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

A. van Hoorn, U Stuttgart, Germany
Monitoring/Dynamic Analysis

Introduction and Overview of Approach

A. van Hoorn, U Stuttgart, Germany
Does the searchBook service respond in <= 0.5 seconds in 95% of all cases?
Does the searchBook service respond in ≤ 0.5 seconds in 95% of all cases?

What is the system's availability?
What is the expected workload profile?
Monitoring/Dynamic Analysis

Introduction and Overview of Approach

BookstoreSystem

<<SystemAssembly>>

<<Component>>
Bookstore

<<Component>>
Catalog

<<Component>>
CRM
How do the components interact?
How do the components interact?

Which component causes the problem?
Monitoring/Dynamic Analysis

Introduction and Overview of Approach

How are my resources utilized?

Bookstore

Catalog

CRM

System Assembly

Component

Bookstore System
"Dynamic analysis, or the analysis of data gathered from a running program,"

(Cornelissen et al. 2009)
Dynamic analysis, or the analysis of data gathered from a running program, has the potential to provide an accurate picture of a software system's actual behavior.

(Cornelissen et al. 2009)
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**.

(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 of Software Systems

Introduction and Overview of Approach

Self-adaptive Software Performance Monitoring for Anomaly Localization

Application-level Monitoring
- Observations in field
  - Extensive infrastructure monitoring, application monitoring not widespread
  - Reactive monitoring probe injection only (after a critical performance drop has occurred)

Business Processes
- Key performance indicators, e.g. process throughput, ...

Services
- SLO appliance, workload, ...

Response times, operational profile, ...

Middleware Container
- Thread/connection pool sizes, ...

Heap size, ...

Virtual Machine
- CPU/memory utilization, ...

Operating System
- Availability, reliability, ...

Hardware

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
Introduction and Overview of Approach

Monitoring Probe
Software System with Monitoring Instrumentation
Kieker: Dynamic Analysis Workflow

Introduction and Overview of Approach

- Measurement
- Monitoring log/stream
- Monitoring Records

A. van Hoorn, U Stuttgart, Germany

Software System with Monitoring Instrumentation
Kieker: Dynamic Analysis Workflow

Introduction and Overview of Approach

Analysis Configuration (via API and WebGUI)

Pipes and Filters

Kieker + Bookstore-Example

Analysis

Monitoring Records

Monitoring log/stream

Measurement

Monitoring Probe

Software System with Monitoring Instrumentation

A. van Hoorn, U Stuttgart, Germany

May 21, 2014 @ U Pavia, Italy
Kieker: Dynamic Analysis Workflow

Introduction and Overview of Approach

Monitoring Records

Measurement

Monitoring log/stream

Software System with Monitoring Instrumentation

Analysis Configuration (via API and WebGUI)

Pipes and Filters

Invocations/minute [x 1000]

Calendar time (hh:mm)

Workload Anomaly Detection

Online and Offline Visualization

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
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

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy
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

Java probes/samplers:

- Manual instrumentation
- AspectJ
- Spring
- Servlet
- CXF/SOAP
- Sigar
- CPU utilization
- Memory usage
- <your technology>
- <your monitoring probe>

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

Kieker.Monitoring

Monitoring Probe

Monitoring Controller

Logging

Time Source

Periodic Sampling

JMX Interface

Monitoring Writer

Monitoring Log/Stream

Monitoring Record

Kieker.Analysis

Pipe & Filter Configuration

Monitoring Reader

Analysis / Visualization Plugin

Analysis Controller

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
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**

- Monitoring Reader
- Analysis / Visualization Plugin
- Analysis Controller

**Java probes/samplers:**

- Manual instrumentation
- AspectJ
- Spring
- Servlet
- CXF/SOAP
- <your interception technology>
- <your 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>

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
Core Kieker Framework Components

Introduction and Overview of Approach ➤ Interactive: Quick Start

Kieker.Monitoring

Monitoring Probe
LogMonitoring Probe
Source
Monitoring Controller
Logging
JMX Interface
Periodic Sampling

Monitoring Log/Stream

Monitoring Record

Kieker.Analysis

Pipe & Filter Configuration
Monitoring Reader
Analysis / Visualization Plugin

Analysis Controller

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
  - Time Source
  - Logging
  - Periodic Sampling
  - JMX Interface

Monitoring Log/Stream
- Monitoring Record

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

Monitoring Log/Stream

- Monitoring Record

Kieker.Analysis

- Monitoring Reader
- 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
- Monitoring Log/Stream
- JMX Interface
- Periodic Sampling
- Logging
- Time Source

Kieker.Analysis
- Monitoring Reader
- Analysis Controller
- Monitoring Record
- Analysis / Visualization Plugin
- Pipe & Filter Configuration
- Dependency graphs
- Sequence diagrams
- Call graphs
- <your visualization>
- <your trace analysis>
- <your reconstruction plugin>
- <your analysis plugin/tool>

