Dynamic Analysis of Software Systems with Kieker
— A Hands-On Lecture —
Guest Lecture in the Course “Enterprise Digital Infrastructure”

André van Hoorn
University of Stuttgart, Germany
Contact: van.hoorn@informatik.uni-stuttgart.de

(including contributions by many colleagues)

May 21, 2014 @ University of Pavia, Italy
Quality of Service (QoS) ...
Introduction and Overview of Approach

Does the searchBook service respond in $\leq 0.5$ seconds in 95% of all cases?

What is the system's availability?

What is the expected workload profile?

How do the components interact?

Which component causes the problem?

How are my resources utilized?
Dynamic analysis, or the analysis of data gathered from a running program, has the potential to provide an accurate picture of a software system because it exposes the system's actual behavior.

This picture can range from class-level details up to high-level architectural views [...].

(Cornelissen et al. 2009)
Among the **benefits over static analysis** are the availability of runtime information and, in the context of object-oriented software, the **exposure of object identities** and the **actual resolution of late binding**.

A **drawback** is that **dynamic analysis** can only provide a **partial picture** of the system, i.e., the results obtained are valid for the **scenarios that were exercised during the analysis**.

(Cornelissen et al. 2009)
Profiling vs. Monitoring

Introduction and Overview of Approach

How to Gather Runtime Data from Executing Systems?

Profiling

• employed in development environments
• considerable performance overhead

Monitoring

• employed in production environments
• captures real usage profile

http://www.jvmmonitor.org/doc/index.html#Getting_started
Monitoring practice in the “real world” (based on what we’ve seen)

- Focus on system level metrics (availability, resource utilization)
- No systematic instrumentation on application level
- Only basic automation of analysis/alarming
- Reactive addition of monitoring probes
1. **Introduction and Overview of Approach**
   - Interactive: Quick Start

2. **Use Cases in Research and Practice**

3. **Kieker's Monitoring Component**

4. **Kieker's Analysis Component & WebGUI**
   - Interactive: WebGUI

5. **Interactive: Java EE Monitoring with Kieker**

6. **A Detailed Look at Selected Use Cases**
Introduction and Overview of Approach

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

About Kieker

The internal behavior of large-scale software systems cannot be determined on the basis of static (e.g., source code) analysis alone. Kieker provides complementary dynamic analysis capabilities, i.e., monitoring and analyzing a software system's runtime behavior — enabling Application Performance Monitoring and Architecture Discovery.

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
Thanks to All Contributors

Introduction and Overview of Approach

Various people contributed to Kieker in the past years.


—Alphabetic list of people who contributed in different form (source code, bug reports, promotion, etc) and intensity
1. Introduction and Overview of Approach
   - Interactive: Quick Start

2. Use Cases in Research and Practice

3. Kieker’s Monitoring Component

4. Kieker’s Analysis Component & WebGUI
   - Interactive: WebGUI

5. Interactive: Java EE Monitoring with Kieker

6. A Detailed Look at Selected Use Cases
Also refer to the Kieker User Guide

1. Chapter 2 (Download and installation)
2. Chapter 2 (Bookstore example)
3. Chapter 5 (AspectJ-based instrumentation)
4. Chapter 5 (TraceAnalysis tool)
5. Appendix A (Wrapper scripts)
Core Kieker Framework Components

Introduction and Overview of Approach ▶ Interactive: Quick Start

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

Monitoring Log/Stream
- Monitoring Record

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

Java probes/samplers:
- Manual instrumentation
- AspectJ
- Spring
- Servlet
- CXF/SOAP
- <your interception technology>

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

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

A. van Hoorn, U Stuttgart, Germany
Core Kieker Framework Components

Introduction and Overview of Approach

Interactive: Quick Start

- 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 Controller
  - Pipe & Filter Configuration
  - 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

Introduction and Overview of Approach ▶ Interactive: Quick Start

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 Controller
- Analysis / Visualization Plugin
- Pipe & Filter Configuration
- Pipe-and-filter framework
- Analysis/Visualization Plugins
- Trace analysis
- Architecture reconstr.
- Dependency graphs
- Sequence diagrams
- Call graphs
- <your visualization>
- <your trace analysis>
- <your reconstruction plugin>
- <your analysis plugin/tool>

