

# Kieker

ICPE '14 Tutorial

André van Hoorn<sup>1</sup> and Nils Christian Ehmke<sup>2</sup>  
*(including contributions by many colleagues)*

<sup>1</sup> University of Stuttgart, Germany

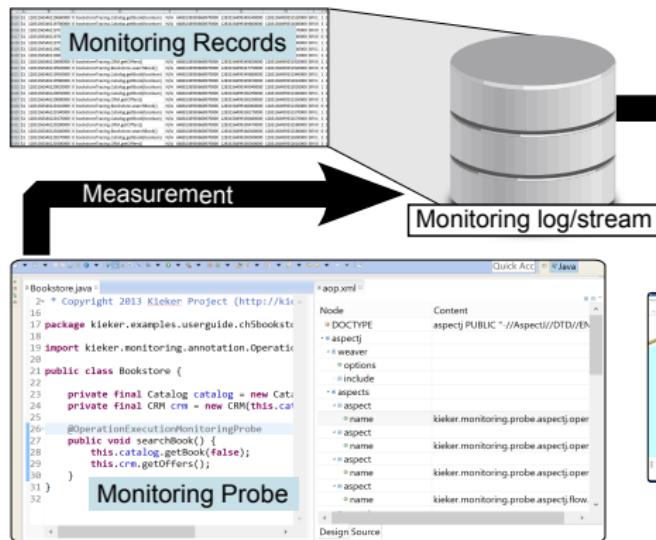
<sup>2</sup> Kiel University, Germany

March 23, 2014 @ Dublin, Ireland

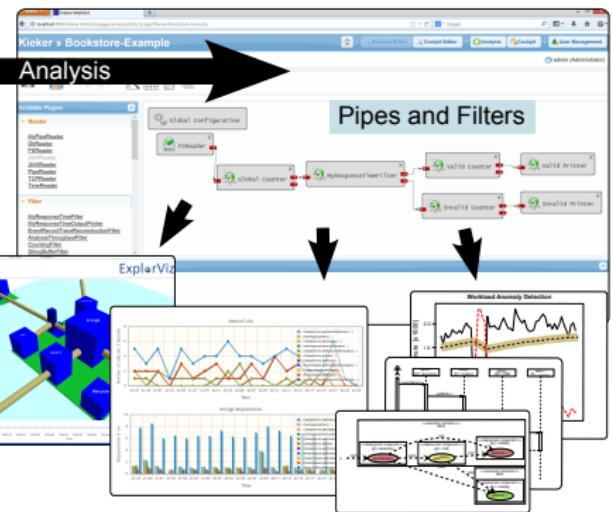


Kieker: Dynamic Analysis Workflow

## Introduction and Overview of Approach



## Analysis Configuration (via API and WebGUI)

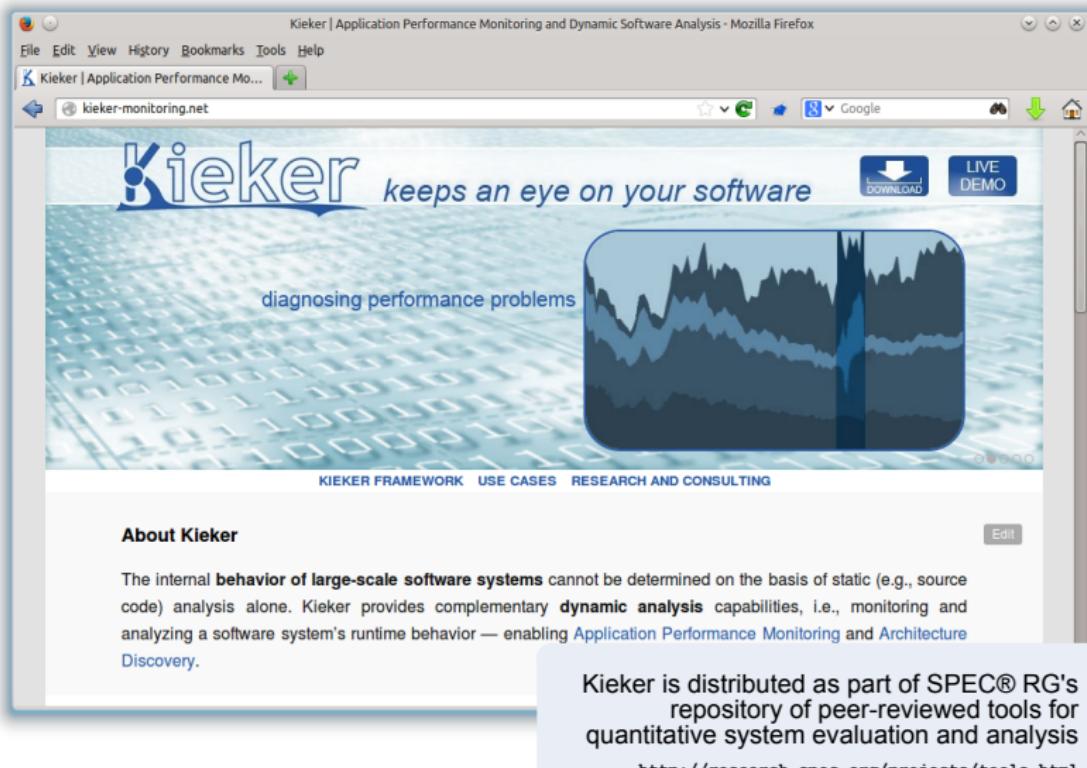


## Software System with Monitoring Instrumentation

## Online and Offline Visualization

- 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



The screenshot shows a Mozilla Firefox browser window displaying the Kieker website. The title bar reads "Kieker | Application Performance Monitoring and Dynamic Software Analysis - Mozilla Firefox". The main content area features the Kieker logo and the tagline "keeps an eye on your software". Below this, a large graphic shows a 3D surface plot of binary code, with the text "diagnosing performance problems" overlaid. To the right is a chart showing fluctuating data over time. Navigation links at the bottom include "KIEKER FRAMEWORK", "USE CASES", and "RESEARCH AND CONSULTING". A "LIVE DEMO" button is also visible. In the "About Kieker" section, there is a detailed description of the tool's purpose and capabilities, mentioning dynamic analysis, monitoring, and system evaluation. A "Edit" button is located next to the text. At the bottom right, there is a "spec Research" logo featuring a bar chart.

Kieker | Application Performance Monitoring and Dynamic Software Analysis - Mozilla Firefox

Kieker | Application Performance Mo... 

kieker-monitoring.net       Google  

# Kieker

keeps an eye on your software

diagnosing performance problems

LIVE DEMO

KIEKER FRAMEWORK USE CASES RESEARCH AND CONSULTING

### 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](#).

Spec Research

# Thanks to All Contributors



Introduction and Overview of Approach

Various people contributed to Kieker in the past years.

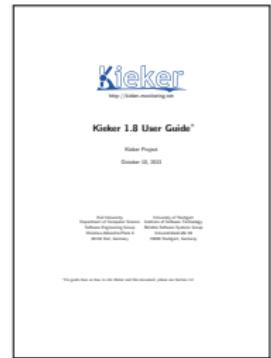
***Tillmann (Till) Bielefeld, Peer Brauer, Philipp Döhring,  
Jens Ehlers, Nils Ehmke, Florian Fittkau, Thilo Focke,  
Sören Frey, Tom Frotscher, Henry Grow,  
Wilhelm (Willi) Hasselbring, André van Hoorn, Reiner Jung,  
Benjamin Kiel, Dennis Kieselhorst, Holger Knoche, Arnd Lange,  
Marius Löwe, Marco Lübecke, Felix Magedanz, Nina Marwede,  
Robert von Massow, Jasminka Matevska, Oliver Preikszas,  
Sönke Reimer, Bettual Richter, Matthias Rohr, Nils Sommer,  
Lena Stöver, Jan Waller, Robin Weiß, Björn Weißenfels,  
Matthias Westphal, Christian Wulf***

—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

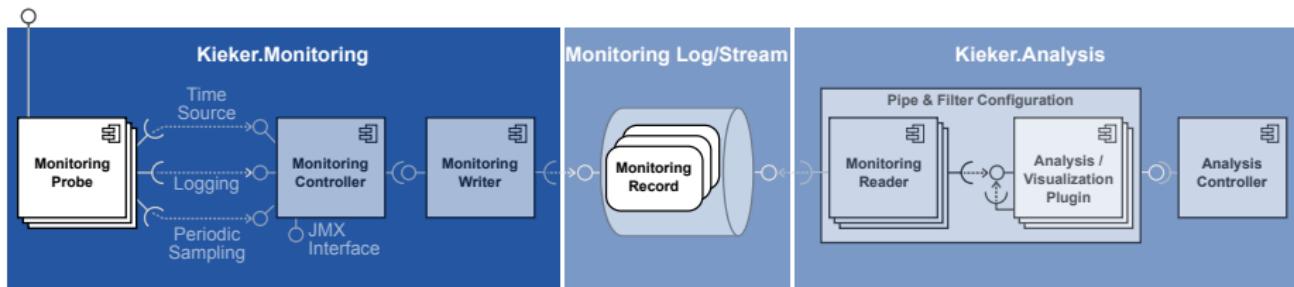
- ① Chapter 2 (Download and installation)
- ② Chapter 2 (Bookstore example)
- ③ Chapter 5 (AspectJ-based instrumentation)
- ④ Chapter 5 (TraceAnalysis tool)
- ⑤ Appendix A (Wrapper scripts)



# Core Kieker Framework Components



Introduction and Overview of Approach > Interactive: Quick Start



## Java probes/samplers:

Monitoring ProbesSamplers	Control-flow tracing		Manual instrumentation	
	AspectJ	Spring	Servlet	CXF/SOAP
<your interception technology>				
Resource monitoring	Servlet	Sigar	CPU utilization	Memory usage
			<your technology>	
	<your monitoring probe>			

