André van Hoorn\textsuperscript{1} and Nils Christian Ehmke\textsuperscript{2}

\textit{(including contributions by many colleagues)}

\textsuperscript{1} University of Stuttgart, Germany
\textsuperscript{2} Kiel University, Germany

March 23, 2014 @ Dublin, Ireland
Kieker: Dynamic Analysis Workflow

Introduction and Overview of Approach

Monitoring Probe
Software System with Monitoring Instrumentation
Measurement
Monitoring log/stream
Monitoring Records
Analysis
Analysis Configuration (via API and WebGUI)
Pipes and Filters

Online and Offline Visualization

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE '14, Dublin
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
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
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**

- **Monitoring Log/Stream**

- **Kieker.Analysis**

---

#### Java probes/samplers:

- Manual instrumentation
- AspectJ
- Spring
- Servlet
- CXF/SOAP

**<your interception technology>**

**<your monitoring probe>**

---

**+ 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>**
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 / Visualization Plugin
- Pipe & Filter Configuration
- Analysis Controller

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 / Visualization Plugin
- Pipe & Filter Configuration

Analysis / Visualization Plugin
- Pipe-and-filter framework
- Analysis and visualization
- Analysis and visualization plugins
- Dependency graphs
- Sequence diagrams
- Call graphs
- <your visualization>
- <your trace analysis>
- <your reconstruction plugin>
- <your analysis plugin/tool>

A. van Hoorn and N. Ehmke

Kieker Tutorial
Introduction and Overview of Approach

Interactive: Quick Start

Use Cases in Research and Practice

Kieker’s Monitoring Component

Kieker’s Analysis Component & WebGUI

Interactive: WebGUI

Interactive: Java EE Monitoring with Kieker

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)
Selected Topics and Results (cont’d)

[Magedanz 2011]

Use Cases in Research and Practice

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)
APM: Anomaly Detection + Diagnosis (cont’d)
Selected Topics and Results (cont’d)
[Fittkau et al. 2013; 2014]
Use Cases in Research and Practice

**APM**: Anomaly Detection + Diagnosis (cont’d)

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE ‘14, Dublin
APM: Anomaly Detection + Diagnosis (cont’d)
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
Details on the Monitoring Component

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)
Program Instrumentation (Here: Manual)

Example: Monitoring Operation Executions

Kieker's Monitoring Component

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)`

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

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

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

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

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

---

A. van Hoorn and N. Ehmke  
Kieker Tutorial  
Mar 23, 2014 @ ICPE ’14, Dublin
(a) 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.

(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

```
11   @OperationExecutionMonitoringProbe
12   public void getOffers() {
13       catalog.getBook(false);
14   }
15 }
```

Annotation-based (AOP) instrumentation for monitoring trace information
Listing 1: META-INF/aop.xml

```xml
<!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN" "http://www.aspectj.org.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-1.8_aspectj.jar BookstoreStarter
```
Monitoring Overhead

[Waller and Hasselbring 2013]
Kieker's Monitoring Component

Response time of...
- Writing (W)
- Collecting (C)
- Instrumentation (I)
- Method time (T)
(mean with 95% CI)

Response time (μs)

<table>
<thead>
<tr>
<th>Kieker version (ASCII writer; operation execution records)</th>
<th>Writing</th>
<th>Collecting</th>
<th>Instrumentation</th>
<th>Method time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.91</td>
<td>6.70</td>
<td>0.49</td>
<td>8.74</td>
<td>15.61</td>
</tr>
<tr>
<td>0.95a</td>
<td>8.78</td>
<td>0.54</td>
<td>8.74</td>
<td>10.66</td>
</tr>
<tr>
<td>1.0</td>
<td>8.74</td>
<td>0.47</td>
<td>8.72</td>
<td>10.15</td>
</tr>
<tr>
<td>1.1</td>
<td>100.19</td>
<td>0.58</td>
<td>10.15</td>
<td>8.66</td>
</tr>
<tr>
<td>1.2</td>
<td>106.09</td>
<td>0.43</td>
<td>10.15</td>
<td>8.66</td>
</tr>
<tr>
<td>1.3</td>
<td>104.02</td>
<td>0.49</td>
<td>10.15</td>
<td>8.66</td>
</tr>
<tr>
<td>1.4</td>
<td>103.72</td>
<td>0.54</td>
<td>10.15</td>
<td>8.66</td>
</tr>
<tr>
<td>1.5</td>
<td>107.43</td>
<td>0.47</td>
<td>10.15</td>
<td>8.66</td>
</tr>
<tr>
<td>1.6</td>
<td>100.19</td>
<td>0.58</td>
<td>10.15</td>
<td>8.66</td>
</tr>
<tr>
<td>1.7</td>
<td>106.09</td>
<td>0.43</td>
<td>10.15</td>
<td>8.66</td>
</tr>
<tr>
<td>1.8</td>
<td>104.02</td>
<td>0.49</td>
<td>10.15</td>
<td>8.66</td>
</tr>
</tbody>
</table>
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 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

