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
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

Software System with Monitoring Instrumentation
Kieker: Dynamic Analysis Workflow

Introduction and Overview of Approach

- Analysis
- Analysis Configuration (via API and WebGUI)
- Pipes and Filters

Monitoring Records → Monitoring log/stream → Analysis Configuration (via API and WebGUI)

Analysis → Pipes and Filters

Software System with Monitoring Instrumentation
Kieker: Dynamic Analysis Workflow

Introduction and Overview of Approach

- Monitoring Records
- Measurement
- Monitoring log/stream
- Analysis
- Analysis Configuration (via API and WebGUI)
- Pipes and Filters
- Invocations/minute [x 1000]
- Calendar time (hh:mm)
- Workload Anomaly Detection
- Online and Offline Visualization

Software System with Monitoring Instrumentation

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
Visit http://kieker-monitoring.net

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 and N. Ehmke

Kieker Tutorial

Mar 23, 2014 @ ICPE '14, Dublin
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
- Time Source
- Logging
- Periodic Sampling
- JMX Interface

Monitoring Log/Stream

- Monitoring Record

Kieker.Analysis

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

Java probes/samplers:

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

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

- Pipe & Filter Configuration
- Analysis / Visualization Plugin
- Analysis Controller

Java probes/samplers:

- Control-flow tracing
  - Manual instrumentation
  - AspectJ
  - Spring
  - Servlet
  - CXF/SOAP

- Resource monitoring
  - Sigar

- Resource instrumentation

- CPU utilization
- Memory usage

- <your technology>
- <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

- Monitoring Probe
- Monitoring Controller
- Monitoring Writer
- Monitoring Log/Stream
- Monitoring Record
- Kieker.Analysis
  - Pipe & Filter Configuration
  - Monitoring Reader
  - Analysis / Visualization Plugin
  - Analysis Controller
Core Kieker Framework Components

Introduction and Overview of Approach ➤ Interactive: Quick Start

Kieker.Monitoring

- 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>

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE ’14, Dublin
Core Kieker Framework Components

Introduction and Overview of Approach ➤ Interactive: Quick Start

- Monitoring Probe
- Monitoring Controller
- Monitoring Writer
- Monitoring Log Stream
- Kieker Analysis

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
- Time Source
- Periodic Sampling
- Logging
- JMX Interface

Monitoring Log/Stream

- Monitoring Record

Kieker.Analysis

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

Monitoring Readers/Writer

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

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
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-***)
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
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
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)
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)

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)
Architecture Discovery: Model Extraction + Visualization (cont’d)
Architecture Discovery: Model Extraction + Visualization (cont’d)
Selected Topics and Results (cont’d)
[Bielefeld 2012, Frotscher 2013]

Use Cases in Research and Practice

APM: Anomaly Detection + Diagnosis

XING

A. van Hoorn and N. Ehmke
APM: Anomaly Detection + Diagnosis (cont’d)
APM: Anomaly Detection + Diagnosis (cont’d)
APM: Anomaly Detection + Diagnosis (cont’d)

Hora
System-level Predictor
Monitoring
Reader
!
!
Kieker, Weka, R, PRISM, ESPER, ...
CDT
PAD
HDD Failure Predictor
Event Log Analyzer
Component-level Predictors
PCM
SLAStic
...

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

Hora

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
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):
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();
}
```
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
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

**Instantiation (static)**

**Controller State**

**Writing**

**Adaptive Monitoring**

**Time Source**

**Periodic Sampling**

**JMX**

**Registry**

A. van Hoorn and N. Ehmke

Kieker Tutorial

Mar 23, 2014 @ ICPE '14, Dublin
# MonitoringController API (Excerpt)

**Kieker's Monitoring Component**

---

**MonitoringController**

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

- **Controller State**

- **Writing**

- **Adaptive Monitoring**

- **Periodic Sampling**

- **JMX**

- **Registry**

- **Time Source**

---

A. van Hoorn and N. Ehmke

Kieker Tutorial

Mar 23, 2014 @ ICPE ’14, Dublin
MonitoringController API (Excerpt)

Kieker's Monitoring Component

MonitoringController

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

Writing
- boolean: newMonitoringRecord(IMonitoringRecord)

Adaptive Monitoring

Controller State

Periodic Sampling

Time Source

JMX

Registry

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE '14, Dublin
MonitoringController API (Excerpt)

**MonitoringController**

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

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

- **Adaptive Monitoring**

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

- **Time Source**

- **Periodic Sampling**

- **JMX**

- **Registry**
MonitoringController

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

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

**Adaptive Monitoring**

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

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

**Periodic Sampling**

**JMX**

**Registry**
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**

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

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

**JMX**

**Registry**
### 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()`

### JMX

### Registry
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)).

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