## + basic adapters for

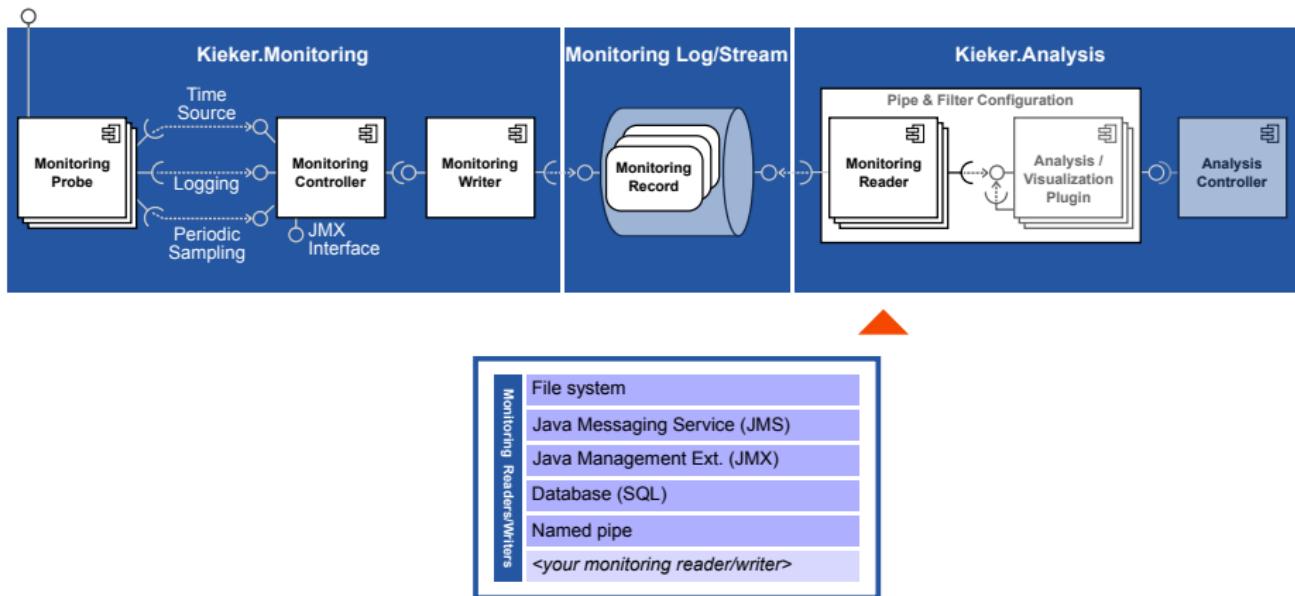
- C#/.NET
- Visual Basic 6/COM
- COBOL

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

# Core Kieker Framework Components



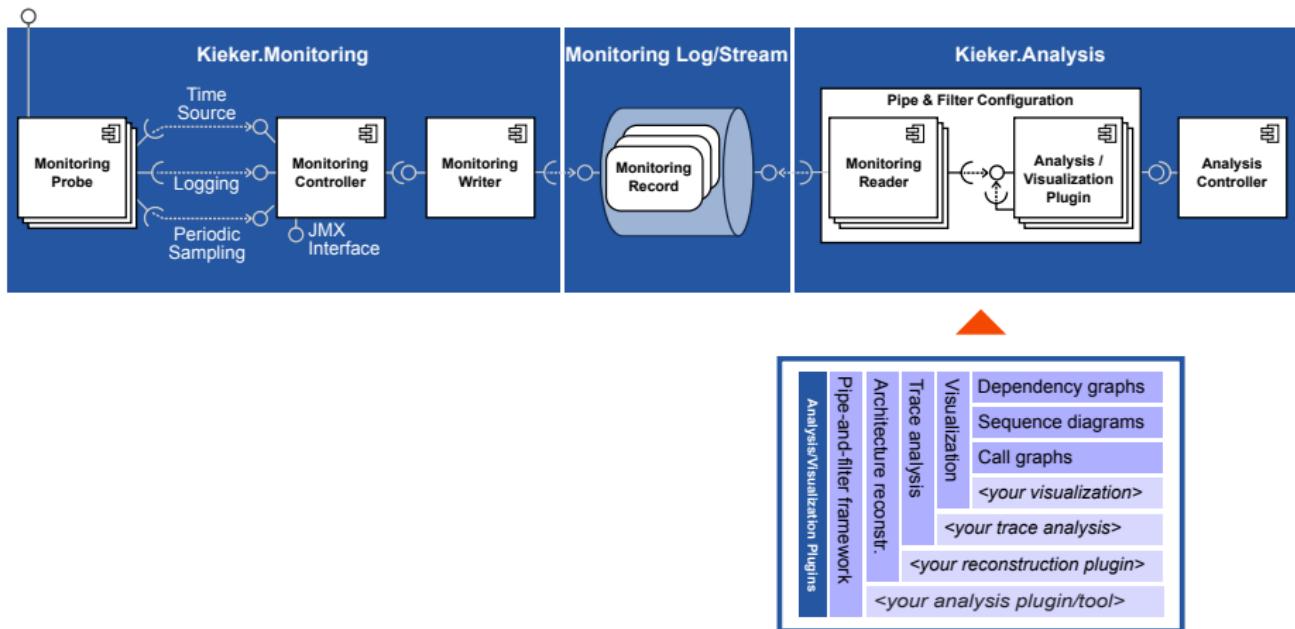
Introduction and Overview of Approach > Interactive: Quick Start



# Core Kieker Framework Components



Introduction and Overview of Approach > Interactive: Quick Start



- 1 Introduction and Overview of Approach
  - Interactive: Quick Start
- 2 Use Cases in Research and Practice
- 3 Kieker's Monitoring Component
- 4 Kieker's Analysis Component & WebGUI
  - Interactive: WebGUI
- 5 Interactive: Java EE Monitoring with Kieker
- 6 A Detailed Look at Selected Use Cases

## ① Architecture Discovery (Dynamic/Hybrid Analysis)

- Extraction of **architectural models** (structure, behavior)
- **Reverse engineering** of legacy systems
- Software **visualization** (2D/3D, static/interactive)
- Trace-based **architecture analysis**

## ② Application Performance Management

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

## ③ Characteristics (cross-cutting)

- Modular, flexible, and **extensible** architecture
- Non-intrusive instrumentation
- Low performance **overhead**
- Model-driven instrumentation and analysis
- Evaluated in lab and industrial **case studies**

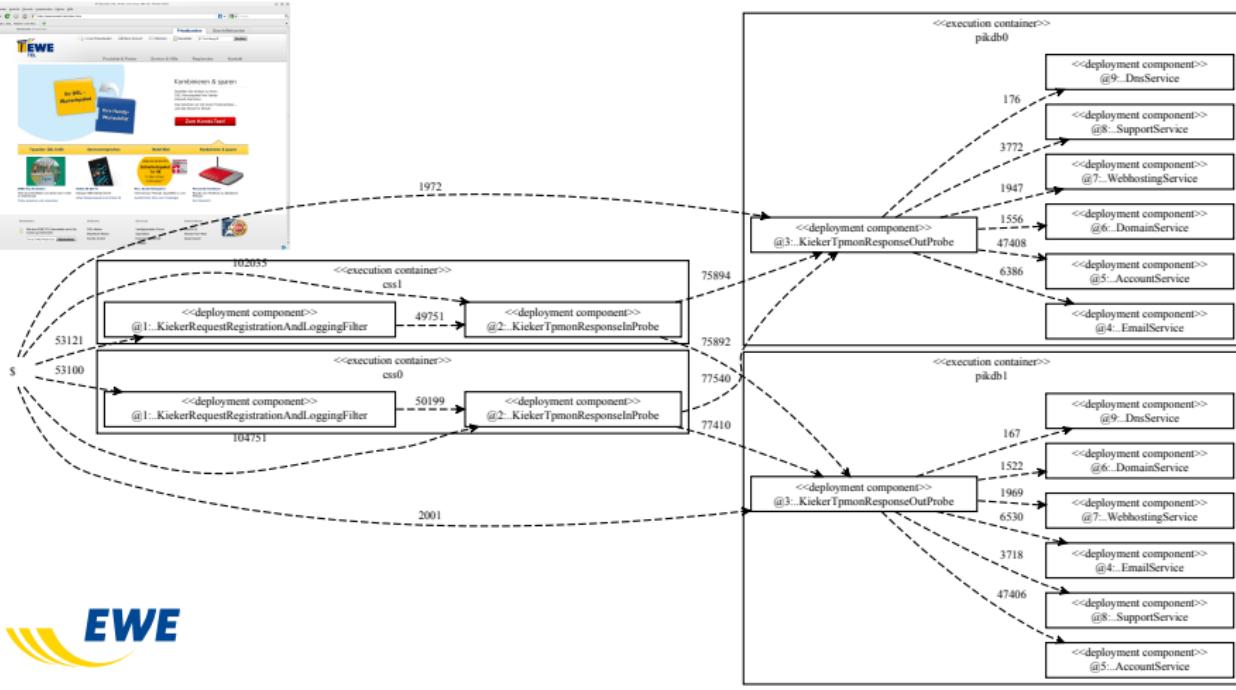
# Selected Topics and Results

[van Hoorn et al. 2009]

