Dynamic Analysis of Software Systems with Kieker
— A Hands-On Lecture —

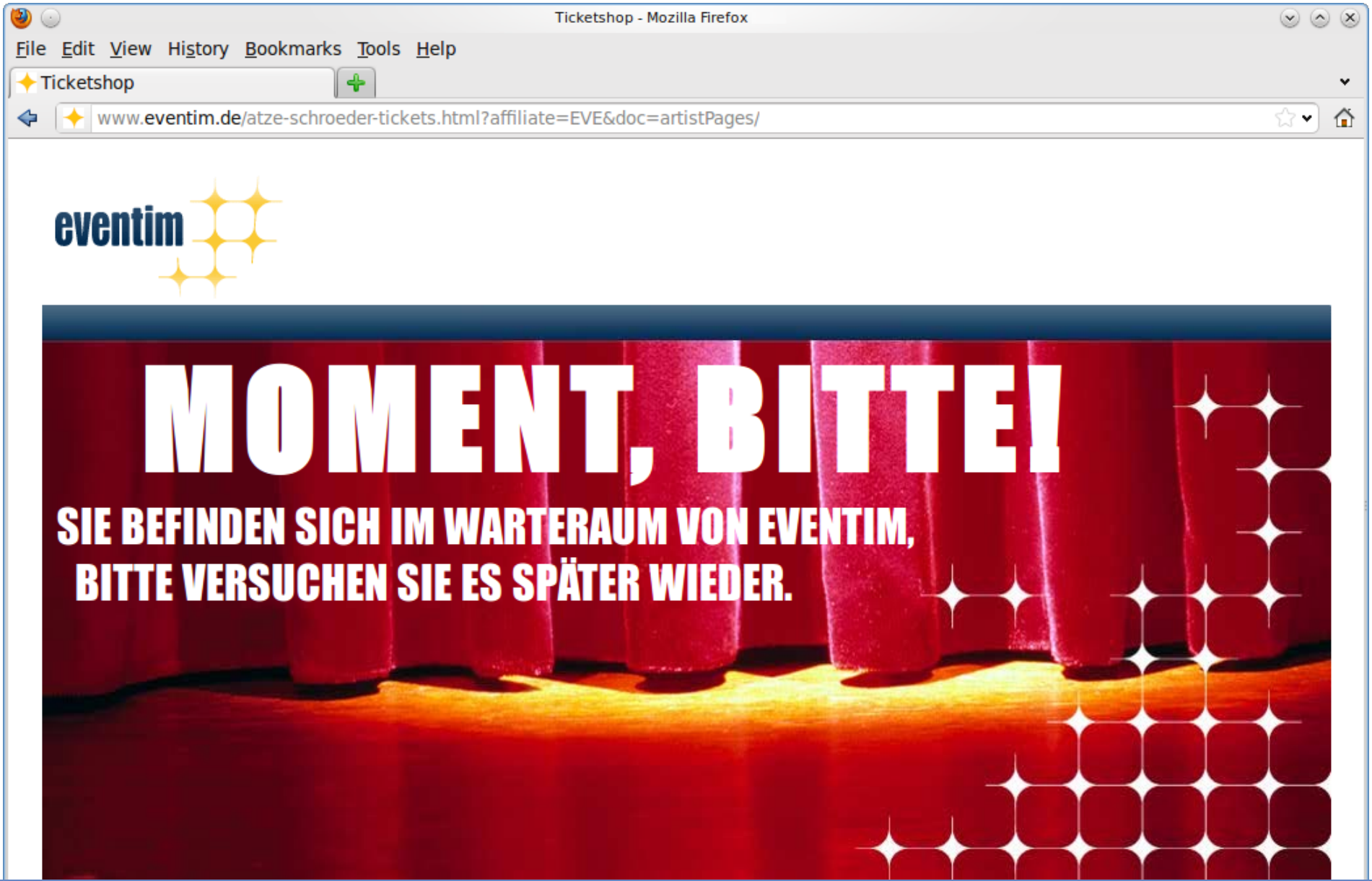
Guest Lecture in the Course “Enterprise Digital Infrastructure”

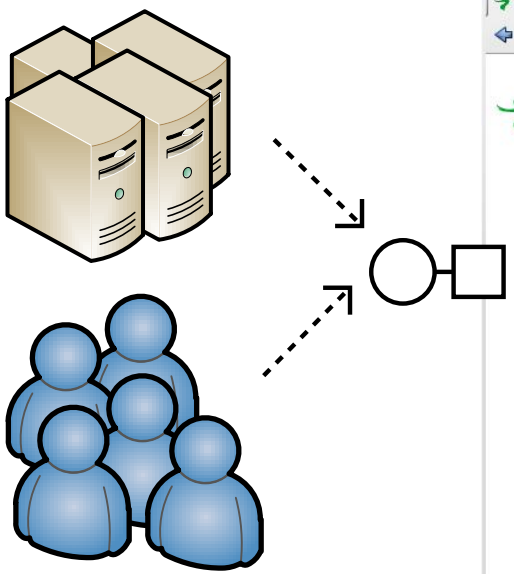
André van Hoorn
University of Stuttgart, Germany
Contact: van.hoorn@informatik.uni-stuttgart.de

(including contributions by many colleagues)

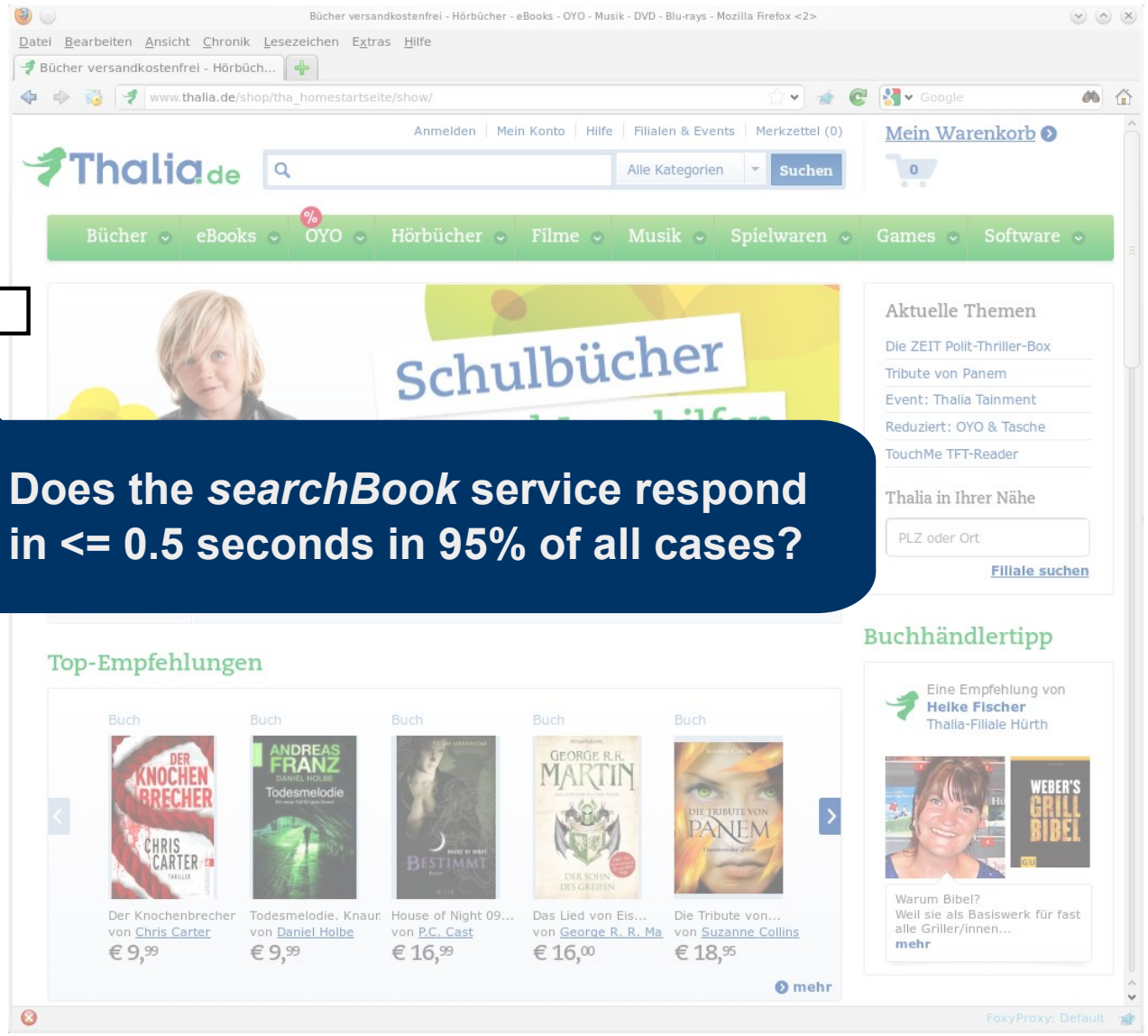
May 21, 2014 @ University of Pavia, Italy

Quality of Service (QoS) . . .

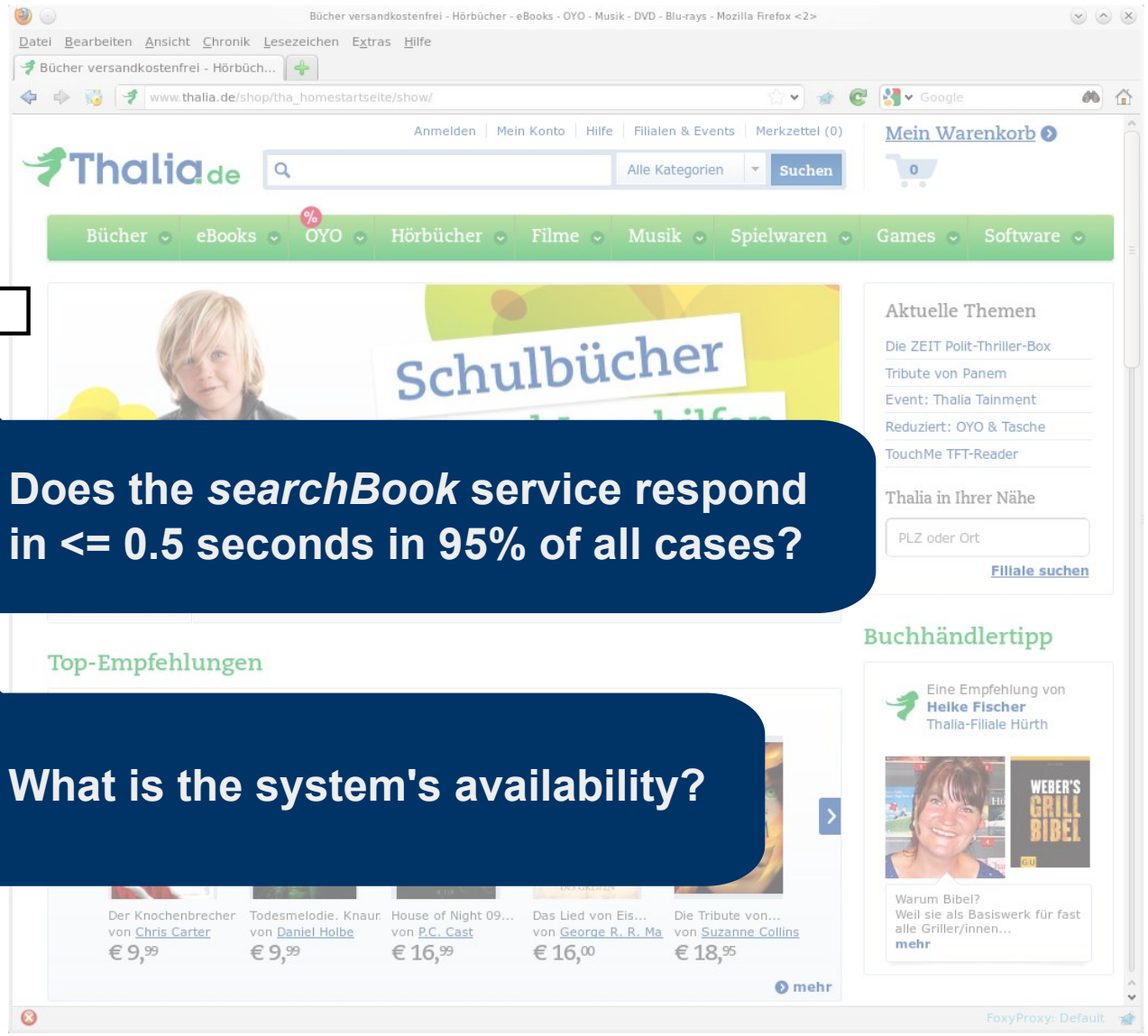





Bücher versandkostenfrei - Hörbücher - eBooks - OYO - Musik - DVD - Blu-rays - Mozilla Firefox <2>
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
Bücher versandkostenfrei - Hörbüch...
www.thalia.de/shop/tha_homestartseite/show/
Google
Anmelden | Mein Konto | Hilfe | Filialen & Events | Merkzettel (0)
Mein Warenkorb
0
Thalia.de
Alle Kategorien
Suchen
Bücher
eBooks
OYO
Hörbücher
Filme
Musik
Spielwaren
Games
Software
Schulbücher und Lernhilfen
Jetzt bestellen
Schulbücher
Europameisterschaft
Grillen & Sommerküche
Thilo Sarrazin
Taschenbücher
Aktuelle Themen
Die ZEIT Polit-Thriller-Box
Tribute von Panem
Event: Thalia Tainment
Reduziert: OYO & Tasche
TouchMe TFT-Reader
Thalia in Ihrer Nähe
PLZ oder Ort
Filiale suchen
Top-Empfehlungen
Buch
Der Knochenbrecher von Chris Carter € 9,99
Todesmelodie. Knaur von Daniel Holbe € 9,99
House of Night 09... von P.C. Cast € 16,99
Das Lied von Eis... von George R. R. Ma € 16,00
Die Tribute von... von Suzanne Collins € 18,95
mehr
Buchhändlertipp
Eine Empfehlung von
Heike Fischer
Thalia-Filiale Hürth
Warum Bibel?
Weil sie als Basiswerk für fast alle Griller/innen...
mehr
FoxyProxy: Default