<<Repository>>
: System model repository

operationExecutions
: Performance anomaly filter

<<Filter>>
: Anomaly graph plotter

anomalyRatings

workloadAnomalyDetection

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE '14, Dublin
/* 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.CONFIGPROPERTY_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);
Programmatic Analysis Creation (cont’d)

Kieker's Analysis Component & WebGUI

/* 4. Save configuration to file (optional) */
analysisController.saveToFile(new File("out.kax");

/* 5. Start the analysis. */
analysisController.run();
public final class CountingFilter extends AbstractFilterPlugin {
}
@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);
    }

}
@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();
    }
}
@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-Filter Configuration:**
- `void: connect(AbstractPlugin, String, AbstractPlugin, String)`
- `void: connect(AbstractPlugin, String, AbstractRepository)`

**Controller State:**
- `STATE: getState()`
- `void: run()`
- `void: terminate(boolean)`
- `void: saveToFile(File)`
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/
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)
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

Execution order index (eoi) \(i\): \(i\)-th started execution in a trace
Execution stack size (ess) \(j\): execution started at stack depth \(j\)
### 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. **eo** *(execution order index)*
10. **ess** *(execution stack size)*
Kieker’s Trace Meta-Model

A Detailed Look at Selected Use Cases  ►  Trace Analysis

- **Trace**:
  - traceld : long
  - sessionId : String

- **Execution**:
  - traceld : long
  - sessionId : String
  - eoi : int
  - ess : int
  - tin : long
  - tout : long
  - assumed : boolean

- **Message**:
  - timestamp : long

- **AllocationComponent**:
  - allocationComponent

- **Operation**:
  - operation

- **ExecutionTrace**:
  - executions

- **MessageTrace**:
  - receivingExecution
  - messages

- **SynchronousCallMessage**

- **SynchronousReplyMessage**
Sequence diagrams

Dynamic call trees

Hierarchical calling dependency graphs

System model

(a) Assembly-level view

(b) Deployment-level view
Dynamic Call Trees

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**
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**
A Detailed Look at Selected Use Cases

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

OPAD 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

Time Series Extraction

Time Series Forecasting

Anomaly Score Calculation

Anomaly Detection

Alerting (e.g., AMQP)

Continuous Time

Discretization Function

Event on ES

Discrete Time Series

Time Series X

Current Time

\[
\begin{align*}
X(1) &= 7 \\
X(2) &= 8 \\
X(3) &= 5 \\
X(4) &= 6 \\
\end{align*}
\]

\[
\Delta X = X(4) - X(3)
\]

\[
\text{select } \text{sum(value) as aggregation} \\
\text{from MeasureEvent.win:time\_batch( 1000 msec )}
\]
Step 2: Time Series Forecasting

ΘPAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases  ΘPAD

Time Series Extraction
Time Series Forecasting
Anomaly Score Calculation
Anomaly Detection
Alerting (e.g., AMQP)
Step 3: Anomaly Score Calculation

ΘPAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases ➤ ΘPAD

- Time Series Extraction
- Time Series Forecasting
- Anomaly Score Calculation
- Anomaly Detection
  (e.g., AMQP)
Step 4: Anomaly Detection

ΠAPD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases ▸ ΠAPD

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

Abnormal Score
Normal Score
Anomaly Threshold
Anomaly Detected
• 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
Literature


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