristics

- Modular, flexible, and extensible architecture
- Extensible probes, readers, writers, records, and plugins
- Integrated monitoring record type model for monitoring and analysis
- Allows to log, reconstruct, analyze, and visualize distributed traces
- Designed for continuous operation in multi-user systems
- Evaluated in lab experiments and industrial case studies (since 2006)
- Kieker is open-source software (Apache License, V. 2.0)

Pipe-and-Filter Configuration for Analysis/Visualization

Invited Tool Demo @ ICPE 2012

Current Activities/Coming Soon
- Monitoring adapters for .NET, VBA, COBOL etc.
- Model-driven instrumentation & analysis
- Web-based graphical user interface
- Plugins for analysis of concurrent behavior

Kieker is distributed as part of SPEC® RG’s repository of peer-reviewed tools for quantitative system evaluation and analysis.
Kieker Monitoring & Analysis Framework


Kieker Monitoring & Analysis Framework

Introduction

- Modular, flexible, and extensible architecture (Probes, records, readers, writers, filters etc.)
- Pipes-and-filters framework for analysis configuration
- Distributed tracing (Spring, SOAP, etc.)
- Low overhead (designed for continuous operation)
- Evaluated in lab and industrial case studies

Kieker is Open-Source Software (Apache License, V. 2.0)

http://kieker-monitoring.net

Recommended Tool of the SPEC Research Group

Kieker is distributed as part of SPEC RG’s repository of peer-reviewed tools for quantitative system evaluation and analysis,

http://research.spec.org/projects/tools.html
Kieker’s Evolution 2006–2012

Kieker Monitoring & Analysis Framework  ▶  Introduction

versions 0.5/0.6
- 1st Sourceforge releases
- Monitoring Component - Tpm
- DB and FS Writer (sync&async)
- 12 Java classes
- 2 AspectJ probes
- (Analysis component - Tpan - not part of the releases)
version 0.91
- Distributed monitoring
- Servlet, CXF, and Spring probes
- 36 Java classes
Kieker’s Evolution 2006–2012

Kieker Monitoring & Analysis Framework  

Introduction

- Major architectural changes (2009 TR)
- Generalized/extensible monitoring record model
- Generalized/extensible writer/reader model
- First release containing parts of the analysis component
- Micro-benchmark
- Version 1.0: 81 Java class files + 28 Java test files

Hasselbring, Jung (CAU)  
Application Performance Monitoring with Kieker  
SPP Design for Future  
9 / 52
Kieker Monitoring & Analysis Framework  

Introduction

- Version 1.1
  - TraceAnalysis tool
  - System meta-model
  - On-the-fly trace reconstruction
  - 141 Java classes

- Version 1.2
  - User Guide

Hasselbring, Jung (CAU)  
Application Performance Monitoring with Kieker  
SPP Design for Future
Kieker’s Evolution 2006–2012

Kieker Monitoring & Analysis Framework ▶ Introduction

version 1.3
- Various new features
- Improved configuration management
- Initial version submitted to SPEC RG
- 205 Java classes + 42 test classes
version 1.5
- Completely reworked pipes-and-filters framework
- Event-based tracing
- 284 Java classes + 92 test classes
Kieker’s Evolution 2006–2012

Kieker Monitoring & Analysis Framework → Introduction

- Adaptive Monitoring
- WebGUI (beta)
- 375 Java classes + 147 test classes

version 1.6

- Adaptive Monitoring
- WebGUI (beta)
- 375 Java classes + 147 test classes

Hasselbring, Jung (CAU)
Application Performance Monitoring with Kieker
Kieker’s Evolution 2006–2012

Kieker Monitoring & Analysis Framework

Introduction

2006
May Sept.

2007
May Sept.

2008
May Sept.

2009
May Sept.

2010
May Sept.

2011
May Sept.

2012
May Sept.

iObserve

1.5

1.6

ESN SL

1.3

1.4

regular meetings

FindBugs

JUnit

SourceForge

cewe color

EWE

Nokia Siemens Networks

Performance Monitoring von
Middleware-basierten Anwendungen

0.5
0.6

0.91
0.95a
1.0

1.1
1.2

1.3
1.4
1.5
1.6

Hasselbring, Jung (CAU)