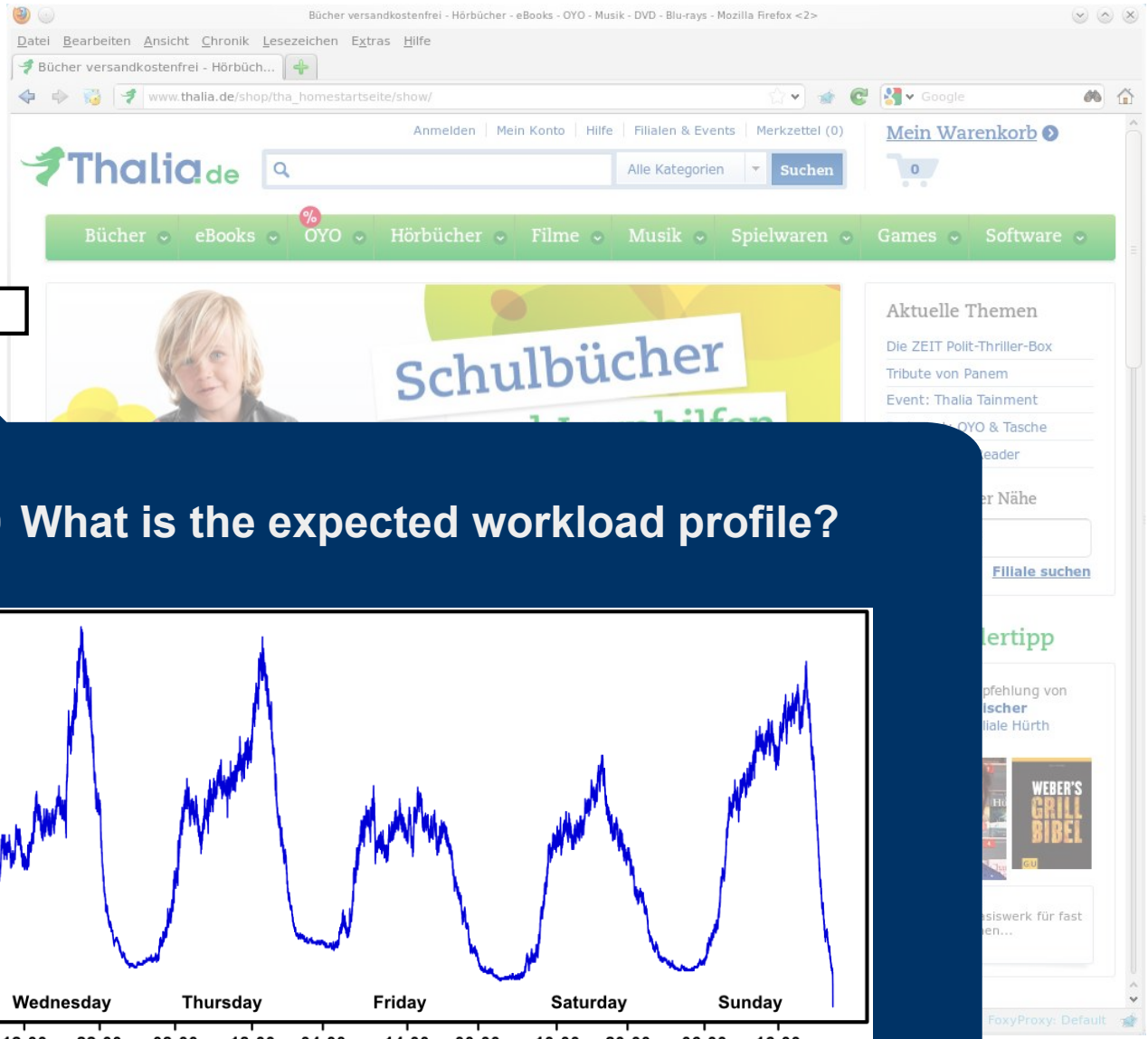


? **Does the *searchBook* service respond in <= 0.5 seconds in 95% of all cases?**

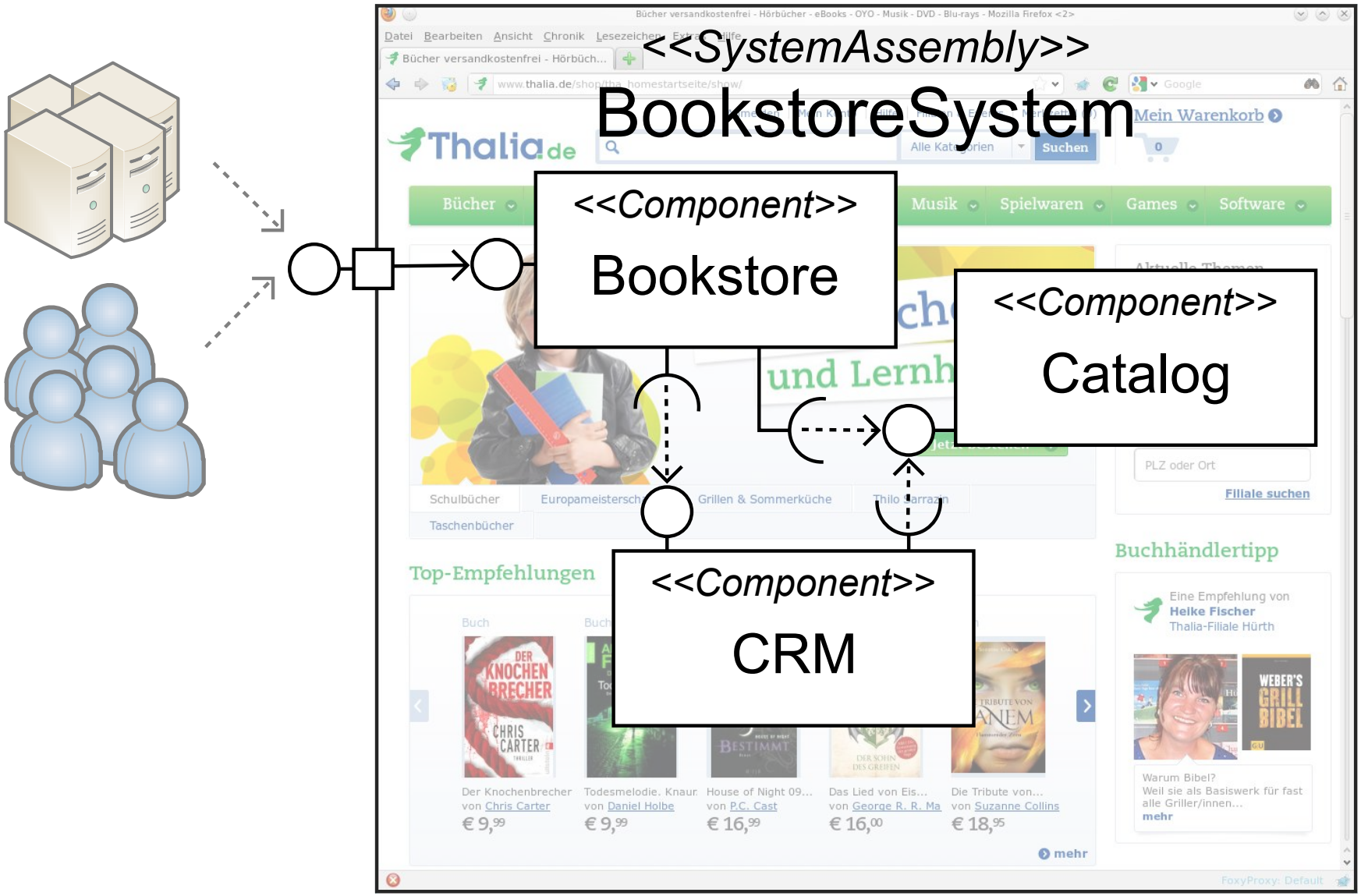


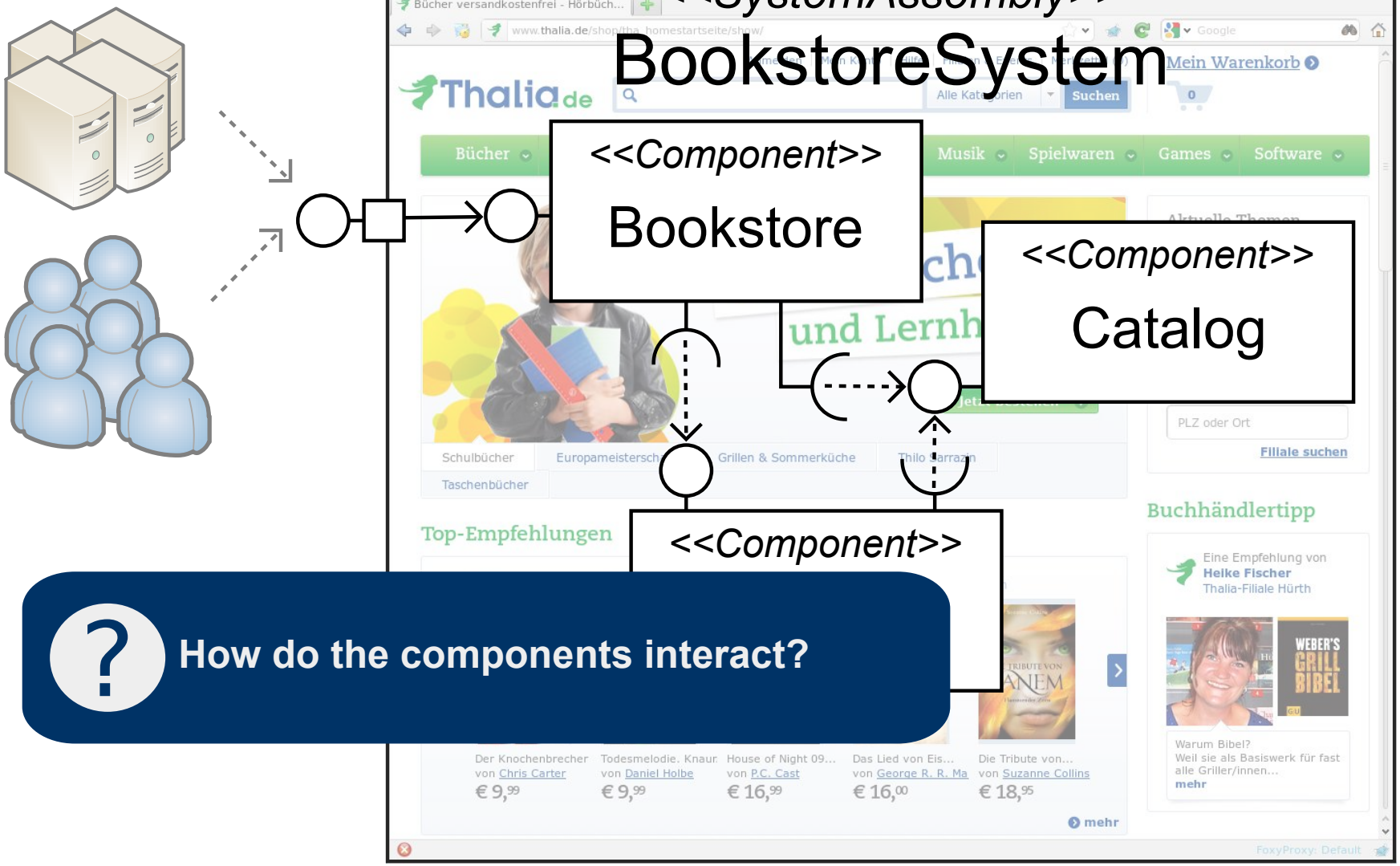
Does the *searchBook* service respond in <= 0.5 seconds in 95% of all cases?