Use Cases in Research and Practice

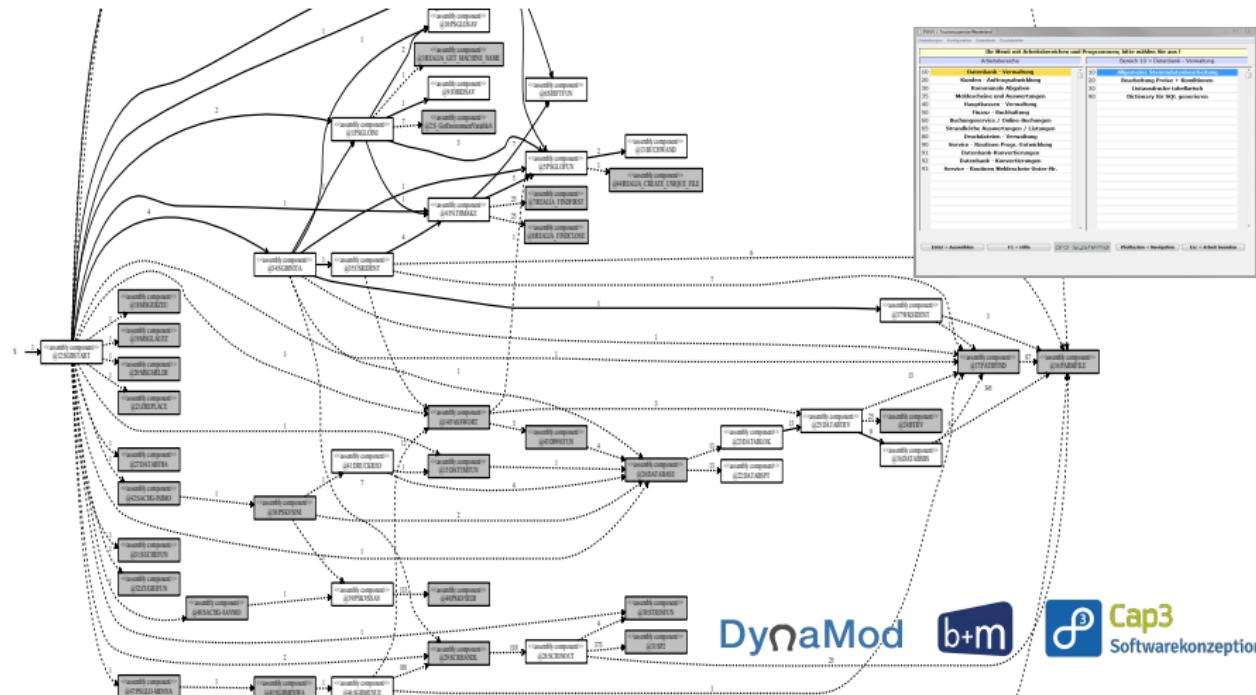
Kieker

## Architecture Discovery: Model Extraction + Visualization (cont'd)

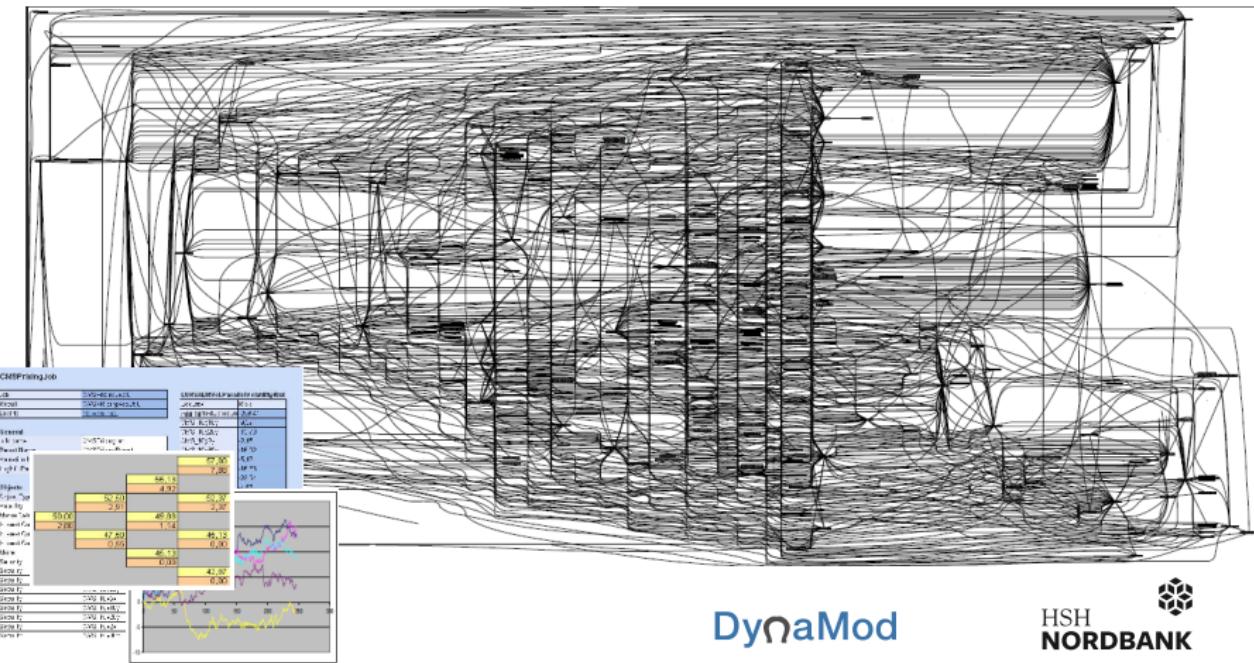


**EWE**

## Architecture Discovery: Model Extraction + Visualization (cont'd)



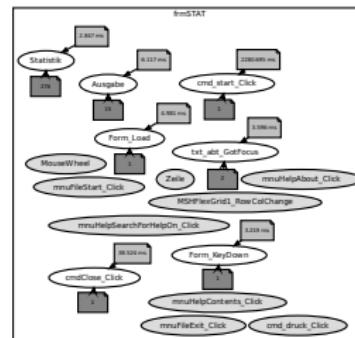
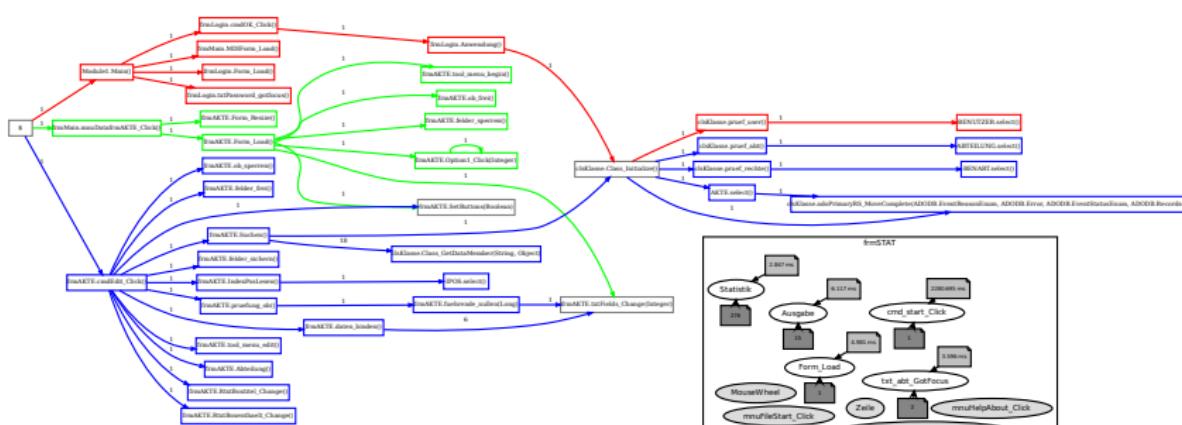
## Architecture Discovery: Model Extraction + Visualization (cont'd)



DynaMod

HSH  
NORDBANK

## Architecture Discovery: Model Extraction + Visualization (cont'd)



# DynaMod

b+m

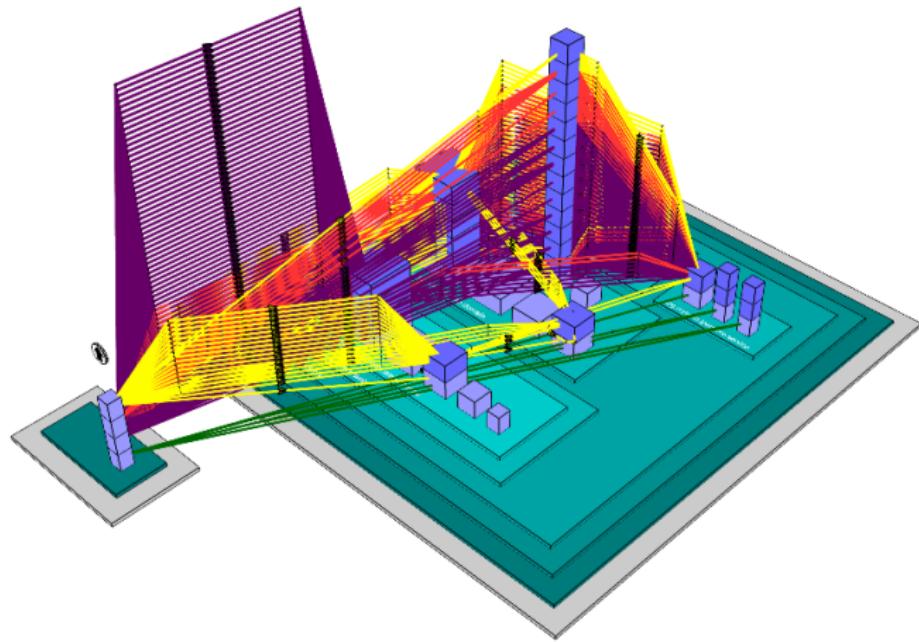
dataport

# Selected Topics and Results (cont'd)

[Döhring 2012, Waller et al. 2013] (based on [Wulf 2010])

Use Cases in Research and Practice

## Architecture Discovery: Model Extraction + Visualization (cont'd)

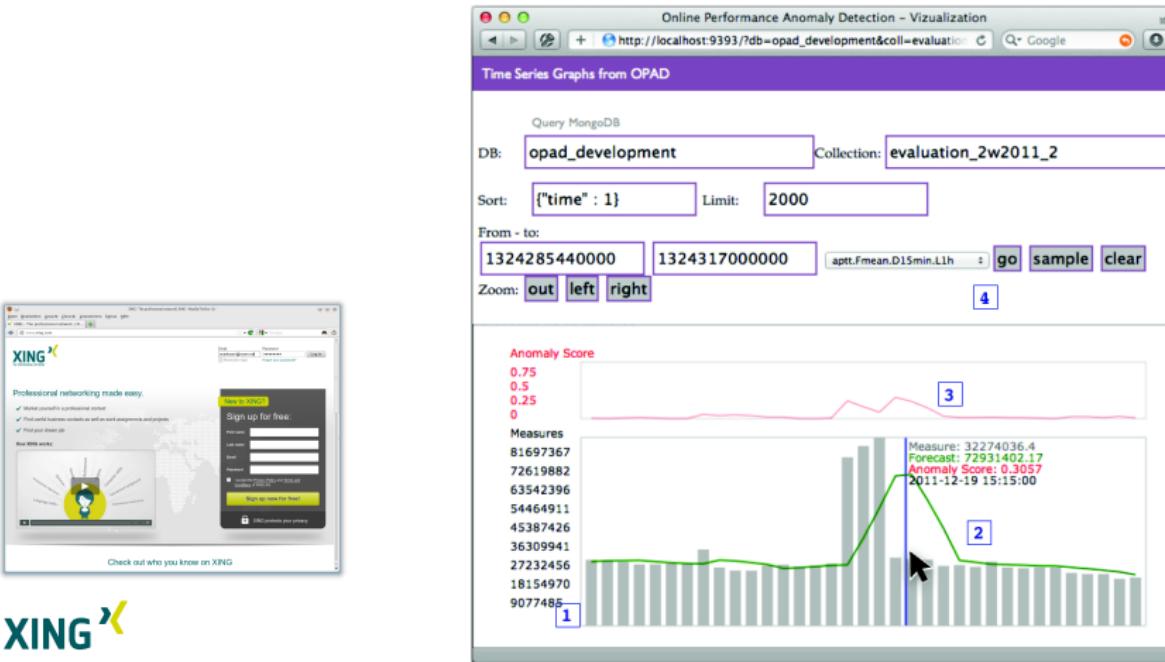


# Selected Topics and Results (cont'd)

[Bielefeld 2012, Frotscher 2013]