A. van Hoorn, U Stuttgart, Germany
Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy 14 / 62
1 Introduction and Overview of Approach
   □ Interactive: Quick Start

2 Use Cases in Research and Practice

3 Kieker’s Monitoring Component

4 Kieker’s Analysis Component & WebGUI
   □ Interactive: WebGUI

5 Interactive: Java EE Monitoring with Kieker

6 A Detailed Look at Selected Use Cases
1 Architecture Discovery (Dynamic/Hybrid Analysis)
   - Extraction of architectural models (structure, behavior)
   - Reverse engineering of legacy systems
   - Software visualization (2D/3D, static/interactive)
   - Trace-based architecture analysis

2 Application Performance Management
   - Continuous QoS monitoring + feedback (self-*)
   - Distributed tracing and trace-based analysis
   - Architecture-based performance analysis
   - Automatic problem detection and diagnosis
   - Extraction of usage profiles (workload intensity, navigational patterns)

3 Characteristics (cross-cutting)
   - Modular, flexible, and extensible architecture
   - Non-intrusive instrumentation
   - Low performance overhead
   - Model-driven instrumentation and analysis
   - Evaluated in lab and industrial case studies
Architecture Discovery: Model Extraction + Visualization (cont’d)
Architecture Discovery: Model Extraction + Visualization (cont’d)
Architecture Discovery: Model Extraction + Visualization (cont’d)
Architecture Discovery: Model Extraction + Visualization (cont’d)
Architecture Discovery: Model Extraction + Visualization (cont’d)
APM: Anomaly Detection + Diagnosis (cont’d)
Selected Topics and Results (cont’d)  
[Marwede et al. 2009]  
Use Cases in Research and Practice

APM: Anomaly Detection + Diagnosis (cont’d)

---

**Deployment Context Level**

**Component Level**

**Operation Level**

---

$40912$

Virtual Machine 'tier'  
$[41472/61098 | 0.03 | 25.90\%]$

$43498$

Virtual Machine 'scooter'  
$[818/2176 | −0.07 | 23.43\%]$

---

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture  
May 21, 2014 @ U Pavia, Italy
APM: Anomaly Detection + Diagnosis (cont’d)
**APM: Anomaly Detection + Diagnosis (cont’d)**

Selected Topics and Results (cont’d)

[Pitakrat 2013, Pitakrat et al. 2014]

Use Cases in Research and Practice

Hora

System-level Predictor

Monitoring

Reader

Kieker, Weka, R, PRISM, ESPER, ...

CDT

PAD

HDD Failure Predictor

Event Log Analyzer

Component-level Predictors

PCM

SLAStic

...
1. Introduction and Overview of Approach
   - Interactive: Quick Start

2. Use Cases in Research and Practice

3. Kieker’s Monitoring Component

4. Kieker’s Analysis Component & WebGUI
   - Interactive: WebGUI

5. Interactive: Java EE Monitoring with Kieker

6. A Detailed Look at Selected Use Cases
Also refer to the Kieker User Guide

1. Ch. 2 (Quick start monitoring)
2. Ch. 3 (Details on the Monitoring component)
3. Ch. 3 (Custom records, probes, writers)
4. Ch. 5 (Monitoring trace information)
5. Appendix E (Configuration file)
**Application code:**

```java
public void getOffers() {
    // EXECUTION to be monitored:
    catalog.getbook(false);
}
```

**Monitoring probe code (schematic):**

```java
// BEFORE execution to be monitored
if (!isMonitoringEnabled()) {
    collectDataBefore();
}

// AFTER execution to be monitored
if (!isMonitoringEnabled()) {
    collectDataAfter();
    writeMonitoringData();
}
```

---

**Instrumentation** — Getting the *monitoring probe* into the *code*

1. Manual instrumentation
2. Aspect-oriented programming (AOP), middleware interception, ...
private static final IMonitoringController MONITORING_CONTROLLER = MonitoringController.getInstance();