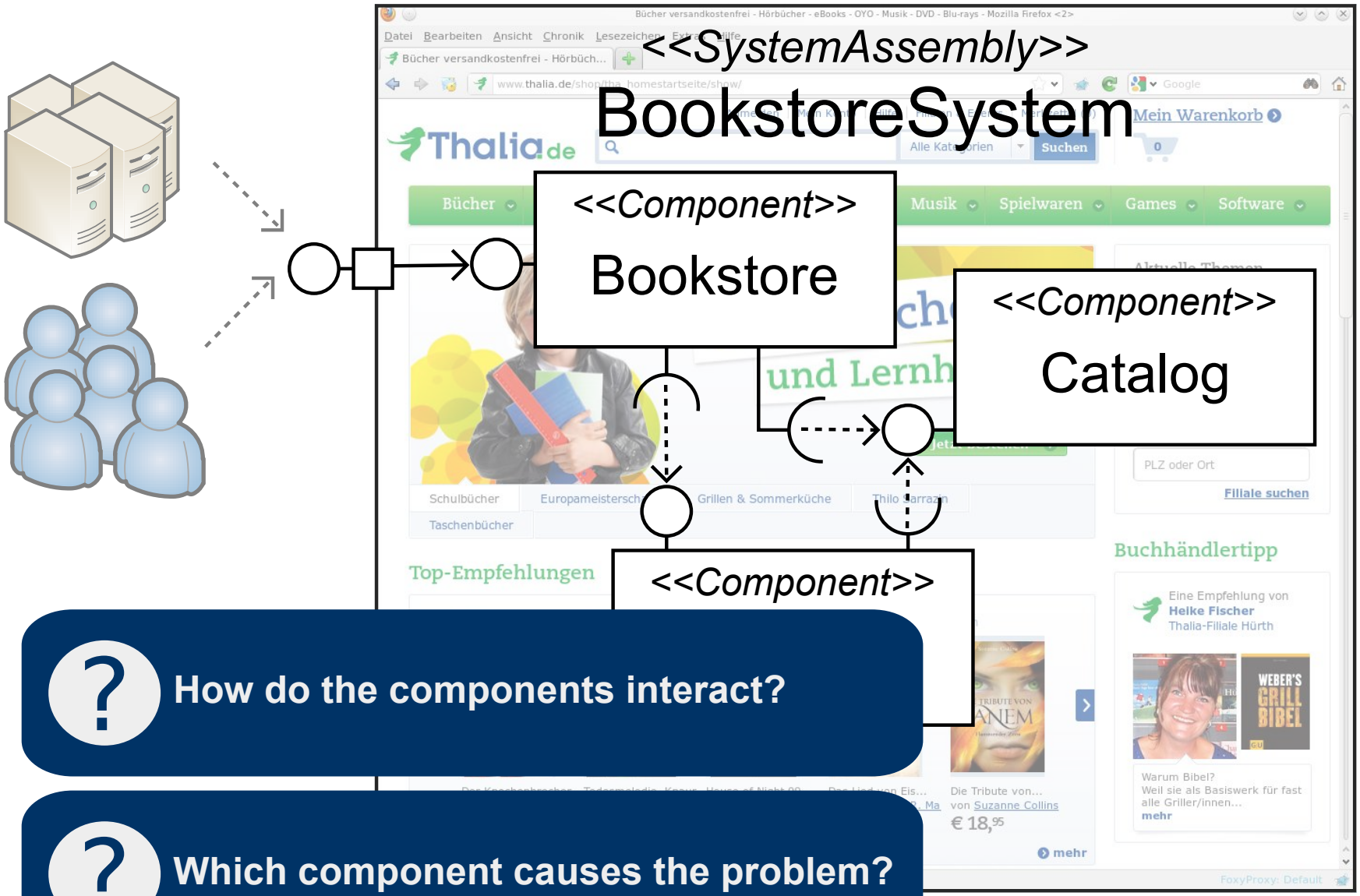
What is the system's availability?



What is the expected workload profile?


<<SystemAssembly>>
BookstoreSystem
<<Component>>
Bookstore
<<Component>>
Catalog
<<Component>>
CRM


<<SystemAssembly>>
BookstoreSystem
<<Component>>
Bookstore
<<Component>>
Catalog
<<Component>>
? How do the components interact?



How do the components interact?

Which component causes the problem?


<<SystemAssembly>>
BookstoreSystem
<<Component>>
Bookstore
<<Component>>
Catalog
CRM
How are my resources utilized?

> **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
> because it exposes the **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)

Dynamic Software Analysis (cont'd)

> 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)

Dynamic Software Analysis (cont'd)

> 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

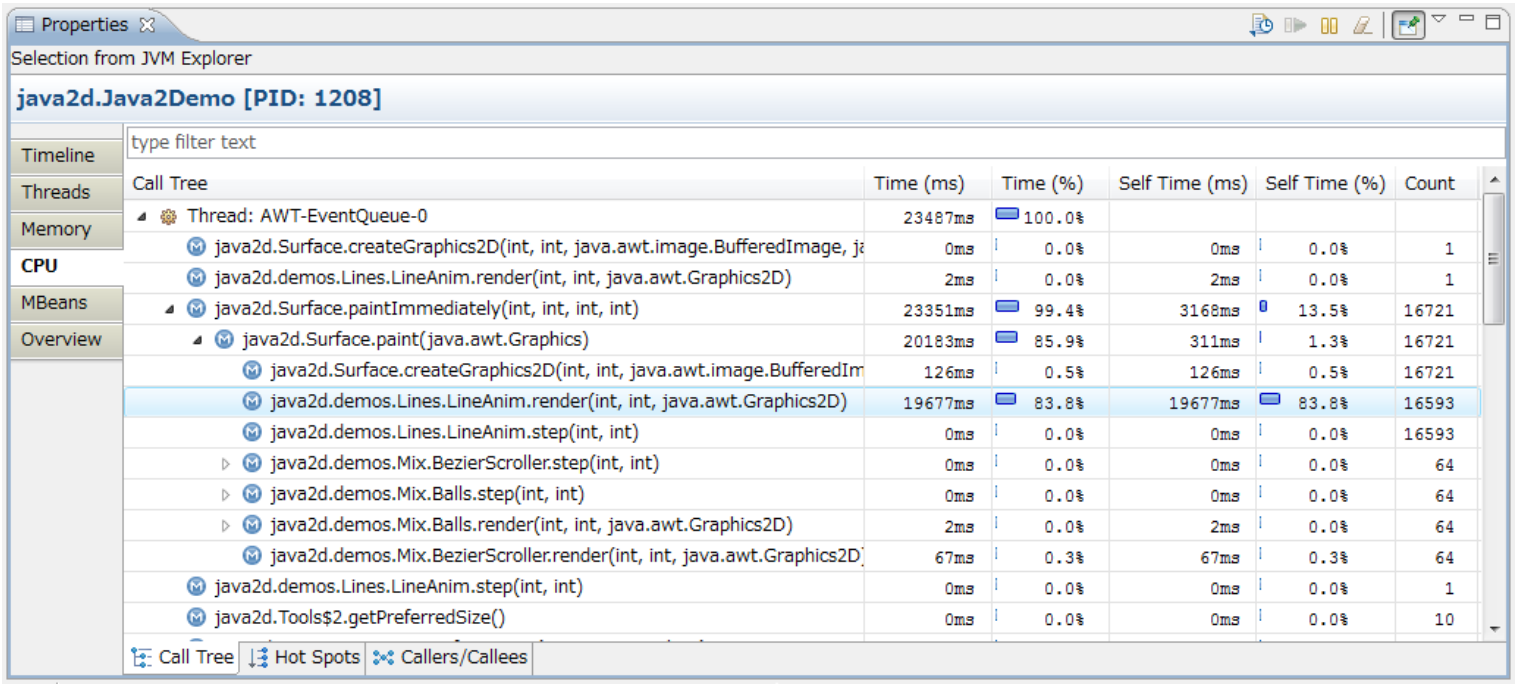
How to Gather Runtime Data from Executing Systems?

Profiling

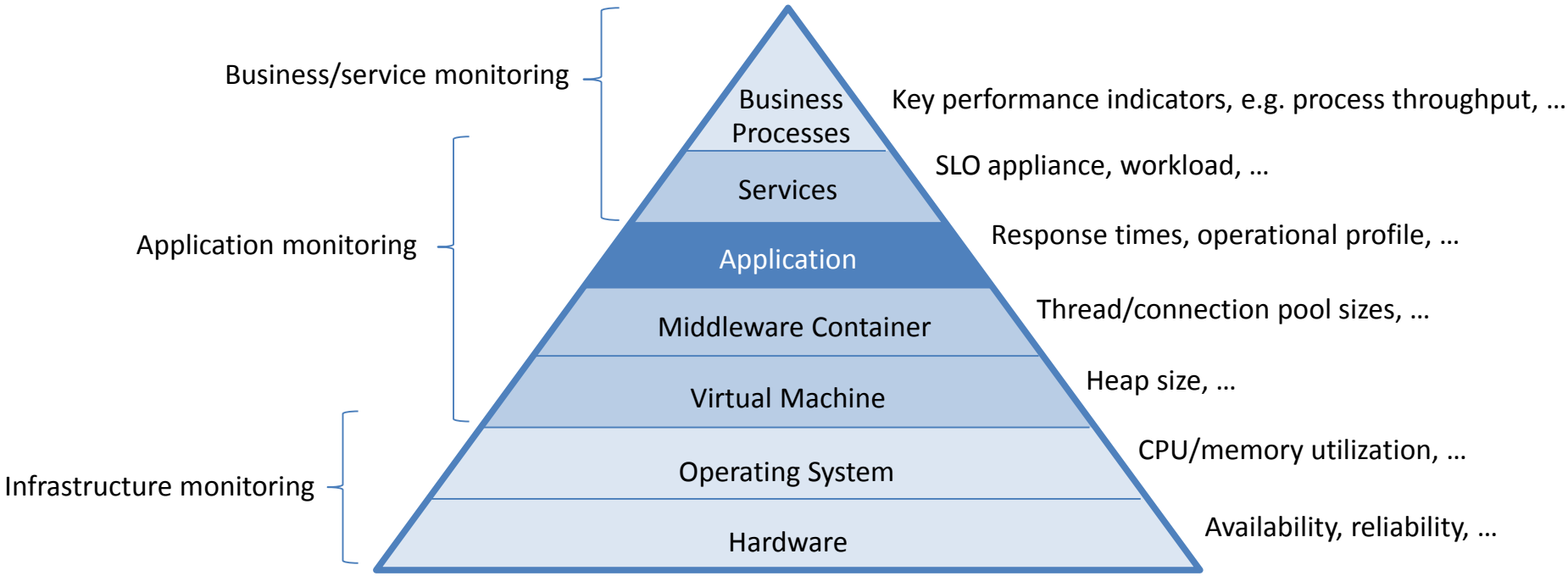
- employed in development environments
- considerable performance overhead

Monitoring

- employed in production environments
- captures real usage profile



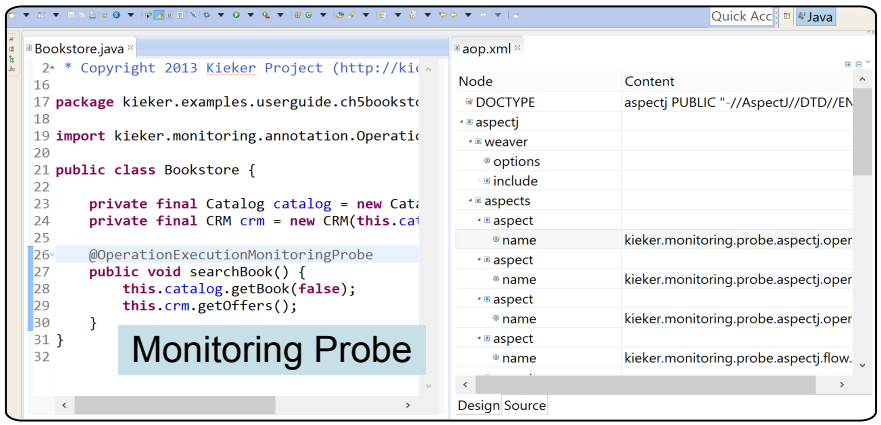
http://www.jvmmonitor.org/doc/index.html#Getting_started



Monitoring practice in the “real world” (based on what we’ve seen)

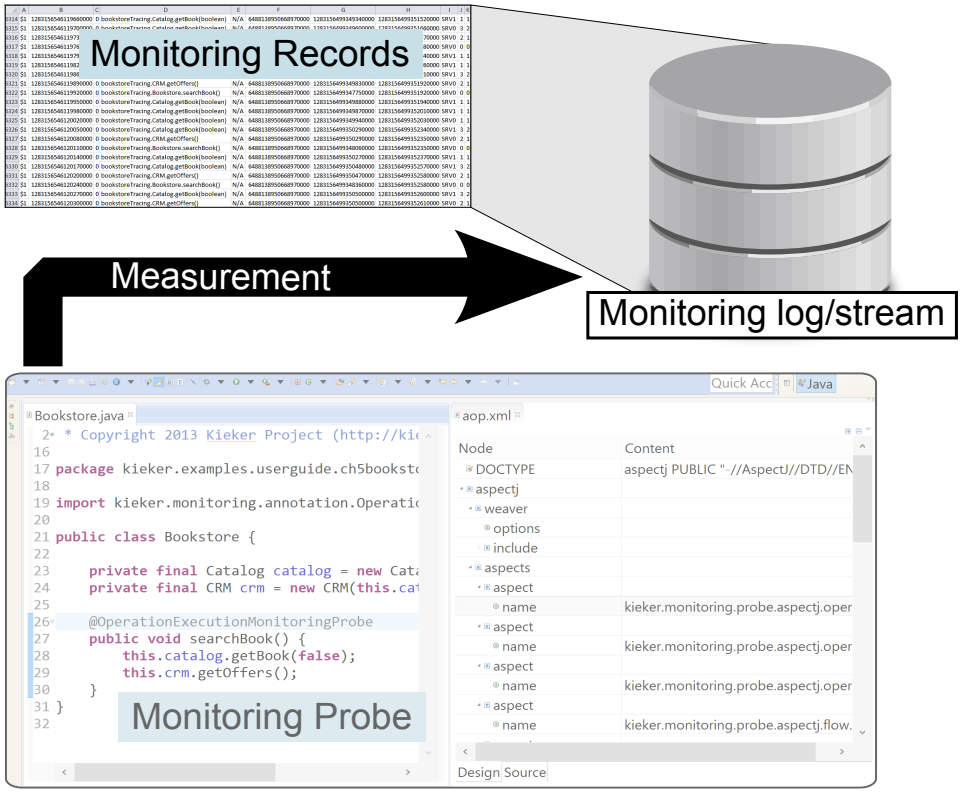
- Focus on system level metrics (availability, resource utilization)
- No systematic instrumentation on application level
- Only basic automation of analysis/alarming
- Reactive addition of monitoring probes