Pipe-and-filter framework
Analysis/Visualization Plugins
- Architecture reconstr.
- Trace analysis

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
Kieker Use Cases/Characteristics

Use Cases in Research and Practice

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)

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

A. van Hoorn, U Stuttgart, Germany
1 Architecture Discovery (Dynamic/Hybrid Analysis)
   • Extraction of architectural models (structure, behavior)

2 Application Performance Management
1 Architecture Discovery (Dynamic/Hybrid Analysis)
   - Extraction of architectural models (structure, behavior)
   - Reverse engineering of legacy systems

2 Application Performance Management
1. **Architecture Discovery** (Dynamic/Hybrid Analysis)
   - Extraction of **architectural models** (structure, behavior)
   - Reverse engineering of legacy systems
   - Software visualization (2D/3D, static/interactive)

2. **Application Performance Management**
Kieker Use Cases/Characteristics

Use Cases in Research and Practice

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
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-***)
Kieker Use Cases/Characteristics

Use Cases in Research and Practice

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

---

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy
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

Application Performance Management

- Continuous QoS monitoring + feedback (self-* )
- Distributed tracing and trace-based analysis
- Architecture-based performance analysis
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
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)*
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
Architecture Discovery: Model Extraction + Visualization (cont’d)
Architecture Discovery: Model Extraction + Visualization (cont’d)
Selected Topics and Results (cont’d)

Use Cases in Research and Practice

Architecture Discovery: Model Extraction + Visualization (cont’d)
Architecture Discovery: Model Extraction + Visualization (cont’d)
APM: Anomaly Detection + Diagnosis
Selected Topics and Results (cont’d)
[Marwede et al. 2009]
Use Cases in Research and Practice

APM: Anomaly Detection + Diagnosis (cont’d)
APM: Anomaly Detection + Diagnosis (cont’d)
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
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)
Example: Monitoring Operation Executions

Kieker’s Monitoring Component

Application code: ___________________________________________________________________

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

Monitoring probe code (schematic): ________________________________
**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();
}
```
**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();
}
```
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();
}
```

```java
// 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, ...
this.catalog.getBook(false); // <-- the monitored execution
```java
this.catalog.getBook(false); // <-- the monitored execution

final OperationExecutionRecord record =
    new OperationExecutionRecord(
        "public void Catalog.getBook(boolean)",
        NO_SESSION_ID, NO_TRACEID,
        tin, tout, "myHost",
        NO_EOI_ESS, NO_EOI_ESS);
```
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);
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

<table>
<thead>
<tr>
<th>Instantiation (static)</th>
<th>Controller State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing</td>
<td></td>
</tr>
<tr>
<td>Adaptive Monitoring</td>
<td></td>
</tr>
<tr>
<td>Periodic Sampling</td>
<td></td>
</tr>
<tr>
<td>JMX</td>
<td>Registry</td>
</tr>
</tbody>
</table>

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
### MonitoringController

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

**Controller State**

**Writing**

**Adaptive Monitoring**

**Periodic Sampling**

**JMX**

**Registry**

**Time Source**
**MonitoringController API** (Excerpt)

**Kieker's Monitoring Component**

---

**MonitoringController**

### Instantiation (static)
- `IMonitoringController: getInstance()`  
- `IMonitoringController: createInstance(Configuration)`

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

### Adaptive Monitoring

### Periodic Sampling

### Controller State

### Time Source

### JMX

### Registry
## MonitoringController API (Excerpt)

### MonitoringController

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

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

- **Adaptive Monitoring**

- **Periodic Sampling**

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

- **Time Source**

- **JMX**

- **Registry**
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()

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

**Adaptive Monitoring**

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

**Periodic Sampling**

**JMX**

**Registry**

A. van Hoorn, U Stuttgart, Germany
Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy 23 / 62
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)

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

### Periodic Sampling

### Time Source
- + TimeSource: getTimeSource()

### JMX

### Registry
MonitoringController

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

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

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

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

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

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

**JMX**

**Registry**

Adaptive Monitoring

- Periodic Sampling
- Writing
- Controller State
- Instantiation (static)

A. van Hoorn, U Stuttgart, Germany
Aspect-Oriented Programming (AOP)

Kieker's Monitoring Component

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.

```
public void b(){
    ...
}
```

AOP — Aspect-oriented programming (following [Kiczales et al. 1996])

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
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)).