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

final OperationExecutionRecord record =
new OperationExecutionRecord(
    "public void Catalog.getBook(boolean)",
    NO_SESSION_ID, NO_TRACEID,
    tin, tout, "myHost",
    NO_EOI_ESS, NO_EOI_ESS);
// Pass record to controller:
MONITORING_CONTROLLER.newMonitoringRecord(record);
MonitoringController API (Excerpt)

Kieker's Monitoring Component

MonitoringController

**Instantiation (static)**
- IMonitoringController: getInstance()
- IMonitoringController: createInstance(Configuration)

**Controller State**
- boolean: isMonitoringEnabled()
- boolean: isMonitoringTerminated()
- boolean: disableMonitoring()
- boolean: enableMonitoring()
- boolean: terminateMonitoring()
- String: getHostname()
- String: toString()

**Time Source**
- TimeSource: getTimeSource()

**Writing**
- boolean: newMonitoringRecord(IMonitoringRecord)

**Adaptive Monitoring**
- boolean: isProbeActivated(String)
- boolean: activateProbe(String)
- boolean: deactivateProbe(String)

**Periodic Sampling**
- ScheduledSamplerJob: schedulePeriodicSampler(ISampler, ..., TimeUnit)
- boolean: removeScheduledSample(ScheduledSamplerJob)

**JMX**

**Registry**

A. van Hoorn, U Stuttgart, Germany
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).

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)).
Listing 1: 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>
```

Start the monitored application:

```
$ java -javaagent:lib/kieker-1.9_aspectj.jar BookstoreStarter
```
Also refer to the Kieker User Guide

1. Ch. 2 (Quick start analysis)
2. Ch. 4 (Details on the Analysis component)
3. Ch. 4 (Custom readers, filters, repositories)
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

- <<Reader>> : FS reader
- <<Filter>> : Performance anomaly filter
- <<Repository>> : System model repository
- anomalyRatings

Workload Anomaly Detection

Anomaly score

Calendar time (hh:mm)

Invocations/minute [x 1000]

A. van Hoorn, U Stuttgart, Germany
/* 1. Create analysis controller for our response time analysis. */
final AnalysisController analysisController
    = new AnalysisController();

/* 2. Configure and register the reader */
final Configuration readerConfig = new Configuration();

readerConfig.setProperty(
    MyPipeReader.CONFIG_PROPERTY_NAME_PIPE_NAME, "somePipe");

final MyPipeReader reader =
    new MyPipeReader(readerConfig, analysisController);
/* 3. Configure, register, and connect the response time filter */
final Configuration filterConfig = new Configuration();

final long rtThresholdNanos =
    TimeUnit.NANOSECONDS.convert(1900, TimeUnit.MICROSECONDS);

filterConfig.setProperty( // configure threshold of 1.9 milliseconds:
    MyResponseTimeFilter.CONFIG_PROPERTY_NAME_TS_NANOS,
    Long.toString(rtThresholdNanos));

final MyResponseTimeFilter filter =
    new MyResponseTimeFilter(filterConfig, analysisController);

analysisController.connect(reader, MyPipeReader.OUTPUT_PORT_NAME,
    filter, MyResponseTimeFilter.INPUT_PORT_NAME_RESPONSE_TIMES);
/* 4. Save configuration to file (optional) */
analysisController.saveToFile(new File("out.kax"));

/* 5. Start the analysis. */
analysisController.run();
public final class CountingFilter extends AbstractFilterPlugin {
}
```java
@Plugin(outputPorts = {
    @OutputPort(name = "eventCount", eventTypes = { Long.class })
})
public final class CountingFilter extends AbstractFilterPlugin {
}
```
@Plugin(outputPorts = {
    @OutputPort(name = "eventCount", eventTypes = { Long.class })})

public final class CountingFilter extends AbstractFilterPlugin {

    private final AtomicLong counter = new AtomicLong();

}
@Plugin(outputPorts = {
    @OutputPort(name = "eventCount", eventTypes = { Long.class })})

public final class CountingFilter extends AbstractFilterPlugin {

    private final AtomicLong counter = new AtomicLong();

    public CountingFilter(Configuration conf, IProjectContext context) {
        super(conf, context);
    }

}
Own Analysis Components (cont’d)

Kieker’s Analysis Component & WebGUI

@Plugin(outputPorts = {
    @OutputPort(name = "eventCount", eventTypes = { Long.class })})

public final class CountingFilter extends AbstractFilterPlugin {

    private final AtomicLong counter = new AtomicLong();

    public CountingFilter(Configuration conf, IProjectContext context) {
        super(conf, context);
    }

    @Override
    public final Configuration getCurrentConfiguration() {
        return new Configuration();
    }

    @Override
    public final Configuration getCurrentConfiguration() {
        return new Configuration();
    }

    }

A. van Hoorn, U Stuttgart, Germany
Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy
@Plugin(outputPorts = {
    @OutputPort(name = "eventCount", eventTypes = { Long.class })})

public final class CountingFilter extends AbstractFilterPlugin {

    ...

    @InputPort(name = "inputEvents", eventTypes = { Object.class })
    public final void inputEvent(final Object event) {
        final Long count = this.counter.incrementAndGet();

        super.deliver("eventCount", count);
    }

}
AnalysisController API (Excerpt)

Kieker's Analysis Component & WebGUI

AnalysisController

**Instantiation:**
+ AnalysisController()
+ AnalysisController(File)

**Persistence:**
+ void: saveToFile(File)

**Pipes-and-Filters Configuration:**
+ void: connect(AbstractPlugin, String, AbstractPlugin, String)
+ void: connect(AbstractPlugin, String, AbstractRepository)

**Controller State:**
+ STATE: getState()
+ void: run()
+ void: terminate(boolean)

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
Also refer to:

1. Example projects included in the WebGUI
2. Tutorial paper: Ehmke [2013]
3. Blog article

http://kieker-monitoring.net/blog/
everything-in-sight-kiekers-webgui-in-action/
###CPU-and-Memory-Example

**Available Plugins**

- **Reader**
  - CoarseReader
  - FineReader
  - JavaReader
  - XMLReader
  - PostReader
  - TCPReader

- **Filter**
  - EventInsertFilter
  - OptionAggregationFilter
  - AnalysisThresholdFilter
  - CountingFilter
  - StringFilter
  - TestFilter
  - MonitoringThresholdFilter
  - RealtimeRecordDelayFilter
  - TimestampFilter
  - TimerFilter
  - TFIFilter
  - CPUNameAggregationFilter
  - MemoryAggregationFilter
  - ItemAndComponentFilter

**Global Configuration**

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classname</td>
<td>InternalAnalysis plugins.reader.filesystem.FSReader</td>
</tr>
<tr>
<td>Name</td>
<td>FSReader</td>
</tr>
<tr>
<td>inputDir</td>
<td>data/CP-and-Memory-Example</td>
</tr>
</tbody>
</table>

**Description**

- A filter computing the throughput of the monitoring

**Input Ports**

- InputRecords