Kieker: Dynamic Analysis Workflow



Software System with Monitoring Instrumentation

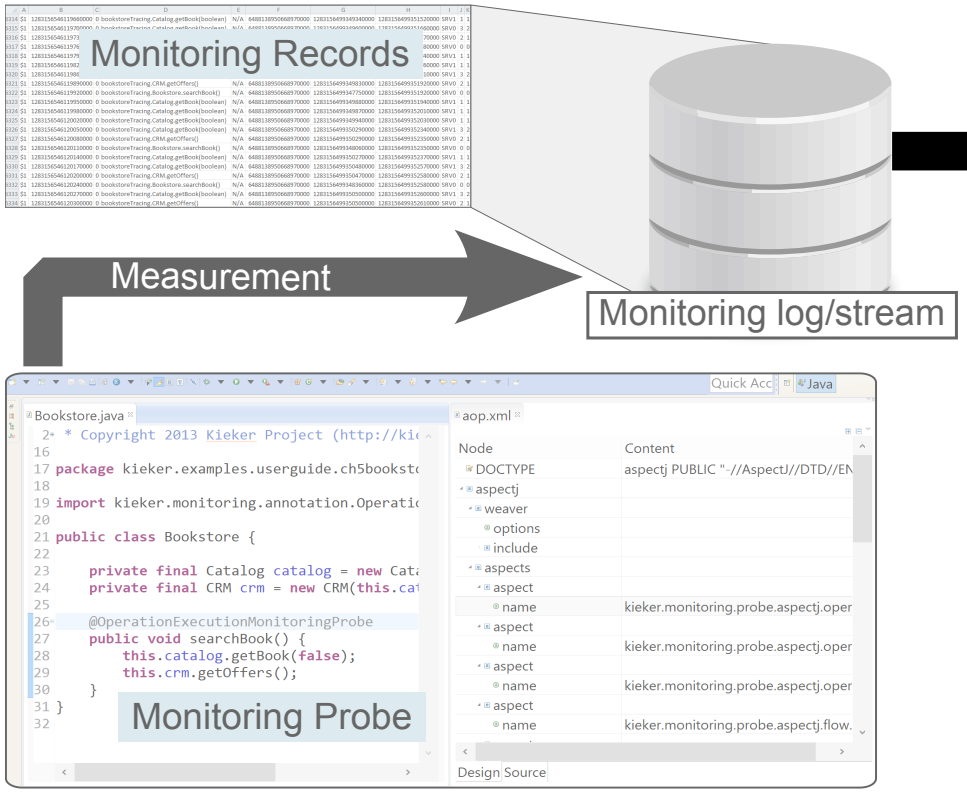
Kieker: Dynamic Analysis Workflow



Software System with Monitoring Instrumentation

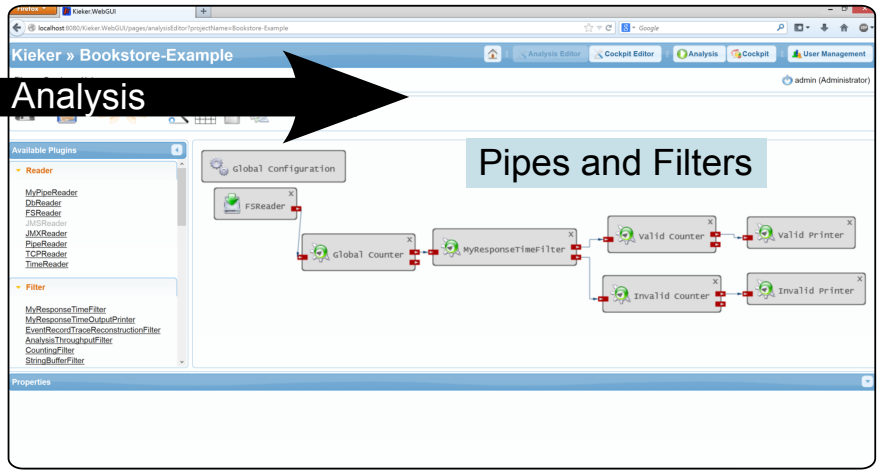
Kieker: Dynamic Analysis Workflow

Introduction and Overview of Approach



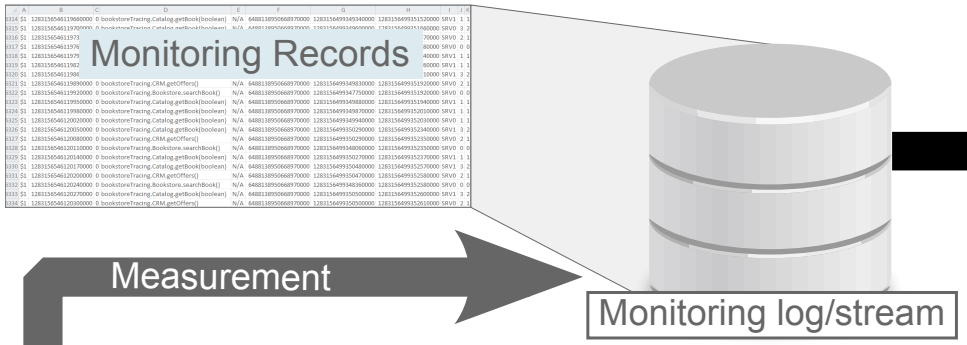
Software System with Monitoring Instrumentation

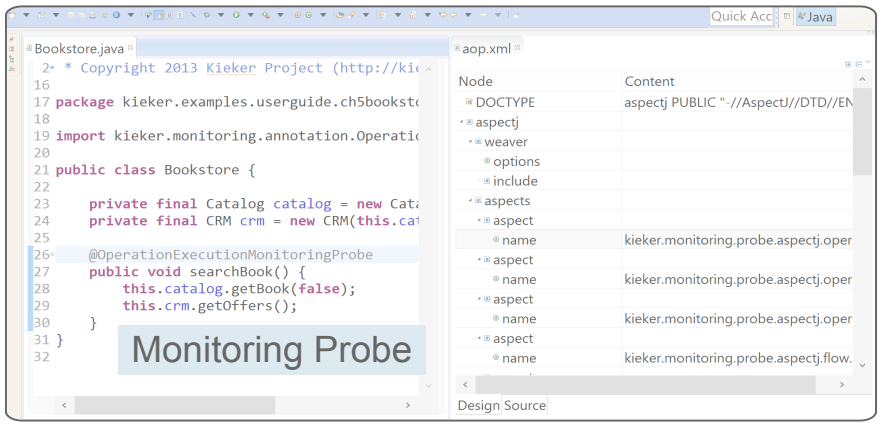
Analysis Configuration (via API and WebGUI)



Kieker: Dynamic Analysis Workflow

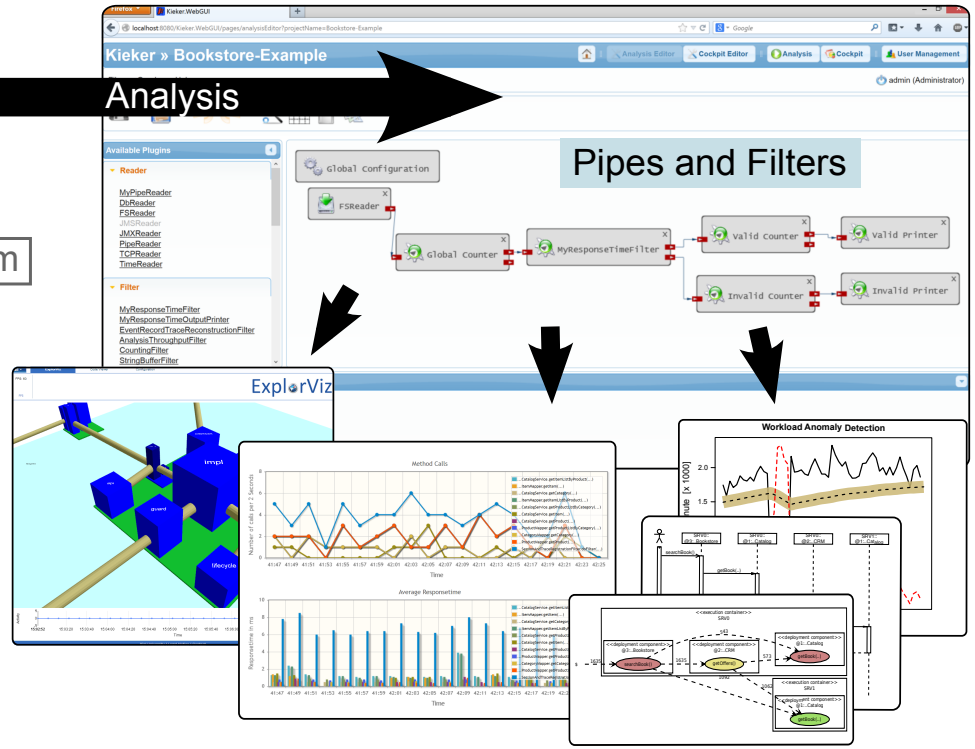
Introduction and Overview of Approach





Software System with Monitoring Instrumentation

Analysis Configuration (via API and WebGUI)



Online and Offline Visualization

Agenda

Kieker is distributed as part of SPEC® RG's repository of peer-reviewed tools for quantitative system evaluation and analysis

`http://research.spec.org/projects/tools.html`

Thanks to All Contributors

Various people contributed to Kieker in the past years.

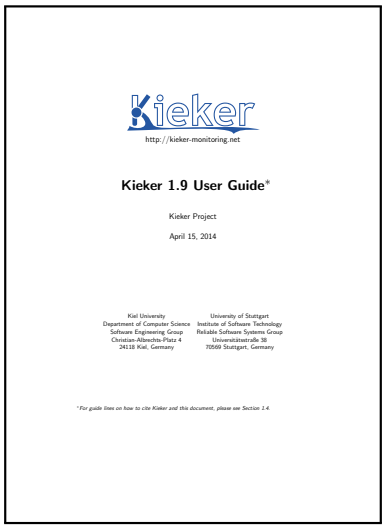
Tillmann (Till) Bielefeld, Peer Brauer, Philipp Döhring, Jens Ehlers, Nils Ehmke, Florian Fittkau, Thilo Focke, Sören Frey, Tom Frotscher, Henry Grow, Wilhelm (Willi) Hasselbring, André van Hoorn, Reiner Jung, Benjamin Kiel, Dennis Kieselhorst, Holger Knoche, Arnd Lange, Marius Löwe, Marco Lübcke, Felix 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

Agenda

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)


Kieker
http://kieker-monitoring.net

Kieker 1.9 User Guide*

Kieker Project

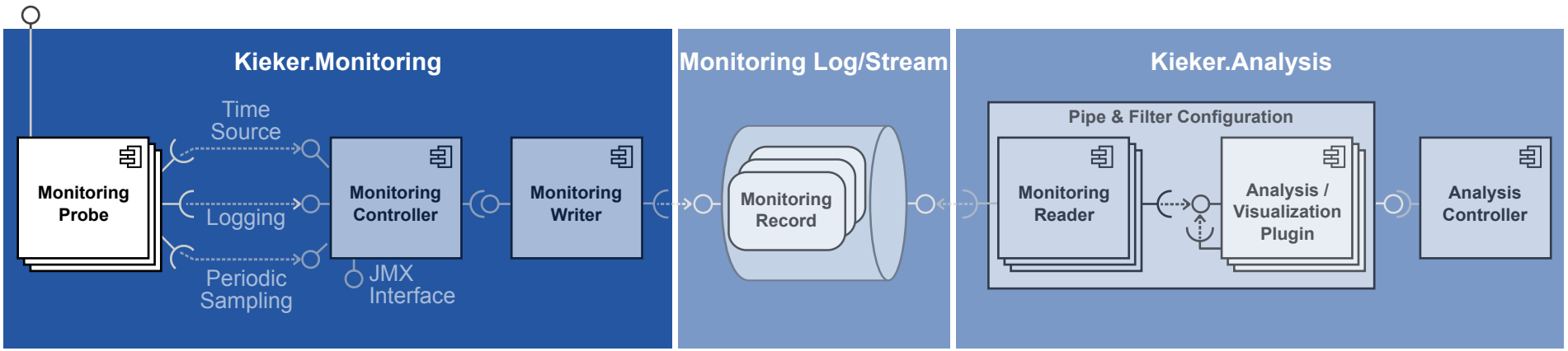
April 15, 2014

Kiel University
Department of Computer Science
Software Engineering Group
Christian-Albrechts-Platz 4
24118 Kiel, Germany

University of Stuttgart
Institute of Software Technology
Reliable Software Systems Group
Universitätsstraße 38
70569 Stuttgart, Germany

*For guide lines on how to cite Kieker and this document, please see Section 1.4.