Use Cases in Research and Practice

## APM: Anomaly Detection + Diagnosis (cont'd)



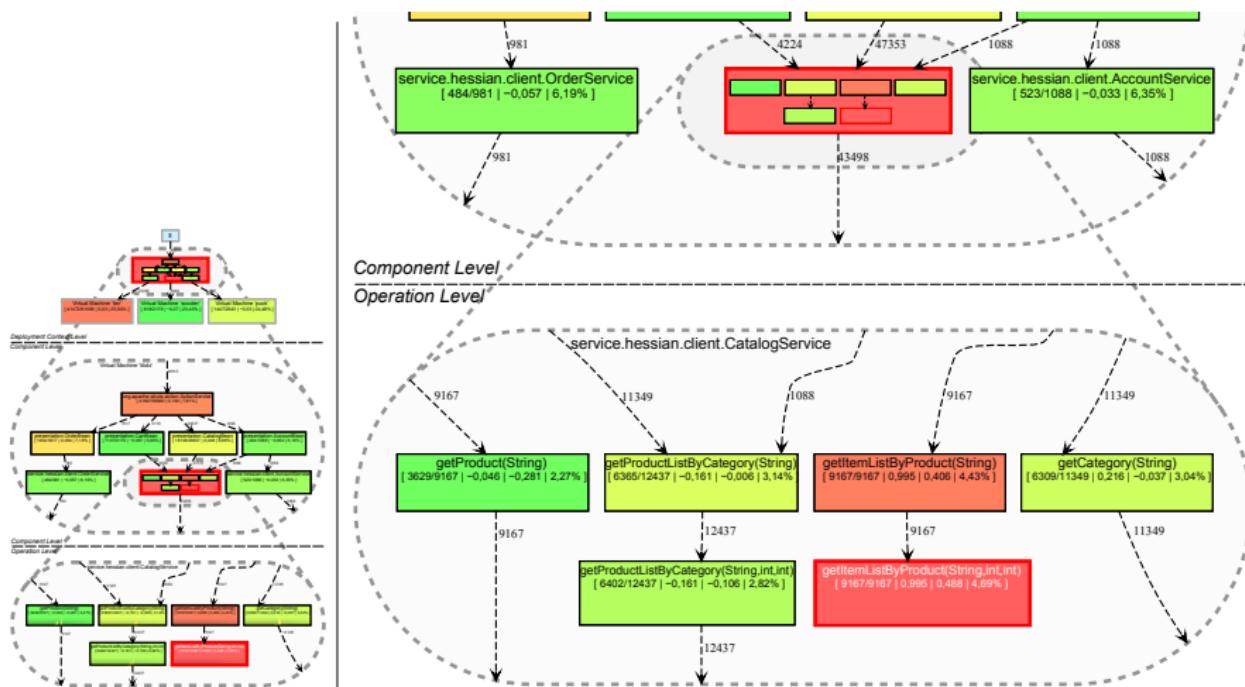
# Selected Topics and Results (cont'd)

[Marwede et al. 2009]

Use Cases in Research and Practice



## APM: Anomaly Detection + Diagnosis (cont'd)



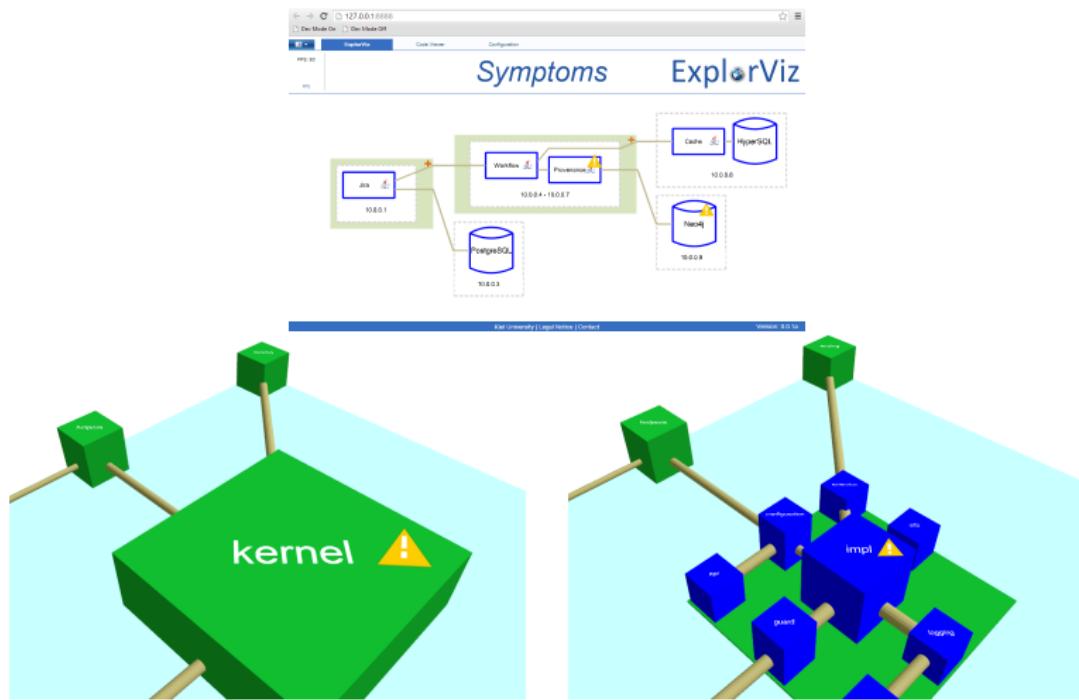
# Selected Topics and Results (cont'd)

[Fittkau et al. 2013; 2014]

Use Cases in Research and Practice



## APM: Anomaly Detection + Diagnosis (cont'd)

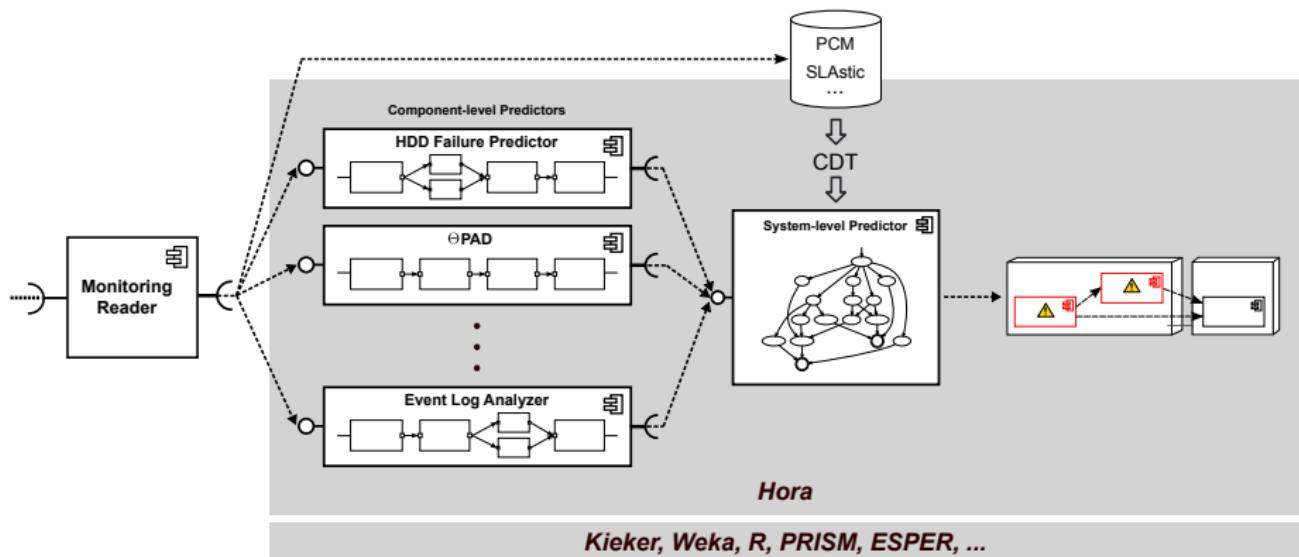


# Selected Topics and Results (cont'd)

[Pitakrat 2013, Pitakrat et al. 2014]

Use Cases in Research and Practice

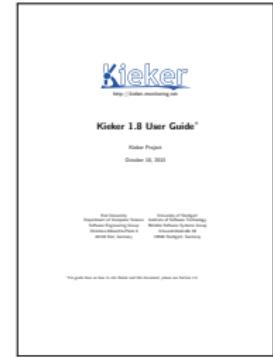
## APM: Anomaly Detection + Diagnosis (cont'd)



- 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)



# Program Instrumentation (Here: Manual)



Example: Monitoring Operation Executions

Kieker's Monitoring Component

Application code:

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

Monitoring probe code (schematic):

```
// 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

### ▶ 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)`

JMX

Registry

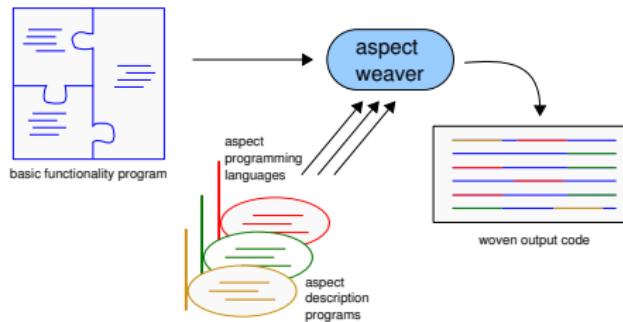
### Controller State

- + boolean: `isMonitoringEnabled()`
- + boolean: `isMonitoringTerminated()`
- + boolean: `disableMonitoring()`
- + boolean: `enableMonitoring()`
- + boolean: `terminateMonitoring()`
- + String: `getHostname()`
- + String: `toString()`

### Time Source

- + TimeSource: `getTimeSource()`

## Kieker's Monitoring Component