**Output Ports**

- IntenrtRecords

**Configuration**

- timeout
  - initialization
1. Introduction and Overview of Approach
   - Interactive: Quick Start

2. Use Cases in Research and Practice

3. Kieker’s Monitoring Component

4. Kieker’s Analysis Component & WebGUI
   - Interactive: WebGUI

5. Interactive: Java EE Monitoring with Kieker

6. A Detailed Look at Selected Use Cases
Also refer to the Kieker User Guide

1. Chapter 5 (AspectJ-based instrumentation)
2. Chapter 5 (TraceAnalysis tool)
3. Appendix B (Java EE example)
4. Appendix C (Continuous analysis with JMS)
5. Appendix D (Monitoring of system metrics)
A Detailed Look at Selected Use Cases

1. Introduction and Overview of Approach
   - Interactive: Quick Start

2. Use Cases in Research and Practice

3. Kieker’s Monitoring Component

4. Kieker’s Analysis Component & WebGUI
   - Interactive: WebGUI

5. Interactive: Java EE Monitoring with Kieker

6. A Detailed Look at Selected Use Cases
Also refer to the Kieker User Guide

1. Chapter 5 (AspectJ-based instrumentation)
2. Chapter 5 (TraceAnalysis tool)
4. Paper [Rohr et al. 2008]
Trace Terminology

A Detailed Look at Selected Use Cases ➤ Trace Analysis

Legend:

- = call message
- = return message
\(\Rightarrow\) = trace
\(\text{execution with } \text{eoi } i \text{ and ess } j\)

Execution order index (eoi) \(i\): \(i\)-th started execution in a trace
Execution stack size (ess) \(j\): execution started at stack depth \(j\)
A single line of the monitoring log – a single monitoring record

$0; 1283156545623365608; public void kieker.examples..CRM.getOffers();
<no-session-id>; 6488138950668976129; 1283156498817823953;
1283156498820007367; Osterinsel; 2; 1

The meaning of this record:

1. Type of monitoring record (see kieker.map; here: OperationExecutionRecord)
2. Logging timestamp (time in ns)
3. Operation signature (fully qualified)
4. Session id (only with web applications)
5. Trace id (unique id of the trace)
6. $t_{in}$ (start time of execution)
7. $t_{out}$ (end time of execution)
8. Hostname (name of the computer)
9. eoi (execution order index)
10. ess (execution stack size)
Kieker's Trace Meta-Model

A Detailed Look at Selected Use Cases

⊿

Trace Analysis

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
### Sequence Diagrams

1. **Sequence diagrams**
2. Dynamic call trees
3. Hierarchical calling dependency graphs
4. System model

#### Diagrams

(a) **Assembly-level view**

(b) **Deployment-level view**
1. Sequence diagrams
2. **Dynamic call trees**
3. Hierarchical calling dependency graphs
4. System model

(a) Dynamic call tree **(single trace)**

(b) **Aggregated deployment-level call tree**
Hierarchical Calling Dependency Graphs

1. Sequence diagrams
2. Dynamic call trees
3. Hierarchical calling dependency graphs
4. System model

(a) Assembly-level **component** dependency graph

(b) Deployment-level **operation** dependency graph
Sequence diagrams
Dynamic call trees
Hierarchical calling dependency graphs
System model (here: HTML representation)
Details to be found in

1. Master’s Thesis by Bielefeld [2012]
2. Master’s Thesis by Frotscher [2013]
A Detailed Look at Selected Use Cases

PAD Processing Steps

- **Time Series Extraction**
- **Time Series Forecasting**
- **Anomaly Score Calculation**
- **Anomaly Detection**
- **Alerting** (e.g., AMQP)
Step 1: Time Series Extraction

ΘPAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases ▶ ΘPAD

![Diagram showing the processing steps]

- **<<Reader>>**
  - **<<Filter>>**
  - **<<Filter>>**
  - **<<Filter>>**
  - **<<Filter>>**
  - **<<Filter>>**

- **Time Series Extraction**
- **Time Series Forecasting**
- **Anomaly Score Calculation**
- **Anomaly Detection**
- **Alerting (e.g., AMQP)**

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

**Continuous Time**

**Discrete Time Series**

![Chart showing time series and events]

- Event on ES
- Discretization Function
- Time Series X
- Current Time

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
Step 2: Time Series Forecasting

ΘPAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases

A. van Hoorn, U Stuttgart, Germany
Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy
Step 3: Anomaly Score Calculation

$\Theta$PAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases $\Rightarrow$ $\Theta$PAD

- Time Series Extraction
- Time Series Forecasting
- Anomaly Score Calculation
- Anomaly Detection
- Alerting (e.g., AMQP)

![Diagram showing the processing steps]

- Anomaly Score Calculation
- Alerting (e.g., AMQP)
Step 4: Anomaly Detection

PAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases  θ θ θ

A. van Hoorn, U Stuttgart, Germany
Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy 59 / 62
Framework Features & Extension Points

Kieker

- Modular, flexible, and extensible architecture (Probes, records, readers, writers, filters etc.)
- Pipes-and-filters framework for analysis configuration
- Distributed tracing (logging, reconstruction, visualization)
- 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

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


For a comprehensive list of publications, talks, and theses about Kieker, visit: http://kieker-monitoring.net/research/