Application Performance Monitoring with Kieker

SPP Design for Future
Kieker: Monitoring Workflow

Kieker Monitoring & Analysis Framework → Architecture

Monitoring records → Measurement → Monitoring log/stream

Analysis configuration (Web GUI)

Analysis

Plugins

Visualizations

Software system with monitoring instrumentation

Hasselbring, Jung (CAU) Application Performance Monitoring with Kieker SPP Design for Future 10 / 52
Core Kieker Framework Components

Kieker Monitoring & Analysis Framework ➔ Architecture

### Monitoring Log/Stream

**Monitoring Probe** ➔ **Monitoring Controller** ➔ **Monitoring Writer**

**Time Source** ➔ **Logging** ➔ **Periodic Sampling** ➔ **JMX Interface**

**Monitoring**

**Java probes/samplers:**
- Manual instrumentation
- AspectJ
- Spring
- Servlet
- CXF/SOAP
- <your interception technology>
- CPU utilization
- Memory usage
- <your technology>
- <your monitoring probe>

**Kieker.Analysis**

**Pipe & Filter Configuration** ➔ **Monitoring Reader** ➔ **Analysis / Visualization Plugin** ➔ **Analysis Controller**

**Monitoring Records**
- Operation execution
- Control-flow events
- CPU utilization
- Memory/swap usage
- Resource utilization
- Current time
- <your monitoring record type>

### Java probes/samplers:
- C#/.NET
- Visual Basic 6/COM
- COBOL

### Basic adapters for:
- C#/.NET
- Visual Basic 6/COM
- COBOL

---

Hasselbring, Jung (CAU)

Application Performance Monitoring with Kieker

SPP Design for Future 11 / 52
Core Kieker Framework Components

Kieker Monitoring & Analysis Framework ➤ Architecture

- Monitoring Probe
- Monitoring Controller
- Monitoring Writer
- Monitoring Log/Stream
- Kieker.Analysis
  - Monitoring Reader
  - Analysis / Visualization Plugin

Monitoring Readers/ Writers:
- File system
- Java Messaging Service (JMS)
- Java Management Ext. (JMX)
- Database (SQL)
- Named pipe
- <your monitoring reader/writer>
Core Kieker Framework Components

Kieker Monitoring & Analysis Framework → Architecture

- **Kieker.Monitoring**
  - Monitoring Probe
  - Monitoring Controller
  - Monitoring Writer
  - JMX Interface
  - Periodic Sampling
  - Logging
  - Time Source

- **Monitoring Log/Stream**
  - Monitoring Record

- **Kieker.Analysis**
  - Monitoring Reader
  - Analysis / Visualization Plugin
  - Pipe & Filter Configuration

**Analysis / Visualization Plugin**
- Dependency graphs
- Sequence diagrams
- Call graphs
- <your visualization>
- <your trace analysis>
- <your reconstruction plugin>
- <your analysis plugin/tool>

Hasselbring, Jung (CAU)
Communication Among Kieker Components

Kieker Monitoring & Analysis Framework ▶ Architecture

1.2: newMonitoringRecord(record)

IMEonitoringProbe

: MonitoringController

2: newMonitoringRecord(record)

: IMonitoringWriter

3: toArray()

1: record = new MonitoringRecordImpl(...)

: IMonitoringRecord

: AnalysisController

1 4.2: inputRecord(record)

: IMonitoringReaderPlugin

: IFilterPlugin

1..* 1..*

1

1

* * * * * ...

4.1: record = initFromArray(...)

: IMonitoringProbe

: IMonitoringWriter

: IMonitoringRecord
Instrumentation — Getting the monitoring probe into the code

1. Manual instrumentation
2. Aspect-Oriented Programming (AOP), middleware interception, …
In an execution environment, three components \(a\), \(b\), and \(c\) each provide services which are monitored by means of \(Tpmon\) using the AOP concept. \(Tpmon\) stores the monitored data into the database.