```
11  
12     @OperationExecutionMonitoringProbe  
13     public void getOffers() {  
14         catalog.getBook(false);  
15     }  
16 }
```

**Annotation-based (AOP) instrumentation** for monitoring trace information

## Listing 1: META-INF/aop.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:

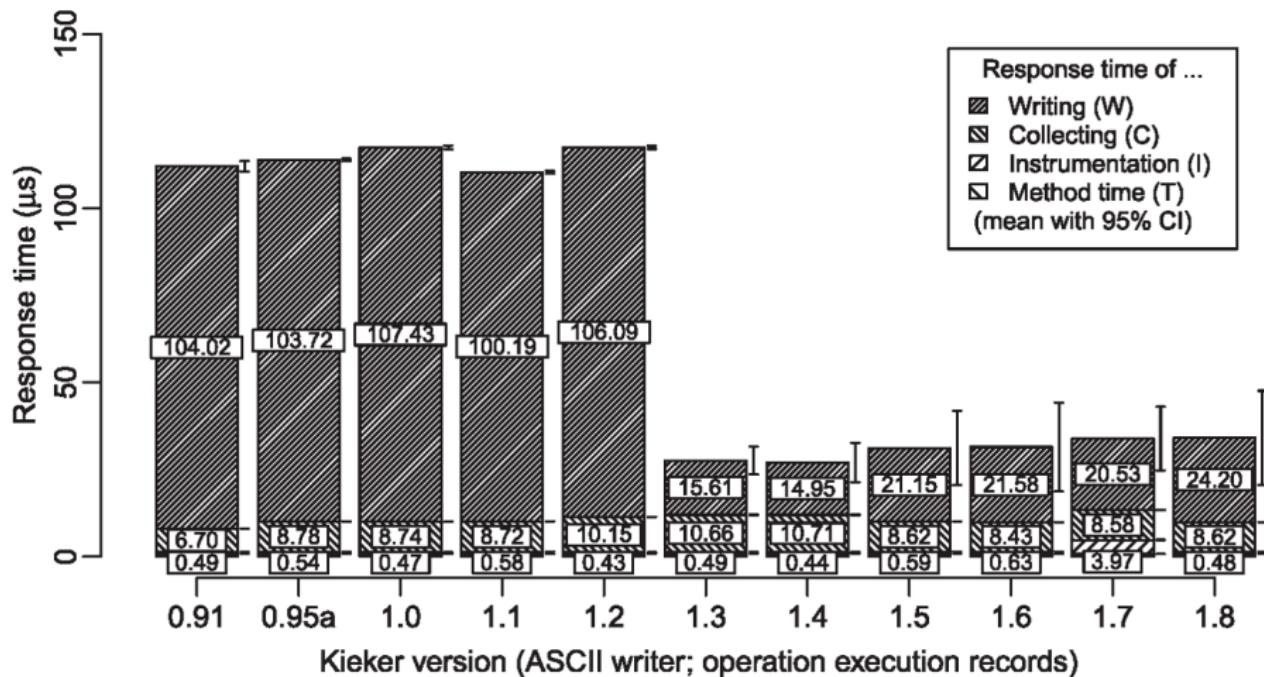
```
$ java -javaagent:lib/kieker-1.8_aspectj.jar BookstoreStarter
```

# Monitoring Overhead

[Waller and Hasselbring 2013]

Kieker's Monitoring Component

Kieker



- 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

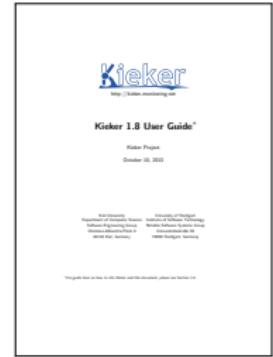
# Details on the Analysis Component



Kieker's Analysis Component & WebGUI

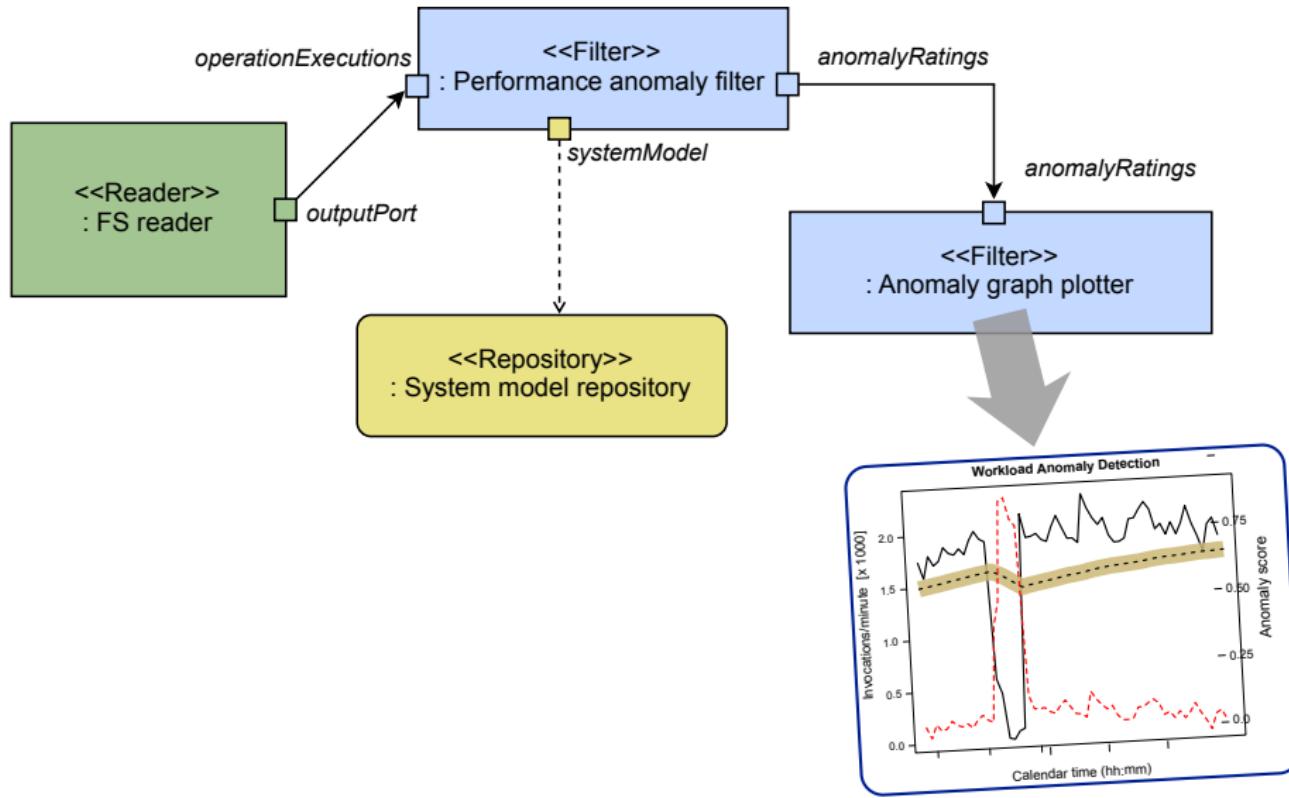
## Also refer to the Kieker User Guide

- 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



# Programmatic Analysis Creation



Kieker's Analysis Component & WebGUI

```
/* 1. Create analysis controller for our response time analysis. */
final AnalysisController analysisController
    = new AnalysisController();
```

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

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

final MyPipeReader reader =
    new MyPipeReader(readerConfig, analysisController);
```

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

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

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

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

analysisController.connect(reader, MyPipeReader.OUTPUT_PORT_NAME,
    filter, MyResponseTimeFilter.INPUT_PORT_NAME_RESPONSE_TIMES);
```

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

```
/* 5. Start the analysis. */
analysisController.run();
```

# Own Analysis Components



Kieker's Analysis Component & WebGUI

```
public final class CountingFilter extends AbstractFilterPlugin {  
}
```

# Own Analysis Components (cont'd)



Kieker's Analysis Component & WebGUI

```
@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();  
    }  
}
```

# Own Analysis Components (cont'd)



Kieker's Analysis Component & WebGUI

```
@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:**

- + `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)`



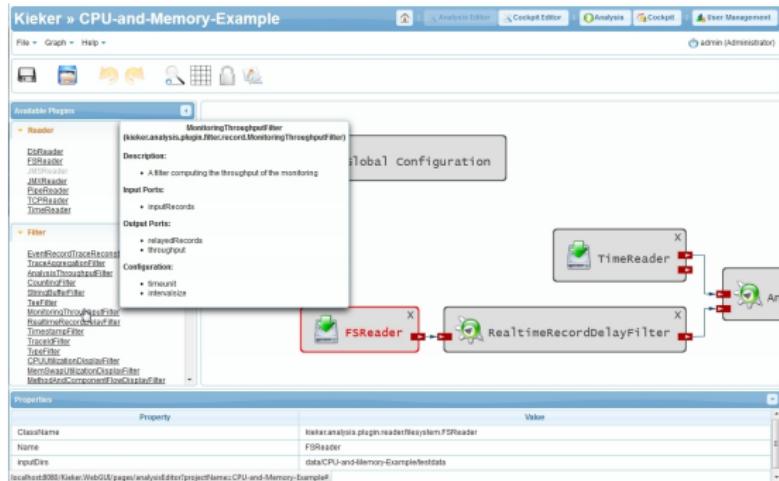
## Also refer to:

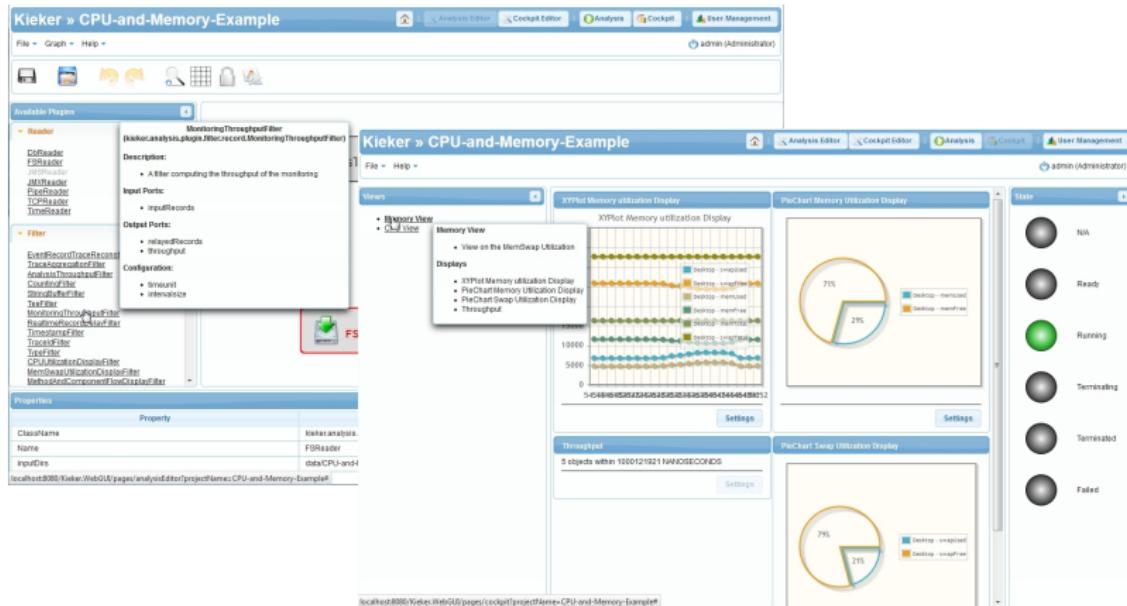
- ① Example projects included in the WebGUI
- ② Tutorial paper: Ehmke [2013]
- ③ Blog article

<http://kieker-monitoring.net/blog/>

[everything-in-sight-kiekers-webgui-in-action/](http://everything-in-sight-kiekers-webgui-in-action/)







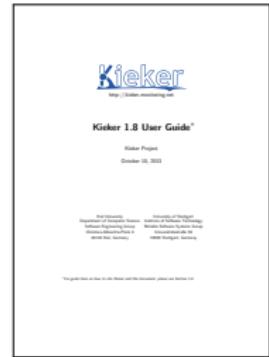
The screenshot displays the Kieker WebGUI interface, specifically the 'CPU-and-Memory-Example' project. The top navigation bar includes links for 'Analysis Editor', 'Cockpit Editor', 'Analysis', 'Cockpit', and 'User Management'. The current user is 'admin (Administrator)'. The main content area is divided into several sections:

- Available Plugins:** A tree view showing various monitoring and analysis plugins, such as 'FSEHeader', 'MonitoringThroughputFilter', 'MemoryView', and 'ThroughputView'.
- Monitoring Throughput Filter:** A detailed configuration panel for the 'MonitoringThroughputFilter' plugin, including its description, input ports ('inputRecords'), output ports ('relatedRecords', 'throughput'), and configuration options ('interval', 'intervall').
- Memory View:** A section containing three charts:
  - XYPPlot Memory utilization Display:** A dual-axis plot showing memory utilization over time for 'desktop - swapfile' and 'desktop - memfile'.
  - PieChart Memory Utilization Display:** A pie chart showing the distribution of memory utilization between 'memfile' (71%) and 'swapfile' (21%).
  - Throughput:** A table showing throughput statistics for 5 objects within 1000/21921 nanoseconds.
- State:** A vertical sidebar listing system states: N/A, Ready, Running, Terminating, and Failed, each represented by a colored circle.

- 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

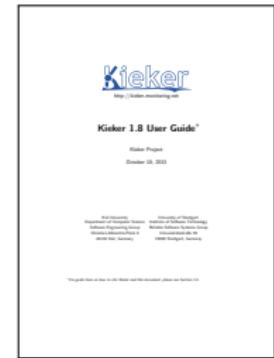
- ① Chapter 5 (AspectJ-based instrumentation)
- ② Chapter 5 (TraceAnalysis tool)
- ③ Appendix B (Java EE example)
- ④ Appendix C (Continuous analysis with JMS)
- ⑤ Appendix D (Monitoring of system metrics)



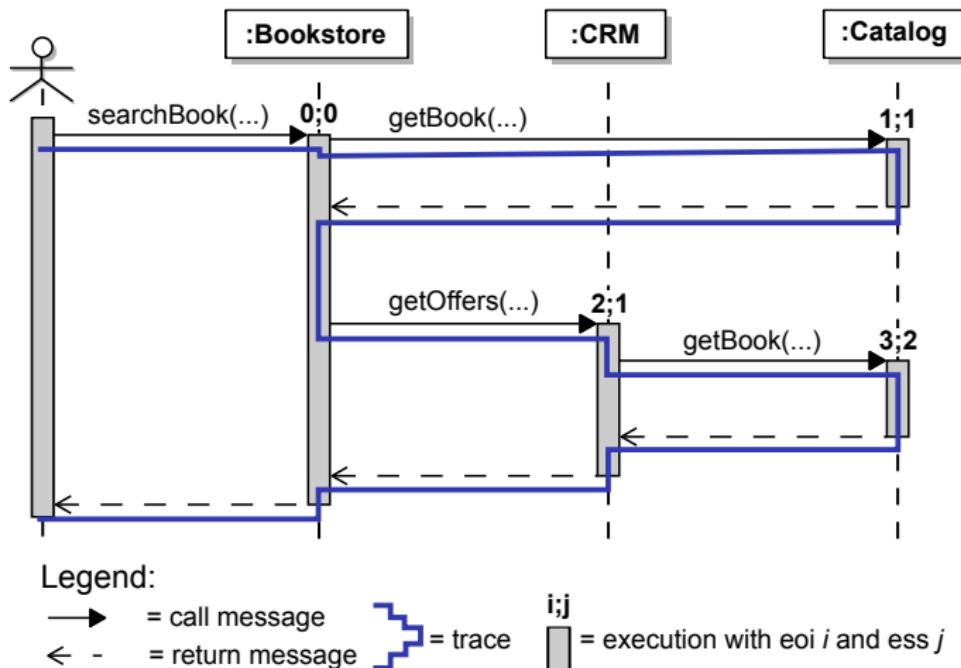
- 1 Introduction and Overview of Approach**
  - Interactive: Quick Start
- 2 Use Cases in Research and Practice**
- 3 Kieker's Monitoring Component**
- 4 Kieker's Analysis Component & WebGUI**
  - Interactive: WebGUI
- 5 Interactive: Java EE Monitoring with Kieker**
- 6 A Detailed Look at Selected Use Cases**

## Also refer to the Kieker User Guide

- ① Chapter 5 (AspectJ-based instrumentation)
- ② Chapter 5 (TraceAnalysis tool)
- ③ Paper [van Hoorn et al. 2009]
- ④ Paper [Rohr et al. 2008]



# Trace Terminology



Execution order index (eoi)  $i$ :  $i$ -th started execution in a trace

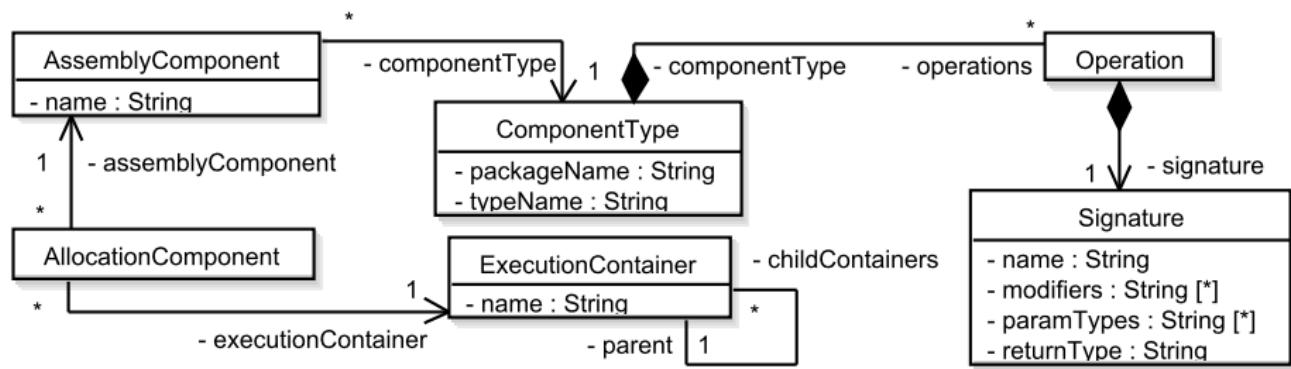
Execution stack size (ess)  $j$ : execution started at stack depth  $j$

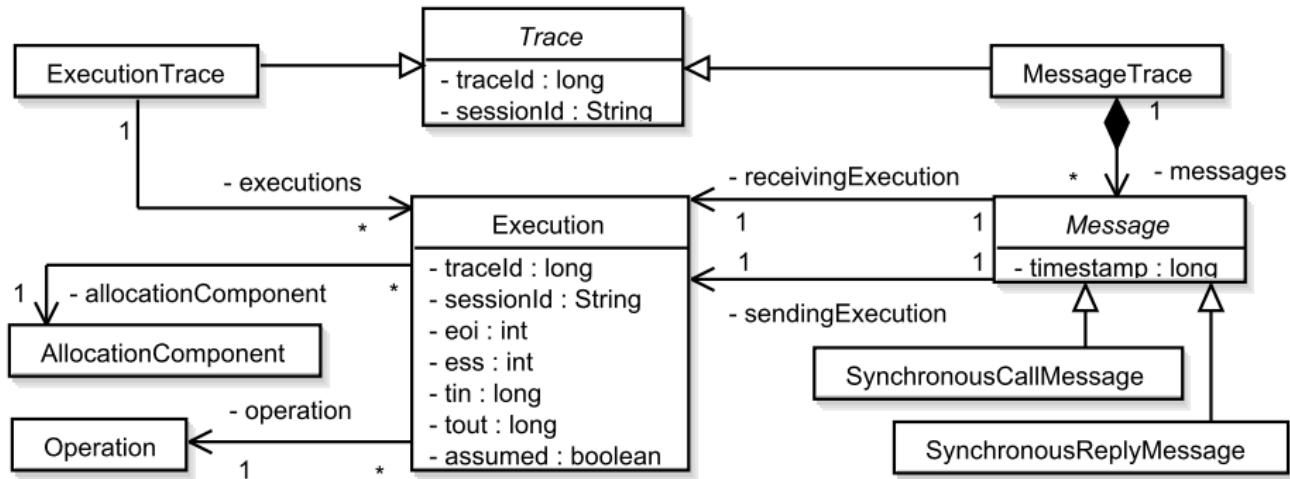
## A single line of the monitoring log – a single *monitoring record*