---

```
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_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
```
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

outputPort
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

<<Reader>>: FS reader
<<Filter>>: Performance anomaly filter
operationExecutions
systemModel

A. van Hoorn, U Stuttgart, Germany
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

<<Reader>> : FS reader

operationExecutions

<<Filter>>
: Performance anomaly filter

systemModel

<<Repository>>
: System model repository

outputPort
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

Operation Executions

<<Filter>>
: Performance anomaly filter

Output Port

<<Reader>>
: FS reader

<<Filter>>
: Anomaly graph plotter

System Model

<<Repository>>
: System model repository

Anomaly Ratings

A. van Hoorn, U Stuttgart, Germany
Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy 29 / 62
Example Pipe-and-Filter Configuration

Kieker’s Analysis Component & WebGUI

<<Reader>>
: FS reader

operationExecutions

<<Filter>>
: Performance anomaly filter

systemModel

<<Repository>>
: System model repository

anomalyRatings

<<Filter>>
: Anomaly graph plotter

Invocations/minute [x 1000]
Calendar time (hh:mm)
Workload Anomaly Detection

A. van Hoorn, U Stuttgart, Germany
Kieker — A Hands-On Lecture
May 21, 2014 @ U Pavia, Italy
/* 1. Create analysis controller for our response time analysis. */
final AnalysisController analysisController
    = new AnalysisController();
/* 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);
/* 4. Save configuration to file (optional) */
analysisController.saveToFile(new File("out.kax"));
4. Save configuration to file (optional)

```java
analysisController.saveToFile(new File("out.kax"));
```

5. Start the analysis

```java
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

**Instantiation:**

**Controller State:**

**Persistence:**

**Pipes-and-Filter Configuration:**
**AnalysisController**

**Instantiation:**
- `AnalysisController()`
- `AnalysisController(File)`

**Controller State:**

**Persistence:**

**Pipes-and-Filters Configuration:**
**AnalysisController API (Excerpt)**

Kieker's Analysis Component & WebGUI

---

### AnalysisController

- **Instantiation:**
  - `AnalysisController()`
  - `AnalysisController(File)`

- **Persistence:**

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

---

Controller State:
AnalysisController

**Instantiation:**
- AnalysisController()
- AnalysisController(File)

**Persistence:**

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

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

= trace

= execution with eoi i 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:
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)*
A single line of the monitoring log – a single monitoring record

|$0; 1283156545623365608; \text{public void kieker.examples..CRM.getOffers();}$

<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)
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)
A single line of the monitoring log – a single monitoring record

$0; 1283156545623365608; public void kieker.examples..CRM.getOffers();
<no-session-id>; 648813895068976129; 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)
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)
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)
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)
A single line of the monitoring log – a single *monitoring record*

```java
$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)*
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)
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

Diagram:
- **Trace**:
  - `traceId : long`
  - `sessionId : String`
- **Execution**:
  - `traceId : long`
  - `sessionId : String`
  - `eoi : int`
  - `ess : int`
  - `tin : long`
  - `tout : long`
  - `assumed : boolean`
- **MessageTrace**:
  - `receivingExecution`
  - `sendingExecution`
- **Message**:
  - `timestamp : long`
- **AllocationComponent**:
  - `allocationComponent`
- **Operation**:
  - `operation`
- **ExecutionTrace**:
  - `executions`
- **Message**:
  - `messages`
1. **Sequence diagrams**
2. Dynamic call trees
3. Hierarchical calling dependency graphs
4. System model

(a) **Assembly-level view**

(b) **Deployment-level view**
Dynamic Call Trees

Kieker.TraceAnalysis Tool (cont’d)

A Detailed Look at Selected Use Cases  ▸ Trace Analysis

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**
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]
PAD Processing Steps

A Detailed Look at Selected Use Cases

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

PAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases ▶ PAD

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

Discretization Function

Continuous Time

Discrete Time Series

Event on ES

select sum(value) as aggregation
from MeasureEvent.win:time_batch( 1000 msec )
Step 2: Time Series Forecasting

ΘPAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases

<<Reader>>

Time Series Extraction

<<Filter>>

Time Series Forecasting

<<Filter>>

Anomaly Score Calculation

<<Filter>>

Anomaly Detection

<<Filter>>

Alerting (e.g., AMQP)

ΔW

4 3 3 4 6 5.79

1 2 3 4 5 6

A. van Hoorn, U Stuttgart, Germany

Kieker — A Hands-On Lecture

May 21, 2014 @ U Pavia, Italy
Step 3: Anomaly Score Calculation

ΘPAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases ➔ ΘPAD

<<Reader>> ➔ <<Filter>> ➔ <<Filter>> ➔ <<Filter>> ➔ <<Filter>> ➔ Alerting (e.g., AMQP)

Time Series Extraction ➔ Time Series Forecasting ➔ Anomaly Score Calculation ➔ Anomaly Detection

---

Graphical representation of the process:

- Time Series Extraction
- Time Series Forecasting
- Anomaly Score Calculation
- Anomaly Detection
- Alerting

Bar chart showing anomaly scores for different time points:

- Time points: 1, 2, 3, 4, 5, 6
- Anomaly scores: 0, 1, 1, 0, 2, 6

A. van Hoorn, U Stuttgart, Germany
Step 4: Anomaly Detection

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)

![Diagram showing the processing steps]

**Abnormal Score**
- Normal Score
- Anomaly Threshold
- Anomaly Detected
Framework Features & Extension Points

• 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/