Component \(a\) calls operation \(b\) of component \(b\). This operation contains a point-cut defined by the annotation \(@TpmonMonitoringProbe\). As defined in the description of the respective aspect \(probeMethod\), \(Tpmon\) saves the current time before and after \(b\) is executed.

Figure 2.17: Sample system instrumented with \(Tpmon\) (a) and how an annotated operation is woven (b).

AOP — Aspect-oriented programming (following Kiczales et al. [1997])

Figure 2.18: An aspect weaver weaves the aspects and the functional part of an application into a single binary (following (Kiczales et al., 1997)).
AOP-Based Instrumentation by Annotation

Chapter 2 Foundations

TPMon Integration

(a) In a execution environment, three components a, b and c each provide services which are monitored by means of TPMon using the AOP concept. TPMon stores the monitored data into the database.

(b) Component a calls operation b of component b. This operation contains a point-cut defined by the annotation @TpmonMonitoringProbe. As defined in the description of the respective aspect probeMethod, TPMon saves the current time before and after b is executed.

Figure 2.17: Sample system instrumented with TPMon (a) and how an annotated operation is woven (b).

Figure 2.18: An aspect weaver weaves the aspects and the functional part of an application into a single binary (following (Kiczales et al., 1997)).

Annotation-based (AOP) instrumentation for monitoring trace information
Visualization of Calling Networks
Clustering the Monitoring Log [Zheng et al. 2011]
Kieker Monitoring & Analysis Framework ➔ Analysis Visualization

Community structure
Layered structure hierarchy

Xi’an Jiaotong University, Shaanxi [Zheng et al. 2011]
is later uploaded into the graph database. For the purpose of the proof-of-concept dynamic data acquisition, a tool called Kieker [17] was used. Kieker is a Java framework for application performance monitoring and dynamic software analysis. The upload process first took the software project source code as an input, and Mikroskop retrieved information on those architectural artifacts (like packages, classes, methods, variables) and their relationships (like methods owned by classes, classes belonging to packages) that could be determined statically. Another stream of facts about the software project was uploaded dynamically when execution of the software project was eavesdropped using Kieker. This made it possible to collect data on dynamic relationships between software artifacts (e.g., count the numbers of executed calls).

**4 Software Intelligence**

In previous section I defined an architecture warehouse that contained architectural facts, that is architectural artifacts and their dependencies that were acquired statically and dynamically. Let software intelligence denote now a toolset that makes it possible to access the facts using data visualisation and data analysis techniques, hence obtaining a higher level of understanding of those facts, that is obtaining knowledge. In this section I deepen the definition by several examples.

**Fig. 2.** Graph-based visualisation of artifacts in JUnit project [Dąbrowski 2012]

Example 9. Figure 2 depicts the graph of all artifacts collected for JUnit software in two different visualisations.

Visualisation of architectural knowledge may be a challenge due to the size of the data set. Such challenges are not unique in computer science, i.e., disciplines like computational biology have already developed libraries, frameworks...
3D Visualization
[Döhring 2012], based on Wulf [2010]
Kieker Monitoring & Analysis Framework ➔ Analysis Visualization
Overhead Evaluation
[Waller and Hasselbring 2012]

Kieker Monitoring & Analysis Framework

Overhead Evaluation

Experiment similar to:
- A5 AsyncFS writer
- 16 cores
- whole system is available
Agenda

1. Monitoring for Dynamic Analysis
2. Kieker Monitoring & Analysis Framework
3. Failure Detection
4. Reverse Engineering
5. Capacity Management
6. Kieker Hands On
7. Summary and Outlook

Hasselbring, Jung (CAU)

Application Performance Monitoring with Kieker

SPP Design for Future

20 / 52
Kieker Analysis example pipes-and-filters configuration
- Performance anomaly detection and visualization
- Architecture and trace reconstruction/visualization
Performance Anomaly Detection
A typical example for using a monitoring framework