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

### The meaning of this record:

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





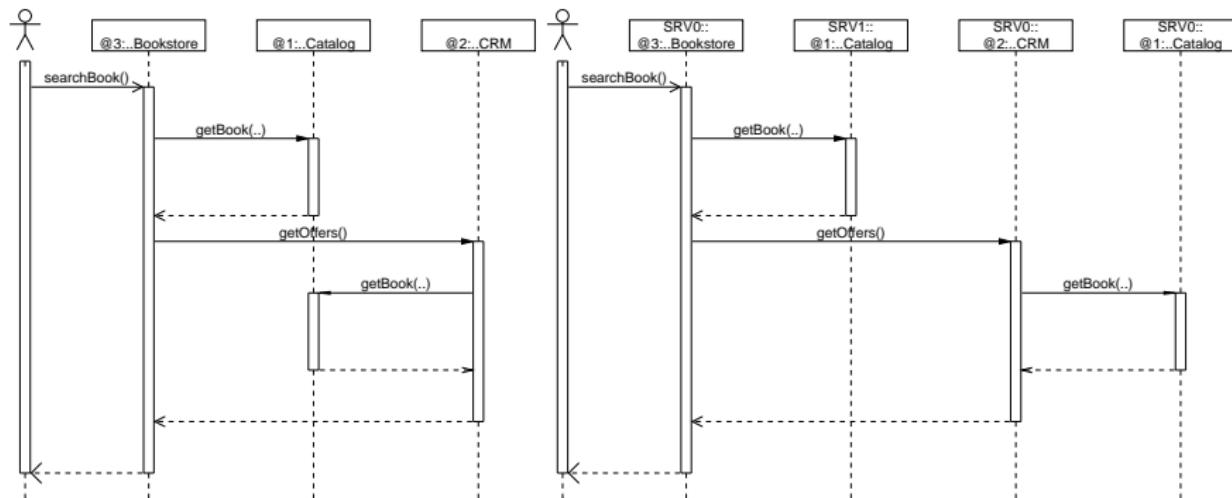
# Sequence Diagrams

Kieker. TraceAnalysis Tool



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) Assembly-level view

(b) Deployment-level view

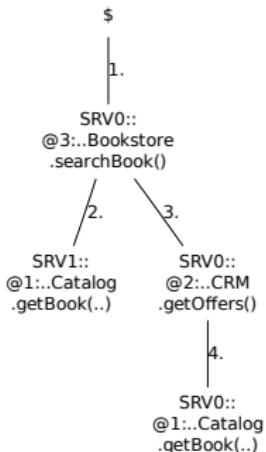
# Dynamic Call Trees

Kieker TraceAnalysis Tool (cont'd)

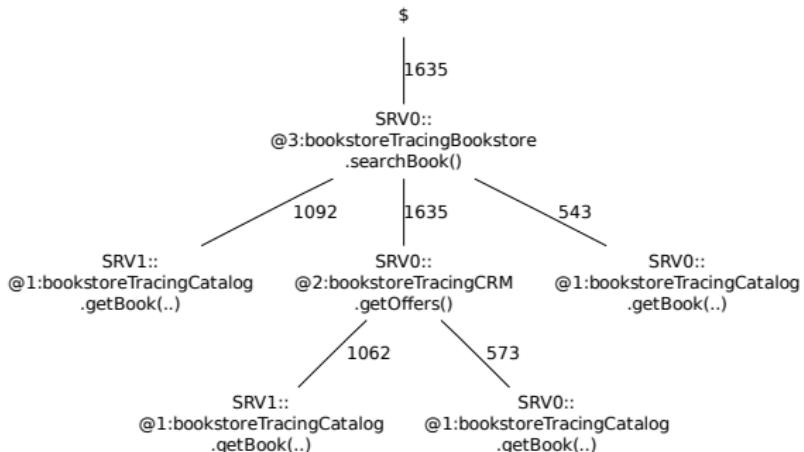


A Detailed Look at Selected Use Cases > Trace Analysis

- 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

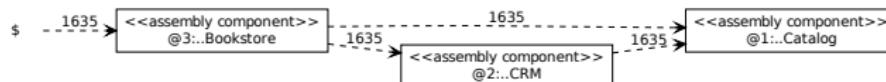
# Hierarchical Calling Dependency Graphs



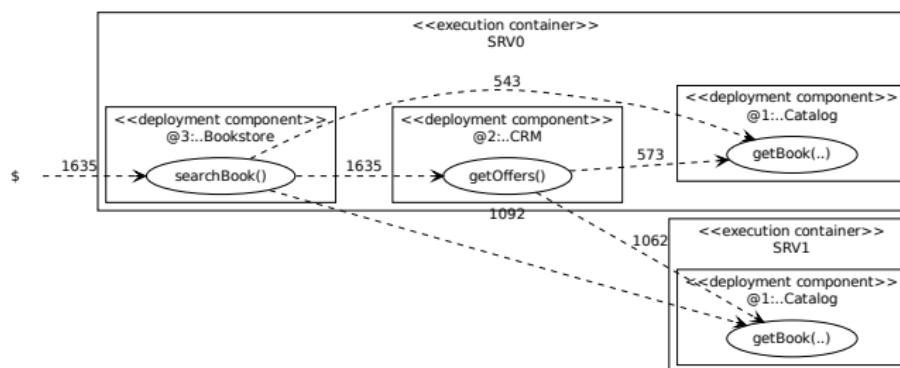
Kieker TraceAnalysis Tool (cont'd)

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) Assembly-level component dependency graph



(b) Deployment-level operation dependency graph

# System Model (HTML Representation)

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** (here: HTML representation)

The screenshot shows a Mozilla Firefox window displaying the "System Model Reconstructed by Kieker. TraceAnalysis". The page contains five tables:

- Component Types**:

ID	Package	Name	Operations
3	bookstoreTracing	Bookstore	• <a href="#">searchBook()</a>
2	bookstoreTracing	CRM	• <a href="#">getOffers()</a>
1	bookstoreTracing	Catalog	• <a href="#">getBook(boolean)</a>
- Operations**:

ID	Component type	Name	Parameter types	Return type
3	<a href="#">bookstoreTracing.Bookstore</a>	searchBook		
2	<a href="#">bookstoreTracing.CRM</a>	getOffers		
1	<a href="#">bookstoreTracing.Catalog</a>	getBook	• boolean	
- Assembly Components**:

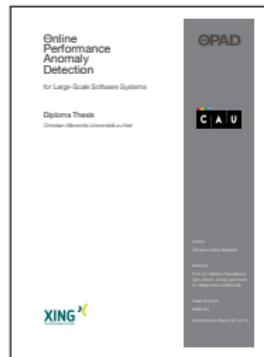
ID	Name	Component type
3	@3	<a href="#">bookstoreTracing.Bookstore</a>
2	@2	<a href="#">bookstoreTracing.CRM</a>
1	@1	<a href="#">bookstoreTracing.Catalog</a>
- Execution Containers**:

ID	Name
2	SRV0
1	SRV1
- Deployment Components**:

ID	Assembly component	Execution container
4	<a href="#">@3:bookstoreTracing.Bookstore</a>	<a href="#">SRV0</a>
3	<a href="#">@2:bookstoreTracing.CRM</a>	<a href="#">SRV0</a>
2	<a href="#">@1:bookstoreTracing.Catalog</a>	<a href="#">SRV0</a>
1	<a href="#">@1:bookstoreTracing.Catalog</a>	<a href="#">SRV1</a>

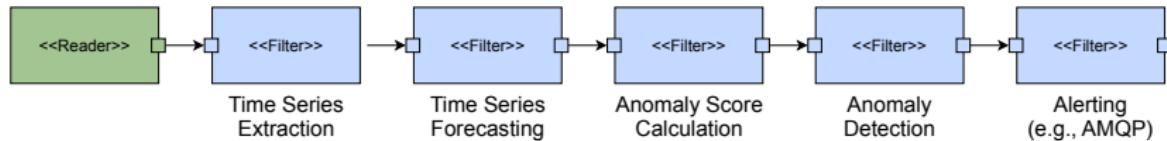
## Details to be found in

- ① Master's Thesis by Bielefeld [2012]
- ② Master's Thesis by Frotscher [2013]



# ΘPAD Processing Steps

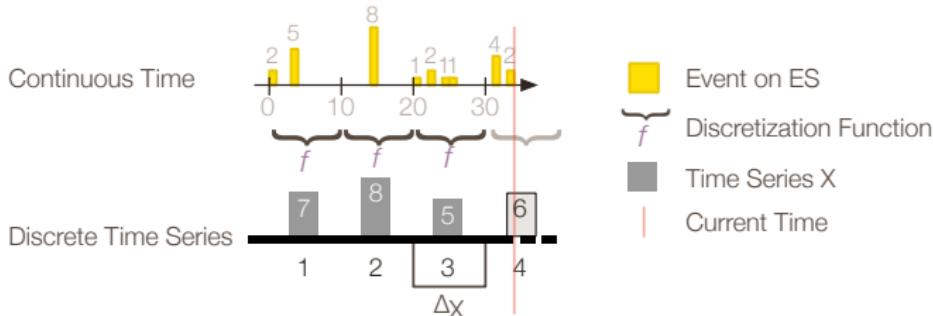
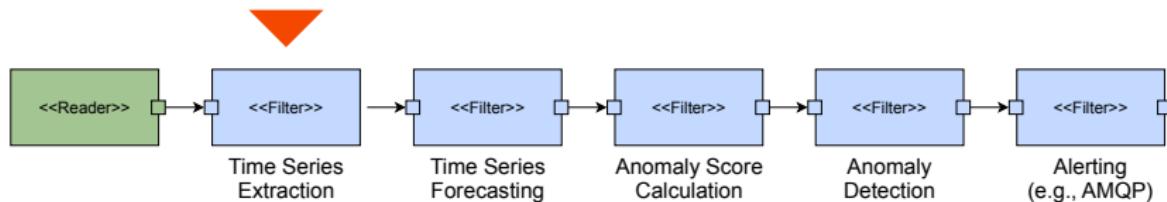
A Detailed Look at Selected Use Cases ▷ ΘPAD