Core Kieker Framework Components



Java probes/samplers:

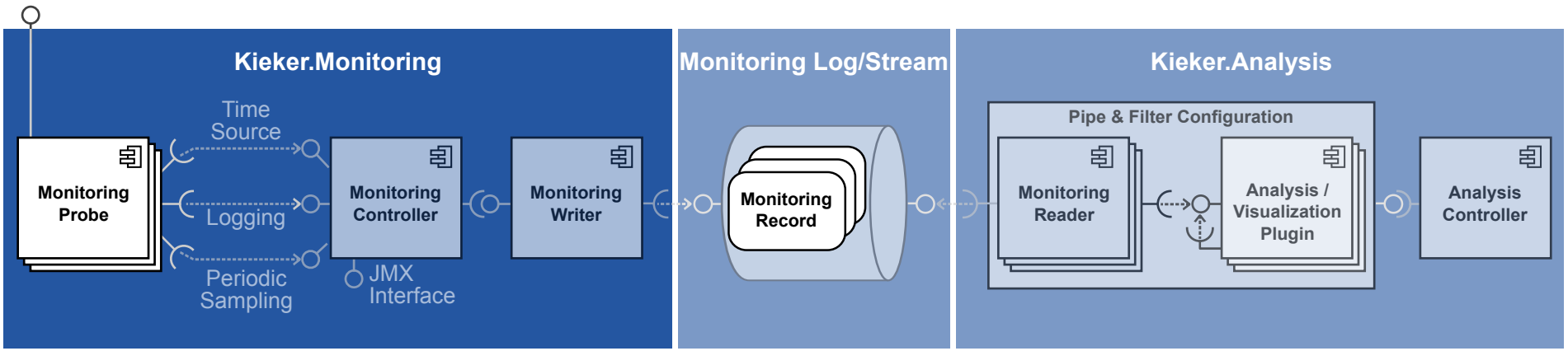
Monitoring Probes/Samplers

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

Core Kieker Framework Components



Java probes/samplers:

Monitoring Probes/Samplers

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>		

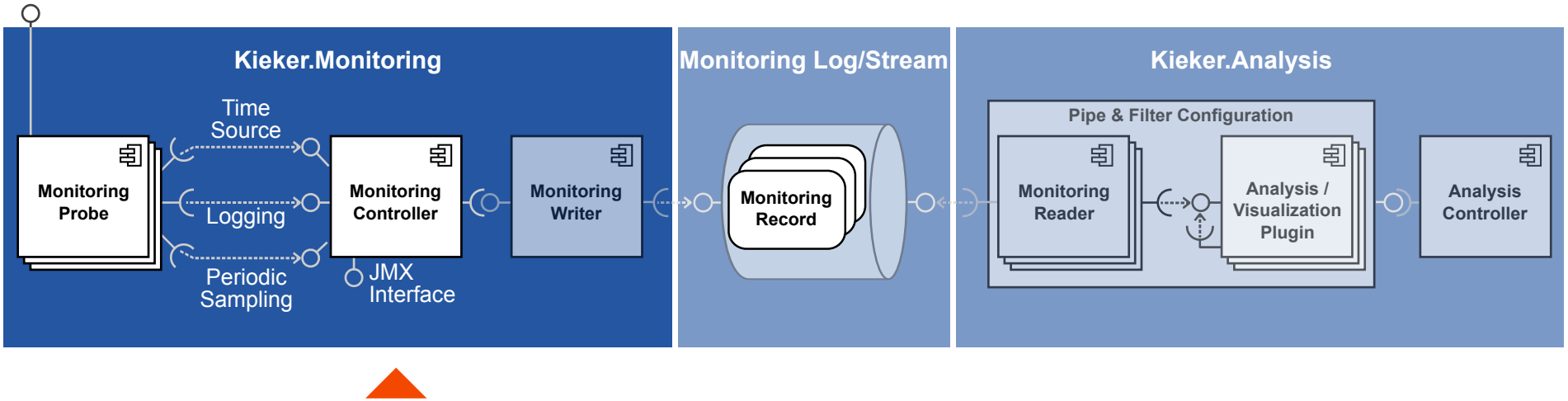
+ basic adapters for

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

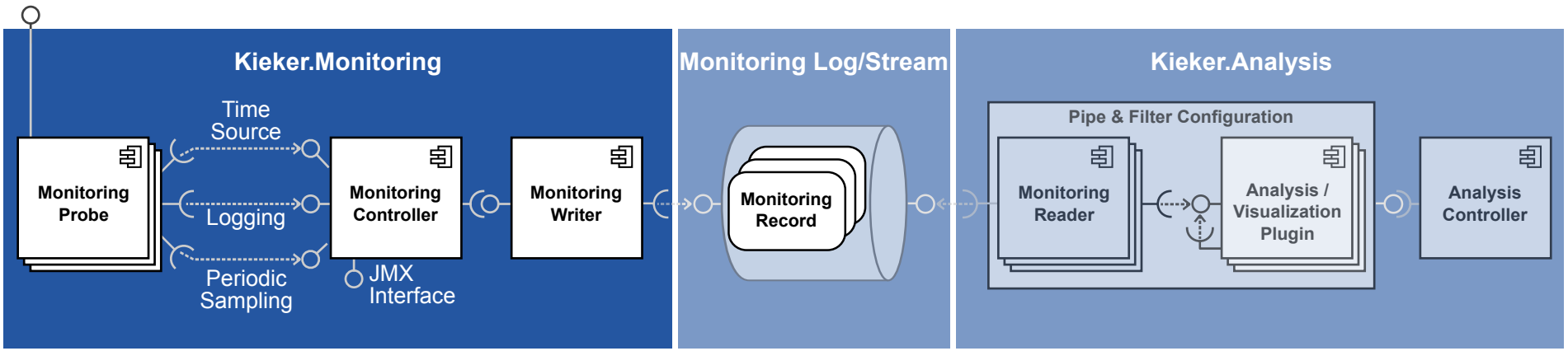
Monitoring Records

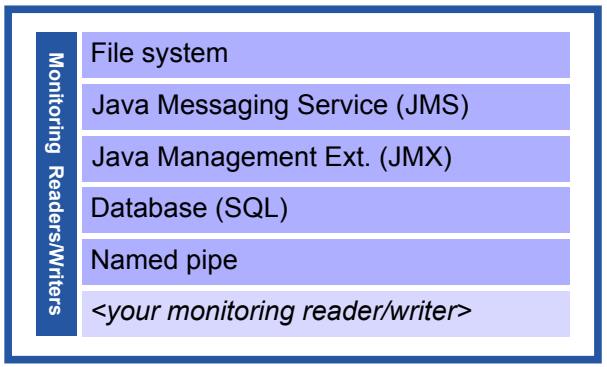
- Operation execution
- Control-flow events
- CPU utilization | Memory/swap usage
- Resource utilization
- Current time
- *<your monitoring record type>*