Realized and deployed at Xing AG [Bielefeld 2012].
Agenda

Reverse Engineering

1. Monitoring for Dynamic Analysis
2. Kieker Monitoring & Analysis Framework
   - Introduction
   - Architecture
   - Program Instrumentation
   - Analysis Visualization
   - Overhead Evaluation
3. Failure Detection
4. Reverse Engineering
   - Reverse Engineering of Java EE
   - Reverse Engineering von C#
   - Reverse Engineering of COBOL
   - Reverse Engineering of VB 6
5. Capacity Management
6. Kieker Hands On
7. Summary and Outlook
Kieker Analysis example pipes-and-filters configuration

- Performance anomaly detection and visualization
- Architecture and trace reconstruction/visualization
Web-based GUI
for Configuring the Analysis Plugins
Reverse Engineering

Kieker » Bookstore-Example

Available Plugins:
- WebReader
- FileReader

Properties:
- Classname: Kieker examples.usergui.ch3and4bookstore.MyResponseTimeFilter
- Name: MyResponseTimeFilter
- ThresholdNanos: 1000000

InvalidExecutionTraceWriterFilter (kieker.tools.traceAnalysis.filter.traceWriter.InvalidExecutionTraceWriterFilter)
A filter allowing to write the incoming InvalidExecutionTrace into a configured file

Input Ports:
- invalidExecutionTraces

Repository Ports:
- system/node/repository

Configuration:
- outputFifo

Hasselbring, Jung (CAU)

Application Performance Monitoring with Kieker

SPP Design for Future
Reverse Engineering of Java EE
Customer portal of EWE TEL GmbH

Reverse Engineering ▸ Reverse Engineering of Java EE

Distributed Enterprise Java System

Servlet, Spring and CXF/SOAP probes
Software architecture level
Component dependency graph
Reverse Engineering  ▸ Reverse Engineering of Java EE
Reverse Engineering of C#
Complete Test Suite for Nordic Analytics [Magedanz 2011]
Reverse Engineering ▶ Reverse Engineering von C#

Case study at HSH Nordbank AG.
After selecting one use case
Reverse Engineering of C#
Reverse Engineering ⊳ Reverse Engineering von C#
Reverse Engineering of COBOL
With consideration of non-instrumented modules [Knoche et al. 2012]

Reverse Engineering

IDENTIFICATION DIVISION.
PROGRAM-ID. MODULE-X.

PROCEDURE DIVISION.
    CALL "MODULE-Y".
    GOBACK.

AOP-based COBOL instrumentation

Module-level call dependency graph with assumed dependencies

Case study of b+m Informatik AG (Kieker extensions by Holger Knoche):

1. Industrial context with more than 1,000 COBOL modules.
2. 140,351 probes with 14 probe types.
Abbildung 6.5: Komponentenabhängigkeits-Modell der Menü-Navigation; Extrahiert aus den Monitoringdaten

6.4 Ergebnisse der Evaluation

Hasselbring, Jung (CAU)  
Application Performance Monitoring with Kieker  
SPP Design for Future 34 / 52
Case study at Dataport
(Kieker extensions by Holger Knoche).
Detection of unused code
Reverse Engineering of Visual Basic 6
Reverse Engineering → Reverse Engineering of VB 6
Agenda

1. Monitoring for Dynamic Analysis
2. Kieker Monitoring & Analysis Framework
3. Failure Detection
4. Reverse Engineering
5. Capacity Management
6. Kieker Hands On
7. Summary and Outlook
Online Capacity Management

SLAstic Framework (Ph.D. thesis of André van Hoorn)

Capacity Management

instrumented, runtime reconfigurable s/w system

SLAstic

MONITORING

SLAstic

RECONFIGURATION

SLAstic.CONTROL

online adaptation engine

Hasselbring, Jung (CAU)
Integrated Observation and Modeling Techniques to Support Adaptation and Evolution of Software Systems