```
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
<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)

Kieker version (ASCII writer; operation execution records)
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

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE '14, Dublin
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

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

operationExecutions

<<Reader>>
: FS reader

<<Filter>>
: Performance anomaly filter

systemModel

outputPort
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

<<Reader>>
: FS reader
outputPort

<<Filter>>
: Performance anomaly filter
operationExecutions
systemModel

<<Repository>>
: System model repository

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE '14, Dublin
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE '14, Dublin
Example Pipe-and-Filter Configuration

Kieker's Analysis Component & WebGUI

- **outputPort**
  - **<<Reader>>**: FS reader
- **systemModel**
  - **<<Filter>>**: Performance anomaly filter
- **anomalyRatings**
  - **<<Filter>>**: Anomaly graph plotter

Workload Anomaly Detection

- **Invocations/minute** [x 1000]
- **Calendar time (hh:mm)**
- **Anomaly score**
/* 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_NAMEPIPE_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"));
/* 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();
   }

}
@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);
    }
}

<table>
<thead>
<tr>
<th><strong>Instantiation:</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Persistence:</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Pipes-and-Filter Configuration:</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Controller State:</strong></th>
</tr>
</thead>
</table>

AnalysisController API (Excerpt)

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

**Controller State:**

**Persistence:**

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

**AnalysisController**

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

**Persistence:**

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

**Controller State:**
AnalysisController API (Excerpt)

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
   - Quick Start

2. Use Cases in Research and Practice

3. Kieker’s Monitoring Component

4. Kieker’s Analysis Component & WebGUI
   - 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

:CRM
getBook(...) getOffers(...) getBook(...) searchBook(...)

:Bookstore
:Catalog

Legend:
= trace
= 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 \( j \)
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*

```plaintext
$0; 1283156545623365608; public void kieker.examples..CRM.getOffers();
<no-session-id>; 648813850668976129; 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)*
### Anatomy of a Monitoring Record

#### A single line of the monitoring log – a single *monitoring record*

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of monitoring record</td>
<td>see kieker.map; here: OperationExecutionRecord</td>
</tr>
<tr>
<td>Logging timestamp</td>
<td>time in ns</td>
</tr>
<tr>
<td>Operation signature</td>
<td>fully qualified</td>
</tr>
</tbody>
</table>

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>; 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)
Anatomy of a Monitoring 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)*
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)*
Anatomy of a Monitoring 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`)
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*

$0; 1283156545623365608; public \ void \ kieker.examples..CRM.getOffers();
\langle\text{no-session-id}\rangle; 6488138950668976129; 1283156498817823953;
1283156498820007367; \text{Osterinsel}; 2; 1$

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

(a) Assembly-level view

(b) Deployment-level view

A. van Hoorn and N. Ehmke
Kieker Tutorial
Mar 23, 2014 @ ICPE ’14, Dublin
Dynamic Call Trees

Kieker.TraceAnalysis Tool (cont’d)

A Detailed Look at Selected Use Cases

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]
A Detailed Look at Selected Use Cases

TH PAD Processing Steps

- 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

Time Series Extraction

Time Series Forecasting

Anomaly Score Calculation

Anomaly Detection

Alerting (e.g., AMQP)

Continuous Time

Discrete Time Series

Event on ES

Discretization Function

Time Series X

Current Time

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

ThetaPAD Processing Steps (cont’d)

A Detailed Look at Selected Use Cases

A. van Hoorn and N. Ehmke

Kieker Tutorial

Mar 23, 2014 @ ICPE ’14, Dublin
Step 3: Anomaly Score Calculation

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)
Step 4: Anomaly Detection

PAD 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)

![Diagram of anomaly detection process]

<table>
<thead>
<tr>
<th>Anomaly Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abnormal Score</td>
</tr>
<tr>
<td>Normal Score</td>
</tr>
<tr>
<td>Anomaly Threshold</td>
</tr>
</tbody>
</table>

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