Core Kieker Framework Components

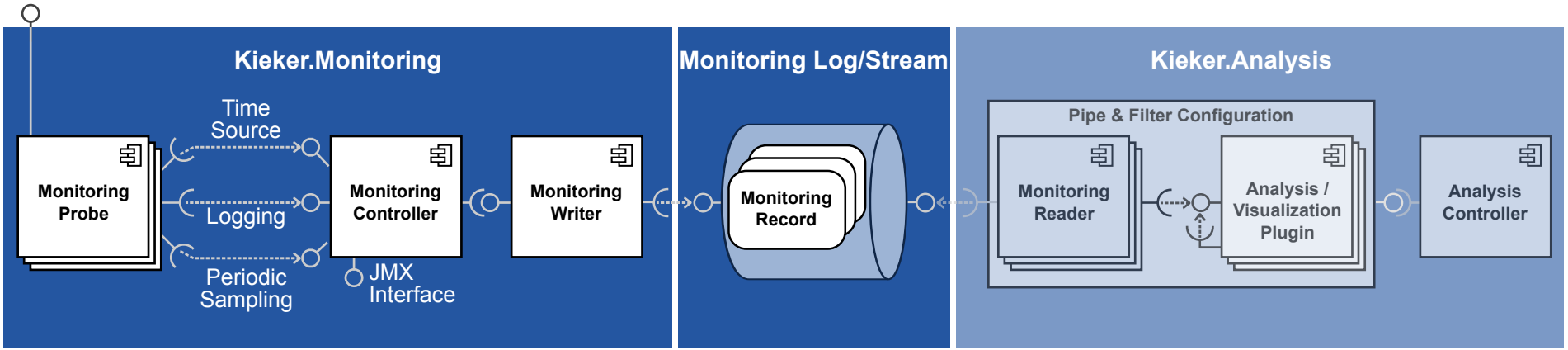


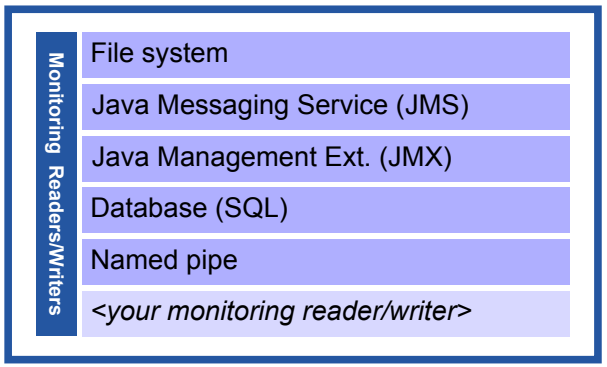
Core Kieker Framework Components



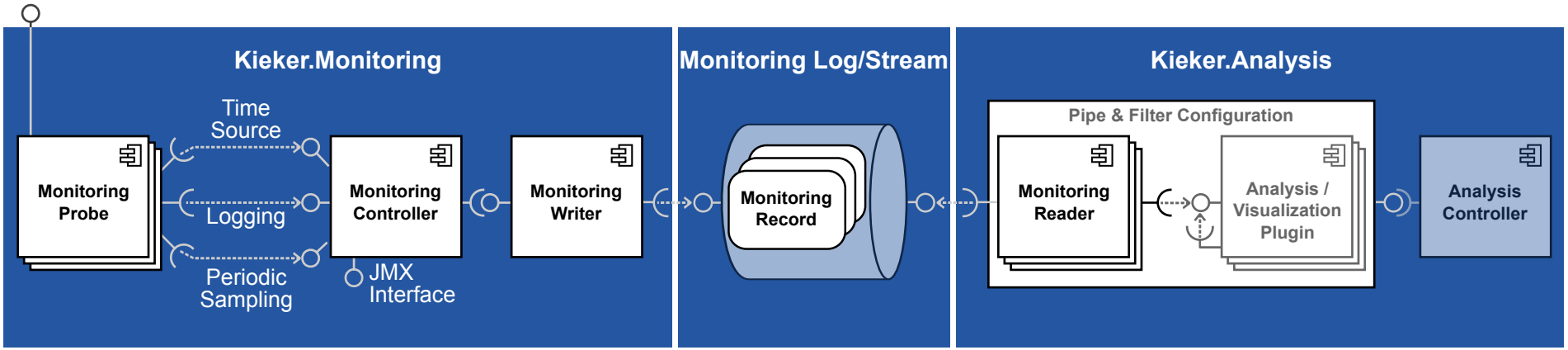


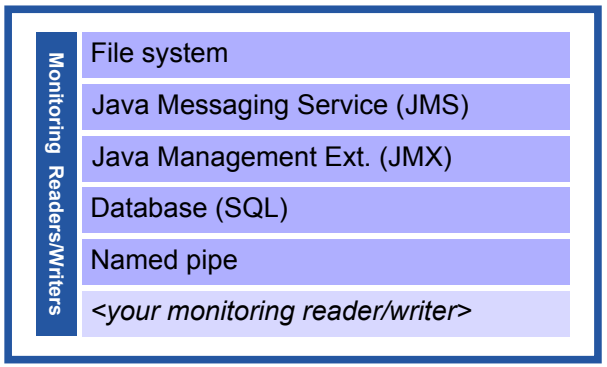
Core Kieker Framework Components



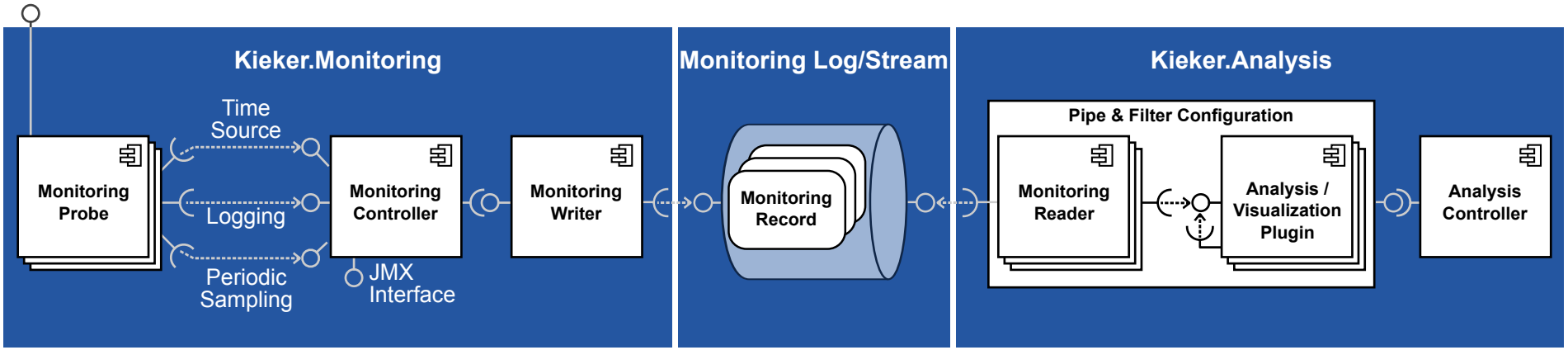


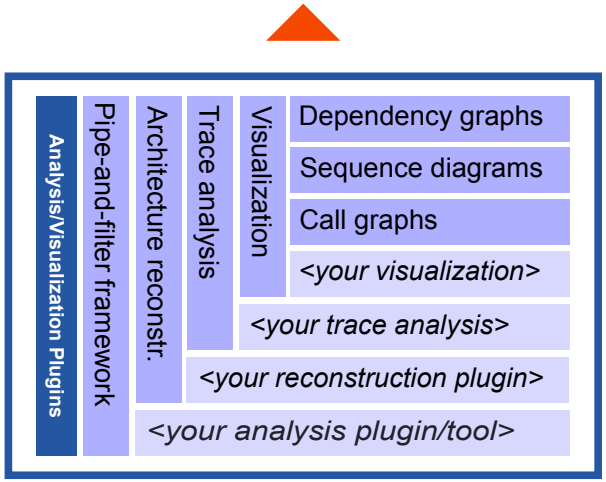
Core Kieker Framework Components





Core Kieker Framework Components





Agenda

Kieker Use Cases/Characteristics

1. Architecture Discovery (Dynamic/Hybrid Analysis)

2. Application Performance Management

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

Kieker Use Cases/Characteristics

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

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

Kieker Use Cases/Characteristics

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

Kieker Use Cases/Characteristics

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

Kieker Use Cases/Characteristics

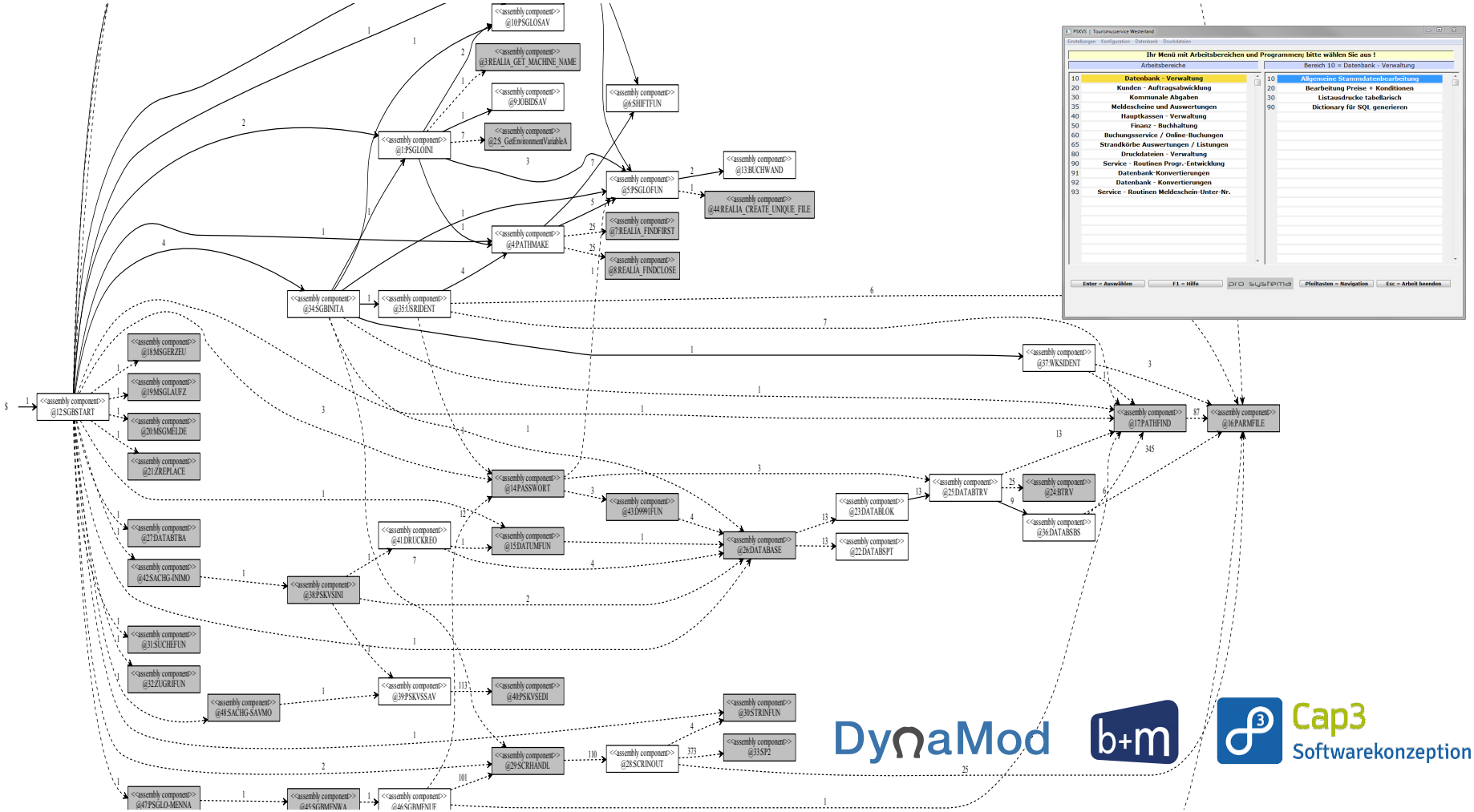
1. Architecture Discovery (Dynamic/Hybrid Analysis)
   - Extraction of architectural models (structure, behavior)
   - Reverse engineering of legacy systems
   - Software visualization (2D/3D, static/interactive)
   - Trace-based architecture analysis
2. Application Performance Management
   - Continuous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

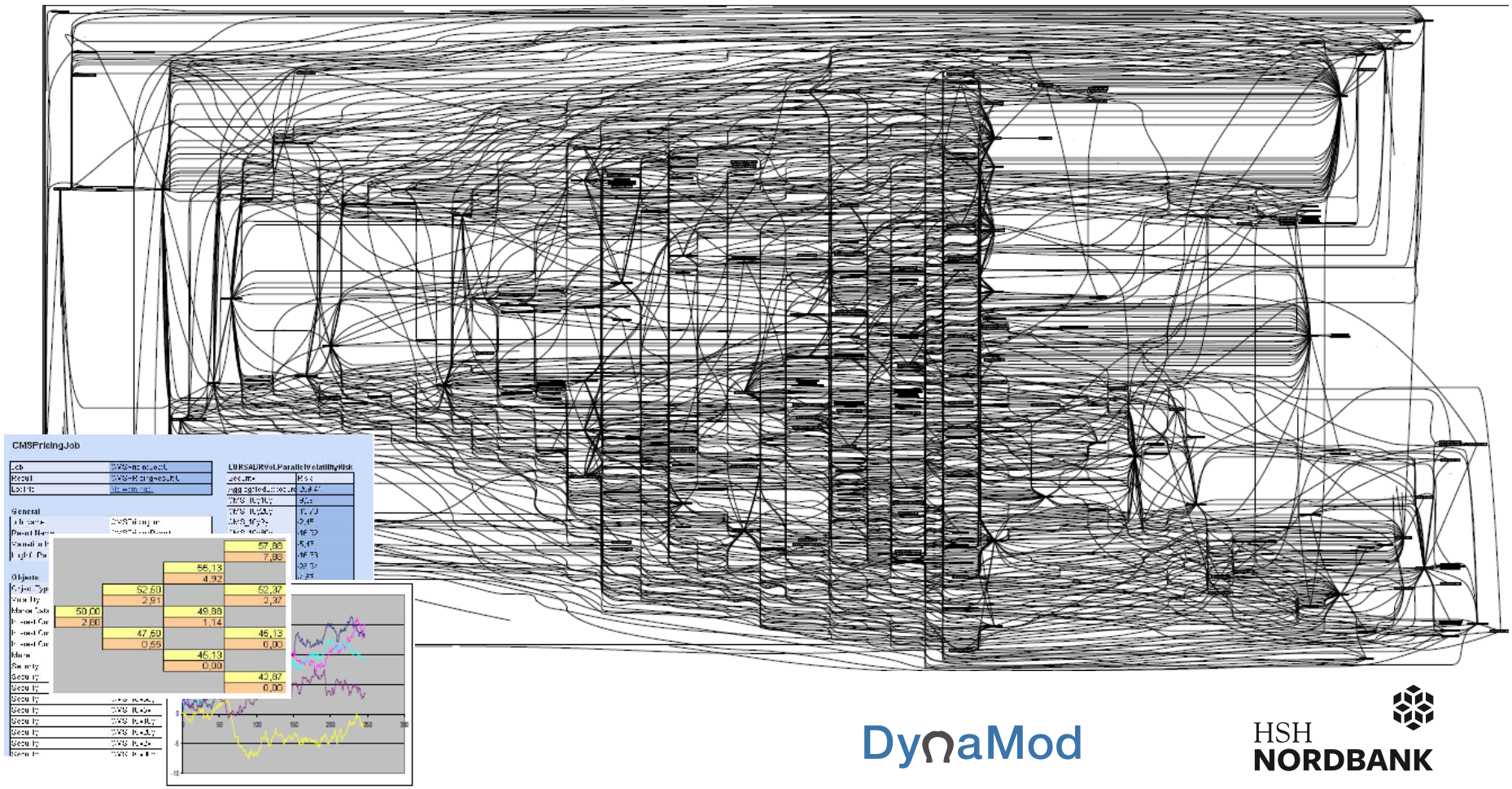
1. Architecture Discovery (Dynamic/Hybrid Analysis)
   - Extraction of architectural models (structure, behavior)
   - Reverse engineering of legacy systems
   - Software visualization (2D/3D, static/interactive)
   - Trace-based architecture analysis
2. Application Performance Management
   - Continuous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

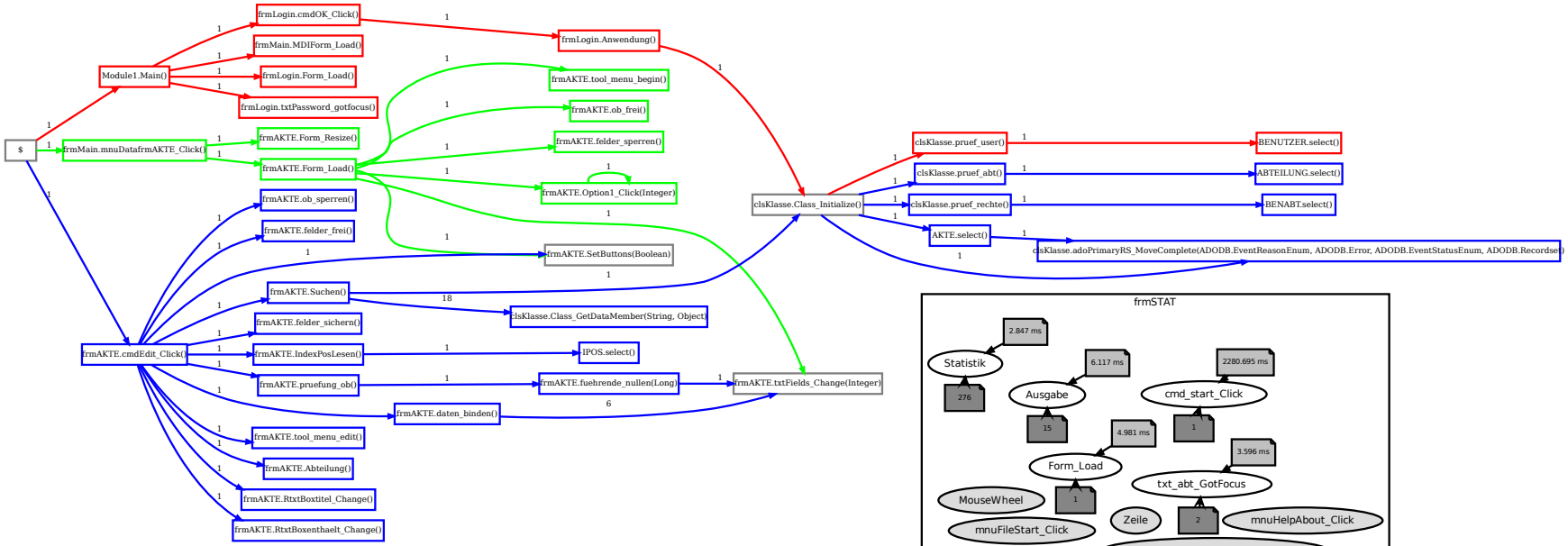
Architecture Discovery: Model Extraction + Visualization (cont'd)