Models@Runtime for Managed Software Evolution.
Agenda

1 Monitoring for Dynamic Analysis
2 Kieker Monitoring & Analysis Framework
3 Failure Detection
4 Reverse Engineering
5 Capacity Management
6 Kieker Hands On
7 Summary and Outlook
public class CRM {
    private final Catalog catalog;

    public CRM(final Catalog catalog) {
        this.catalog = catalog;
    }

    public void getOffers() {
        this.catalog.getBook(false);
    }
}


private static final IMonitoringController MONITORING_CONTROLLER = MonitoringController.getInstance();

public void getOffers() {
    final long tin = MONITORING_CONTROLLER.getTimeSource().getTime();
catalog.getBook(false); // <-- the monitored execution
    final long tout = MONITORING_CONTROLLER.getTimeSource().getTime();

    final OperationExecutionRecord e = new OperationExecutionRecord(
        "public void Catalog.getBook(boolean)",
        NO_SESSION_ID,
        NO_TRACEID,
        tin, tout, "myHost",
        NO_EOI_ESS,
        NO_EOI_ESS);
    MONITORING_CONTROLLER.newMonitoringRecord(e);
}
AOP Konfiguration META-INF/aop.xml

```xml
<!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN" "http://www.aspectj.org/dtd/aspectj_1_5_0.dtd">
<aspectj>
  <weaver options="">
    <include within="*"/>
  </weaver>
  <aspects>
    <aspect name="kieker.monitoring.probe.aspectj.operationExecution.OperationExecutionAspectFull"/>
  </aspects>
</aspectj>
```

AspectJ

```
$ java -javaagent:lib/kieker-1.6_aspectj.jar -jar BookstoreApplication.jar 50
```
### Monitoring JPetStore

**Kieker Hands On**

**3-Layer Architecture**
- **Presentation layer:** Struts
- **Service layer:** POJOs
- **Persistence layer:** iBatis

#### Deployment
- **Application Server** (here: Jetty 7.6.7)
- **SQL DBMS Server** (here: HSQL)

---

**iBatis JPetStore 5.0**

(\texttt{http://ibatis.apache.org/})
**Datei** `webapps/jpetstore/WEB-INF/classes/META-INF/aop.xml`

```xml
<!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN" "http://www.aspectj.org/dtd/aspectj_1_5_0.dtd">
<aspectj>
  <weaver options="">
    <include within="org.mybatis.jpetstore..*"/>
    <include within="net.sourceforge.stripes.controller.DispatcherServlet"/>
  </weaver>

  <aspects>
    <aspect name="kieker.monitoring.probe.aspectj.operationExecution.OperationExecutionAspectFull"/>
  </aspects>
</aspectj>
```
<beans xmlns="http://www.springframework.org/schema/beans" [...]> [...]

<!-- Kieker’s instrumentation probes based on the Spring AOP interception framework -->
<bean id="opEMII"
     class="kieker.monitoring.probe.spring.executions.OperationExecutionMethodInvocationInterceptor" />
<aop:config>
    <aop:advisor advice-ref="opEMII"
                 pointcut="execution(public * org.mybatis.jpetstore..*.*(..))"/>
</aop:config>

[...]
</beans>
Kieker Analysis

Method Calls Per 10 Seconds

Average Response Time

Hasselbring, Jung (CAU) Application Performance Monitoring with Kieker  SPP Design for Future  48 / 52
Kieker Analysis

Method Calls Per 3 Seconds

Average Responsetime
Kieker is . . .

- a framework for application monitoring and analysis,
- employed in various industry use cases,
- flexible and extensible.

Kieker supports . . .

- Java, J2EE (in production)
- .NET, VB6, Cobol (industry case studies)
- C, Perl (currently developed)

http://kieker-monitoring.net/
Current work addresses:

- Extensions for workflow monitoring [Brauer and Hasselbring 2012]
- Extensions for profiling embedded systems
- Model-driven instrumentation & analysis
- Dynamic and static software measuring & analysis [Frey et al. 2012]
- Extendable Kieker Data Bridge