# Step 1: Time Series Extraction

## ΘPAD Processing Steps (cont'd)

A Detailed Look at Selected Use Cases ▷ ΘPAD

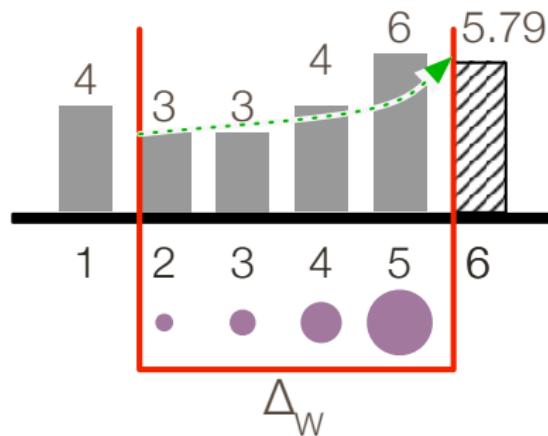
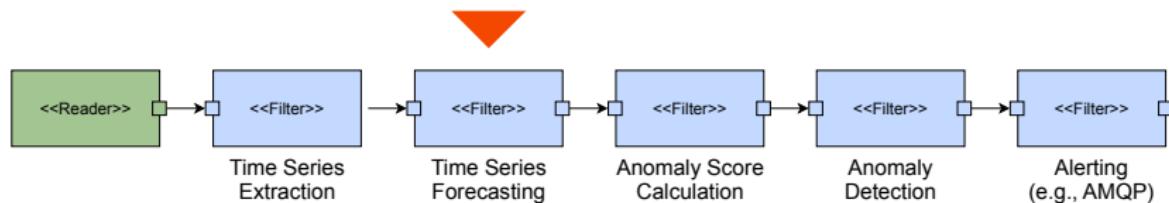


```
select sum(value) as aggregation  
from MeasureEvent.win:time_batch( 1000 msec )
```

# Step 2: Time Series Forecasting

## ΘPAD Processing Steps (cont'd)

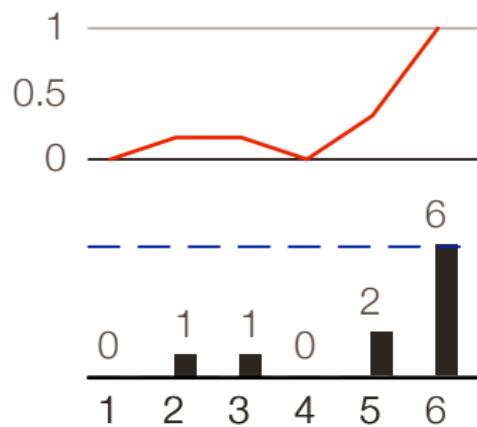
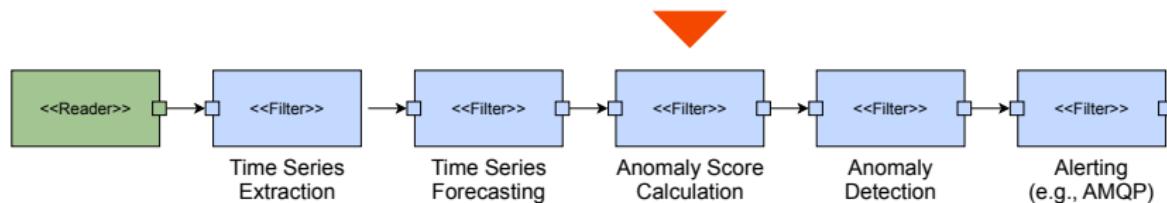
A Detailed Look at Selected Use Cases ▷ ΘPAD



# Step 3: Anomaly Score Calculation

## ΘPAD Processing Steps (cont'd)

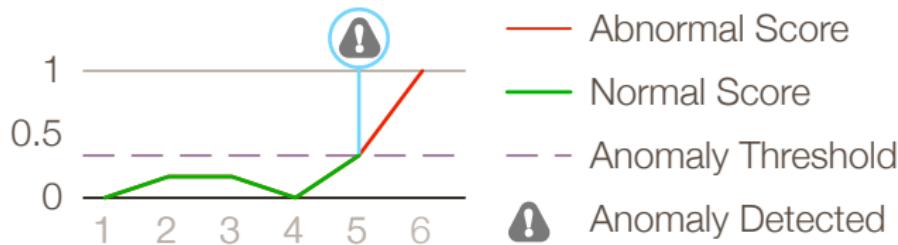
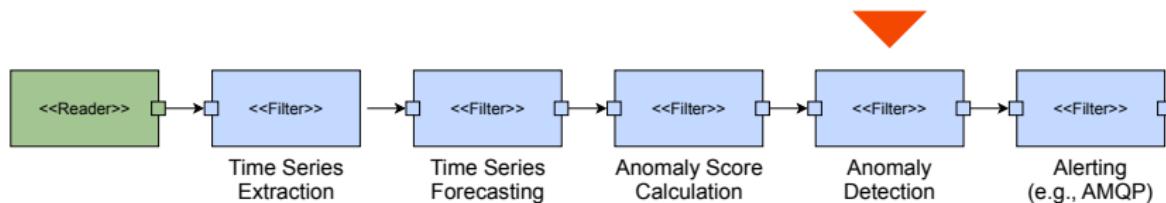
A Detailed Look at Selected Use Cases ▷ ΘPAD



# Step 4: Anomaly Detection

## ΘPAD Processing Steps (cont'd)

A Detailed Look at Selected Use Cases ▷ ΘPAD





- 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



cewe color

dataport



HSH  
NORDBANK

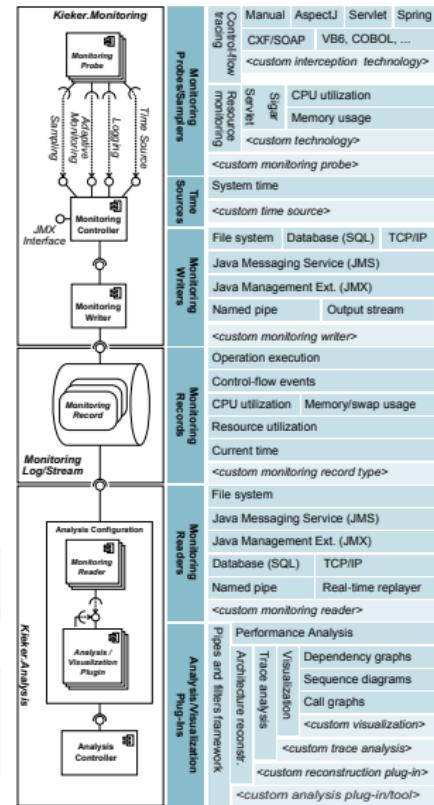


Kieker is open-source software (Apache License, V. 2.0)

<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>



- T. C. Bielefeld. Online performance anomaly detection for large-scale software systems, Mar. 2012. Diploma Thesis, Kiel University.
- P. Döhring. Visualisierung von Synchronisationspunkten in Kombination mit der Statik und Dynamik eines Softwaresystems. Master's thesis, Kiel University, Oct. 2012.
- N. C. Ehmke. Everything in sight: Kieker's WebGUI in action (tutorial). In S. Becker, W. Hasselbring, A. van Hoorn, and R. Reussner, editors, *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days (KP'DAYS '13)*, volume 1083 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.
- F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: The explorviz approach. In *1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)*, September 2013. URL <http://eprints.uni-kiel.de/21840/>.
- F. Fittkau, A. van Hoorn, and W. Hasselbring. Towards a dependability control center for large software landscapes. In *Proceedings of the 10th European Dependable Computing Conference (EDCC '14)*, 2014. To appear.
- T. Frotscher. Architecture-based multivariate anomaly detection for software systems, Oct. 2013. Master's Thesis, Kiel University.
- Kieker Project. Kieker 1.8 user guide. <http://kieker-monitoring.net/documentation/>, Oct. 2013a.
- Kieker Project. Kieker web site. <http://kieker-monitoring.net/>, 2013b.
- F. Magedanz. Dynamic analysis of .NET applications for architecture-based model extraction and test generation, Oct. 2011. Diploma Thesis, Kiel University.
- N. S. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring. Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR '09)*, pages 47–57. IEEE, Mar. 2009. ISBN 978-0-7695-3589-0. doi: 10.1109/CSMR.2009.15.
- T. Pitakrat. Hora: Online failure prediction framework for component-based software systems based on kieker and palladio. In *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days 2013, Karlsruhe, Germany, November 27-29, 2013*, volume 1083 of *CEUR Workshop Proceedings*, pages 39–48. CEUR-WS.org, 2013.
- T. Pitakrat, A. van Hoorn, and L. Grunske. Increasing dependability of component-based software systems by online failure prediction. In *Proceedings of the 10th European Dependable Computing Conference (EDCC '14)*, 2014. To appear.
- B. Richter. Dynamische Analyse von COBOL-Systemarchitekturen zum modellbasierten Testen ("Dynamic analysis of cobol system architectures for model-based testing", in German), Aug. 2012. Diploma Thesis, Kiel University.
- M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stöver, S. Giesecke, and W. Hasselbring. Kieker: Continuous monitoring and on demand visualization of Java software behavior. In *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE '08)*, pages 80–85. ACTA Press, Feb. 2008. ISBN 978-0-88986-715-4.

- A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous monitoring of software services: Design and application of the Kieker framework. Technical Report TR-0921, Department of Computer Science, University of Kiel, Germany, Nov. 2009.
- A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12)*, pages 247–248. ACM, Apr. 2012. ISBN 978-1-4503-1202-8.  
doi: 10.1145/2188286.2188326.
- J. Waller and W. Hasselbring. A benchmark engineering methodology to measure the overhead of application-level monitoring. In *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days 2013*, pages 59–68. CEUR Workshop Proceedings, November 2013. URL <http://eprints.uni-kiel.de/22326/>.
- J. Waller, C. Wulf, F. Fittkau, P. Döhring, and W. Hasselbring. Synchrovis: 3d visualization of monitoring traces in the city metaphor for analyzing concurrency. In *1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)*, September 2013. URL <http://eprints.uni-kiel.de/21839/>.
- C. Wulf. Runtime visualization of static and dynamic architectural views of a software system to identify performance problems, 2010.

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