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

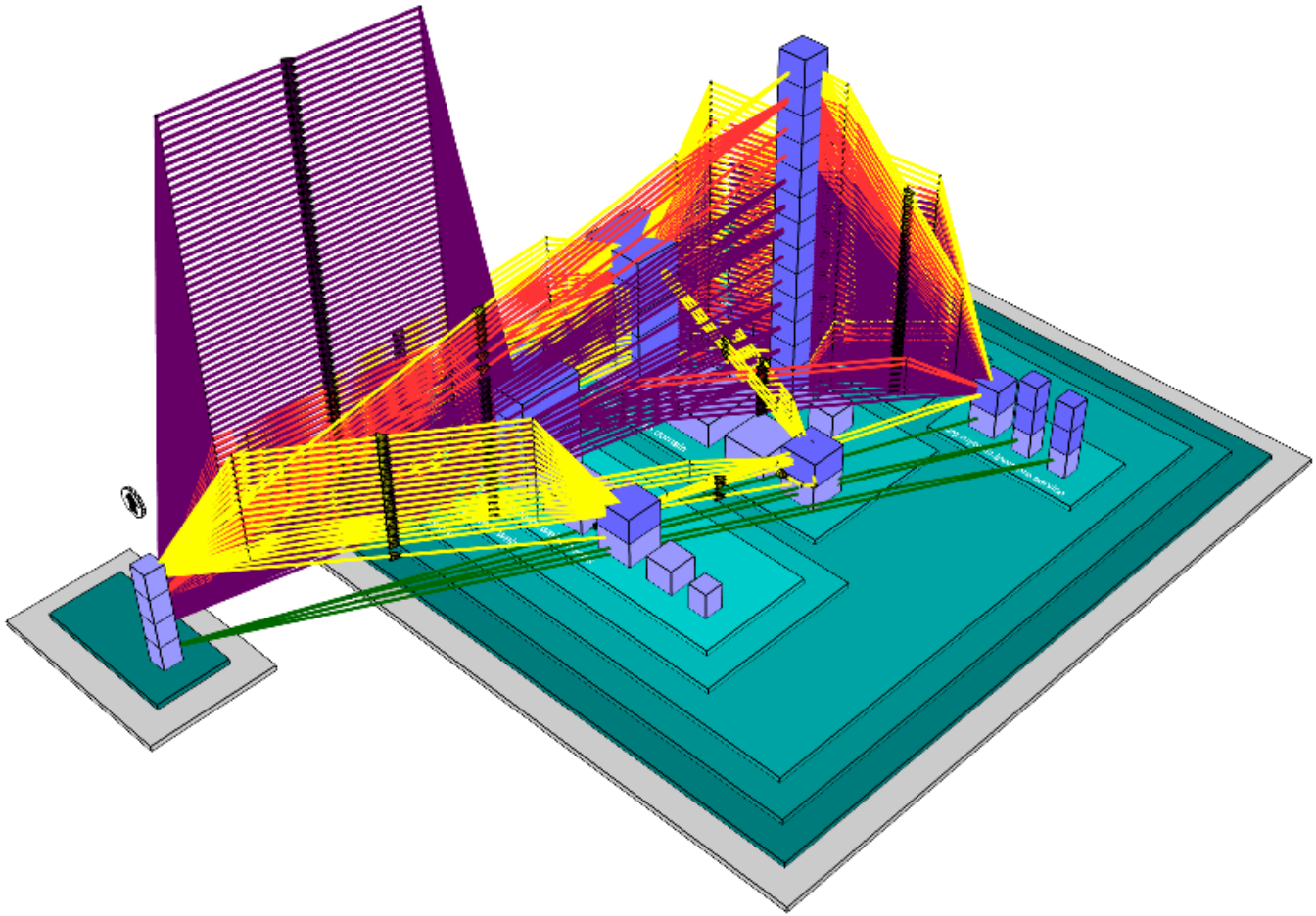


Architecture Discovery: Model Extraction + Visualization (cont'd)

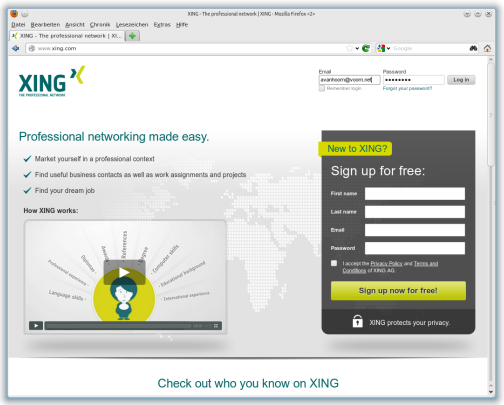


DynaMod b+m dataport

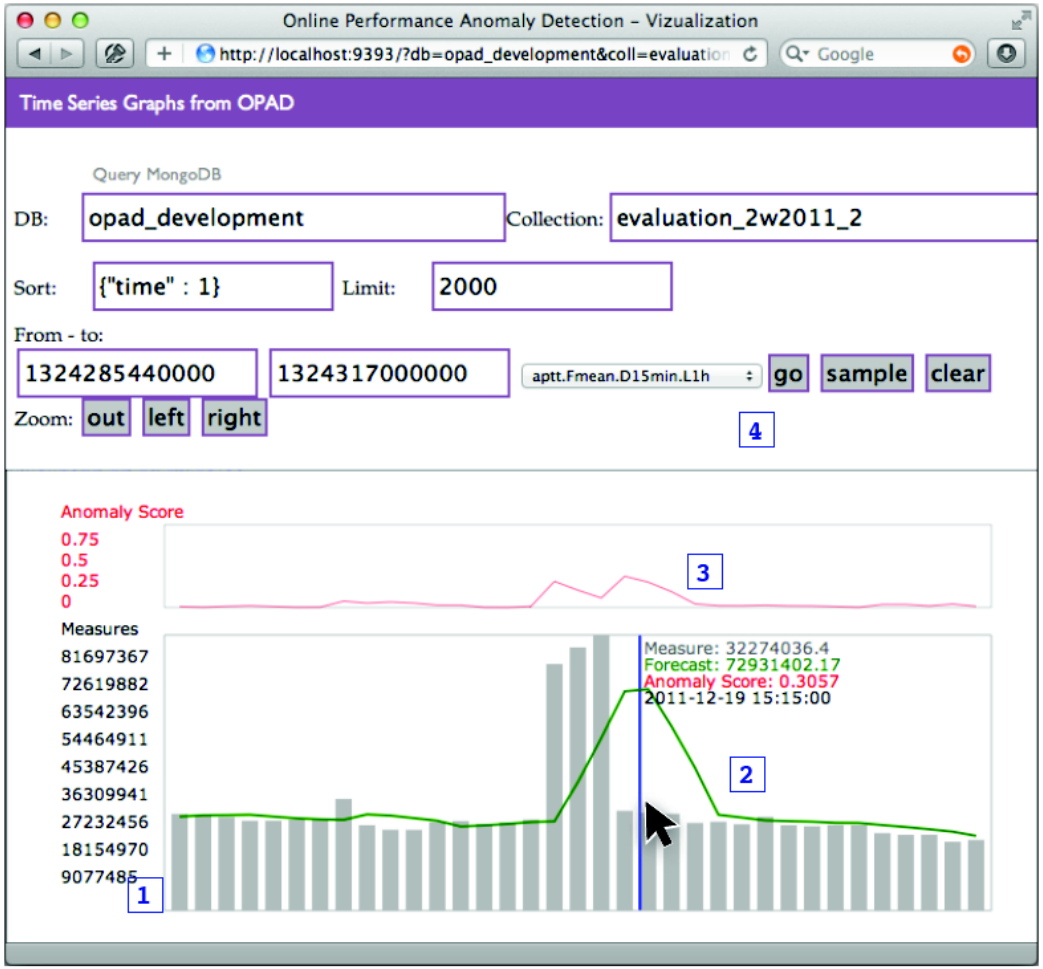
Architecture Discovery: Model Extraction + Visualization (cont'd)



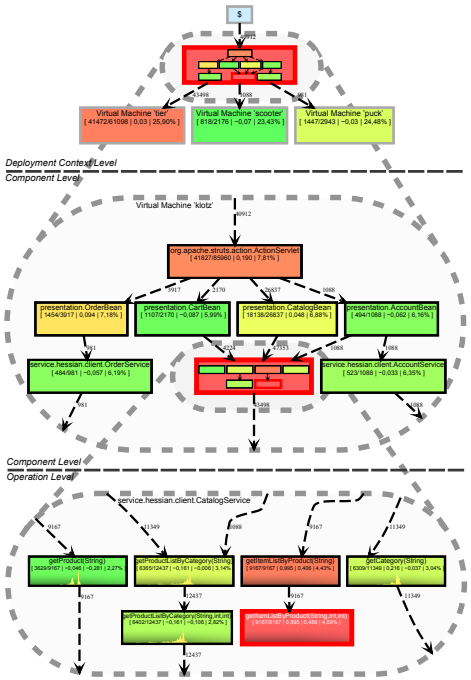
APM: Anomaly Detection + Diagnosis

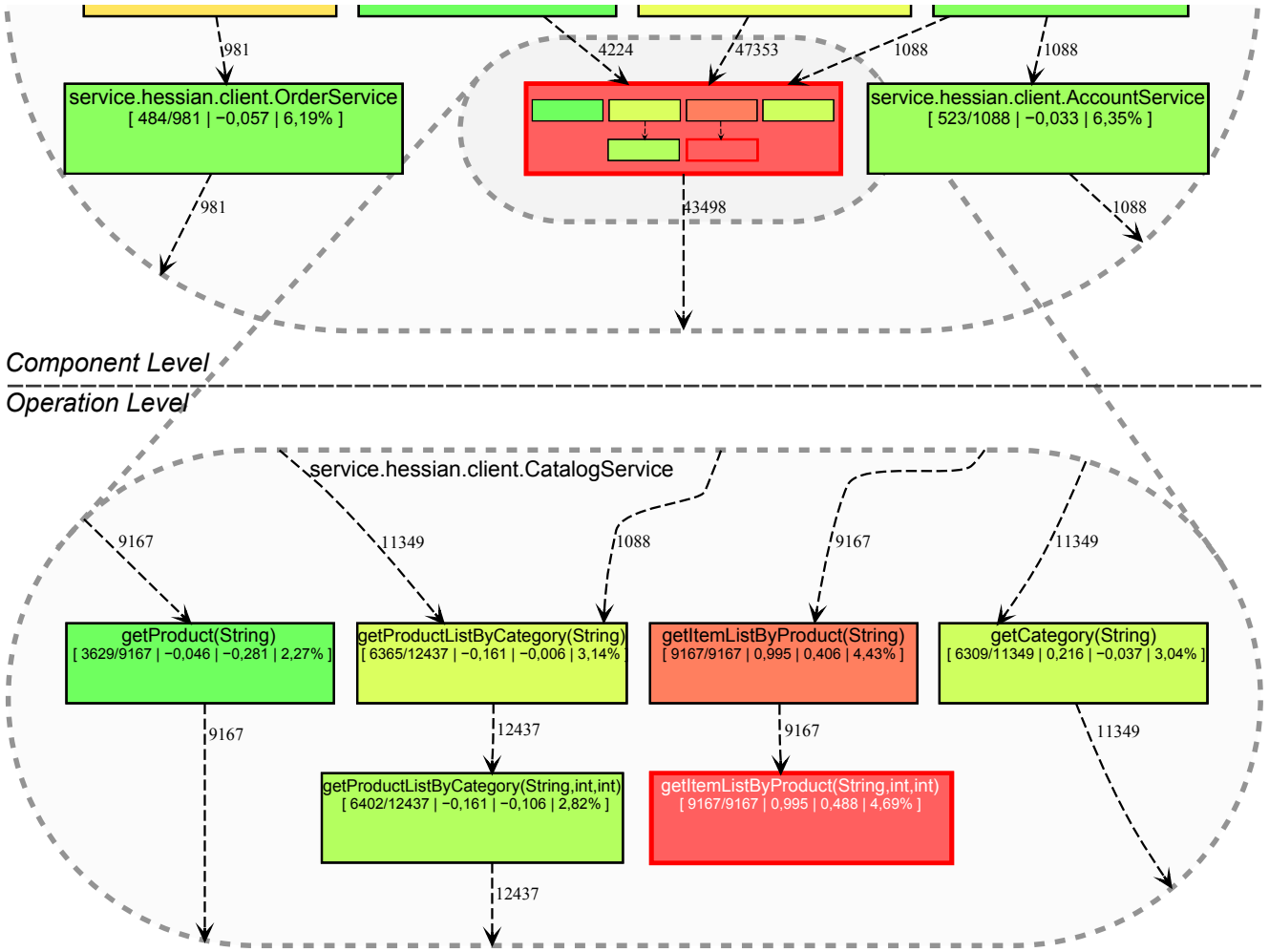


XING

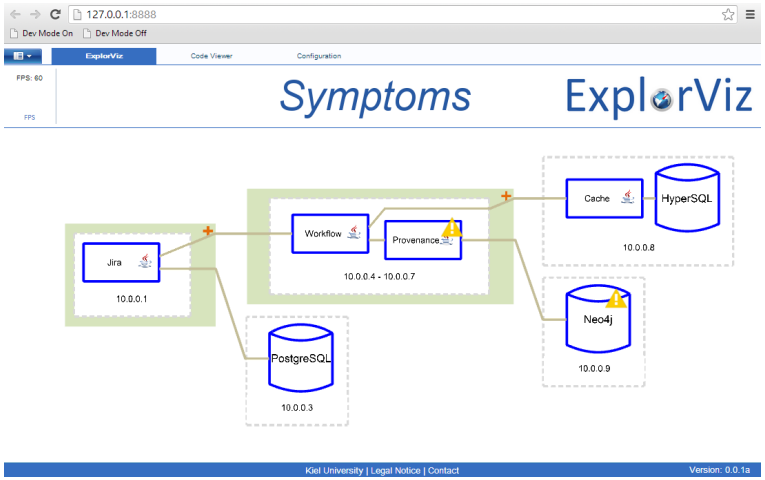


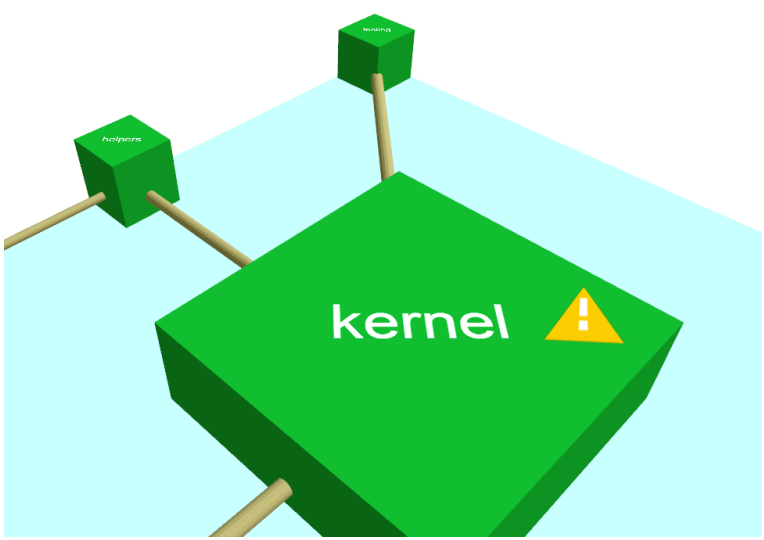
APM: Anomaly Detection + Diagnosis (cont'd)

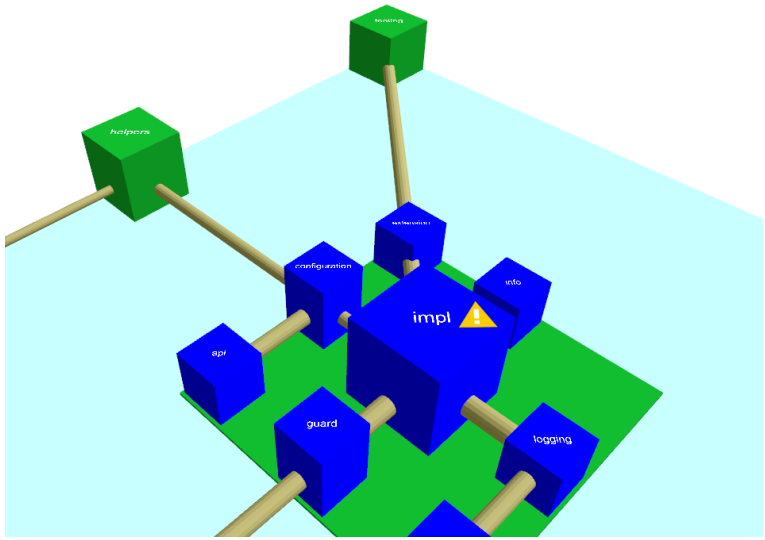




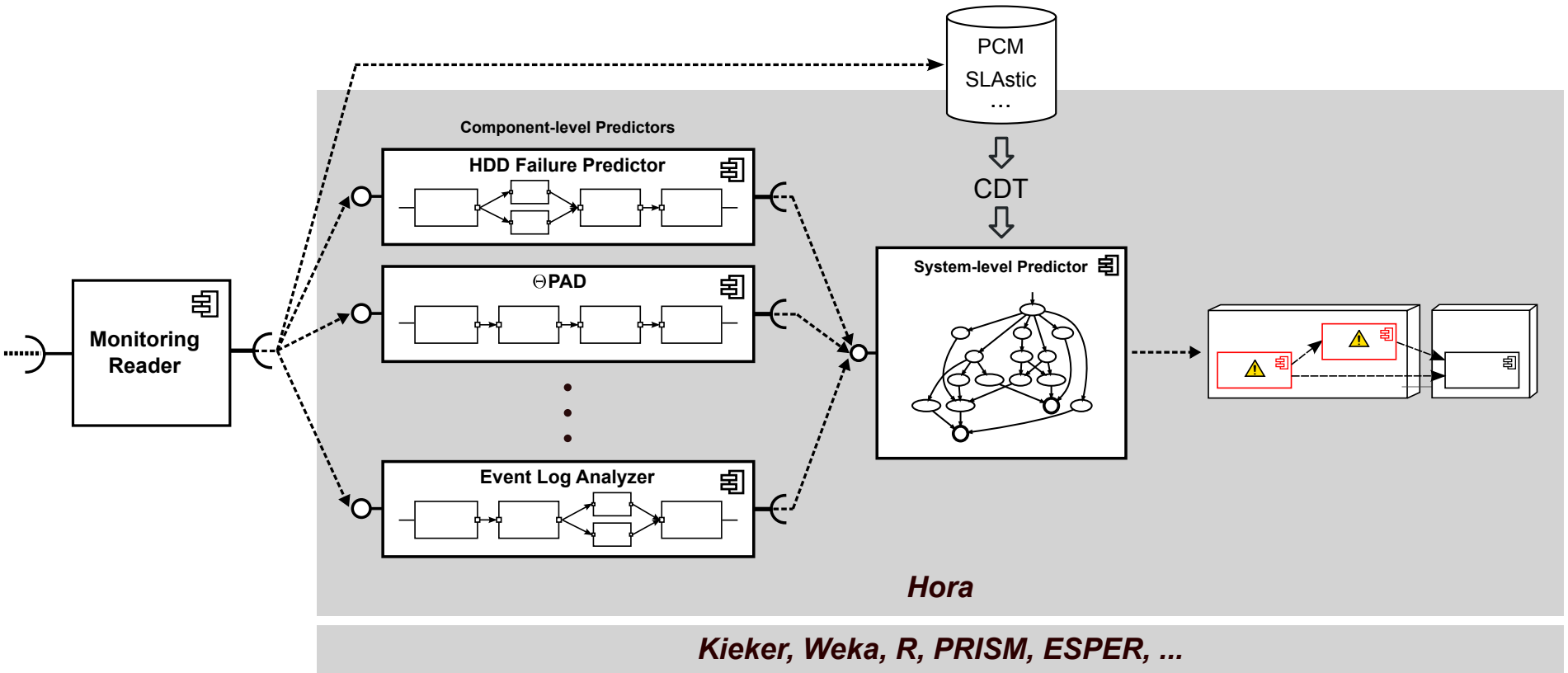
APM: Anomaly Detection + Diagnosis (cont'd)







APM: Anomaly Detection + Diagnosis (cont'd)

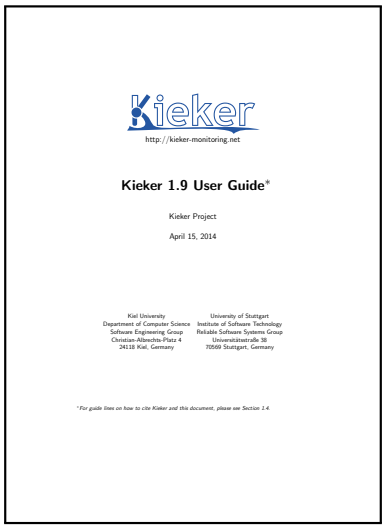


Agenda

Details on the Monitoring Component

Also refer to the Kieker User Guide

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


Kieker
http://kieker-monitoring.net

Kieker 1.9 User Guide*

Kieker Project

April 15, 2014

Kiel University
Department of Computer Science
Software Engineering Group
Christian-Albrechts-Platz 4
24118 Kiel, Germany

University of Stuttgart
Institute of Software Technology
Reliable Software Systems Group
Universitätsstraße 38
70569 Stuttgart, Germany

*For guide lines on how to cite Kieker and this document, please see Section 1.4.

Application code:

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

Monitoring probe code (schematic):

Application code:

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

Monitoring probe code (schematic):

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

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();
}
```

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, ...

Kieker Probe Example


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

Kieker Probe Example

```
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);
```

Kieker Probe Example


```
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);
```

Kieker Probe Example

```
private static final IMonitoringController MONITORING_CONTROLLER =
                MonitoringController.getInstance();
```

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

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

MonitoringController API (Excerpt)

MonitoringController

Instantiation (static)

Writing

Adaptive Monitoring

Controller State

Time Source

Periodic Sampling

JMX

Registry

MonitoringController API (Excerpt)

MonitoringController

Instantiation (static)

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

Writing

Adaptive Monitoring

Controller State

Time Source

Periodic Sampling

JMX

Registry

MonitoringController API (Excerpt)

MonitoringController

Instantiation (static)

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

Writing

+ boolean: **newMonitoringRecord**(IMonitoringRecord)

Adaptive Monitoring

Controller State

Time Source

Periodic Sampling

JMX

Registry

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 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
+ TimeSource: **getTimeSource**()

Periodic Sampling

JMX

Registry

MonitoringController API (Excerpt)

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**()

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

Periodic Sampling

JMX

Registry

MonitoringController API (Excerpt)

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**()

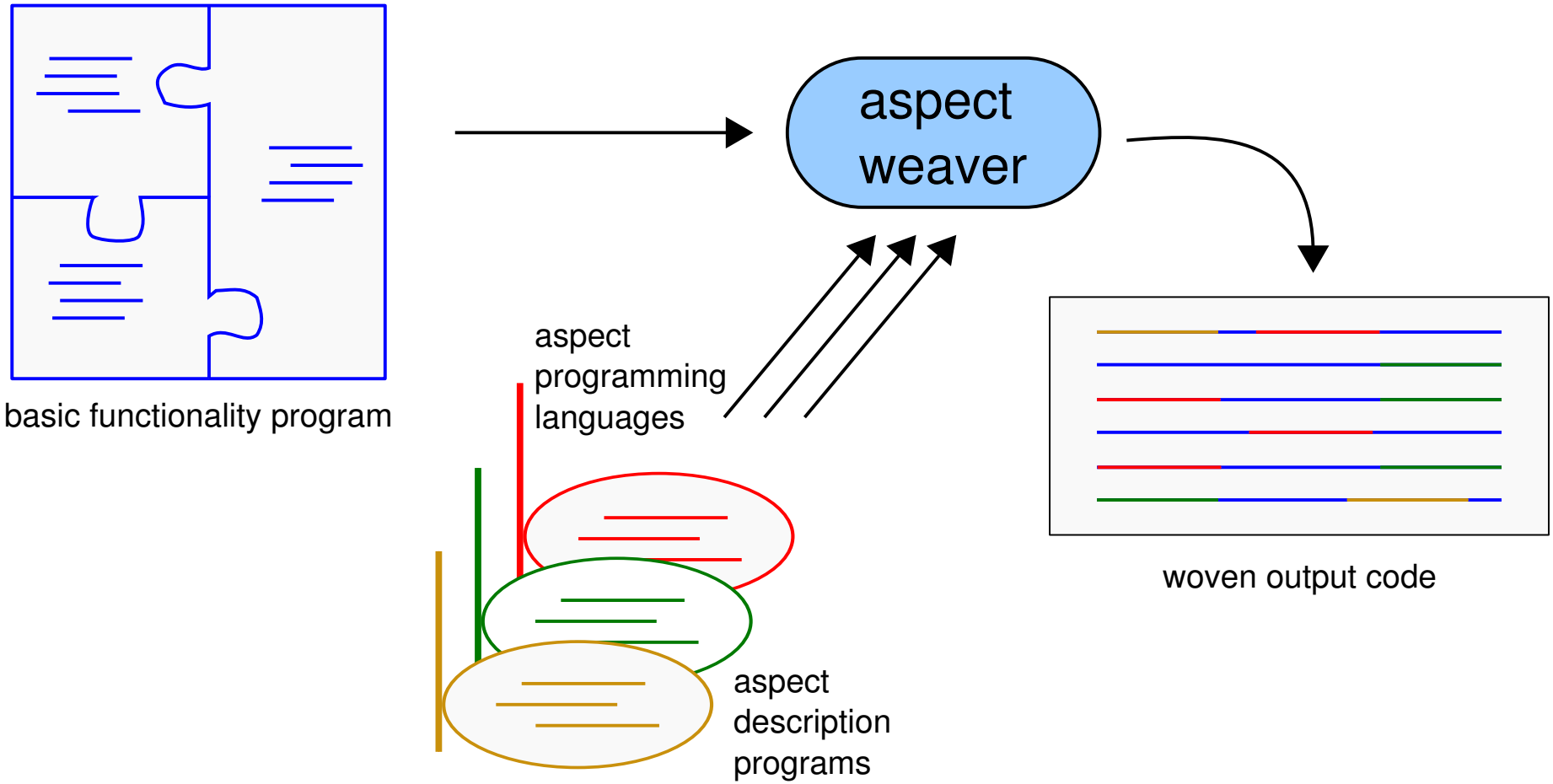
Time Source
+ TimeSource: **getTimeSource**()

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

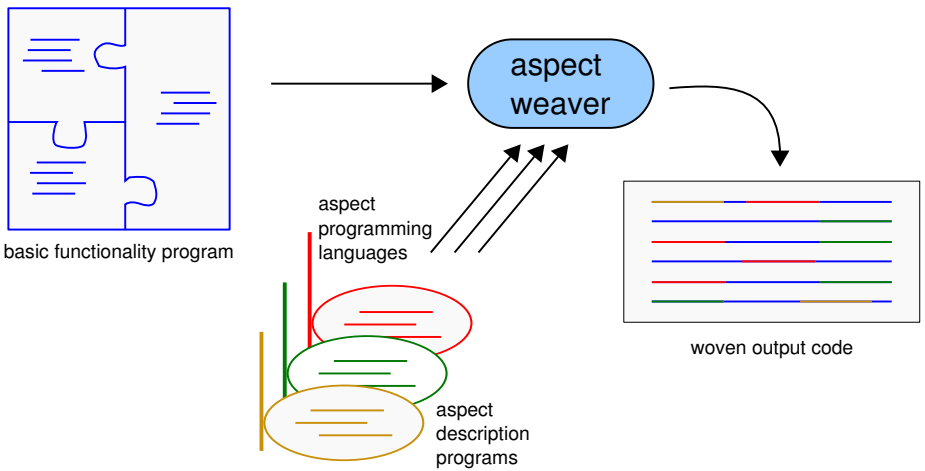
JMX

Registry

Aspect-Oriented Programming (AOP)



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




```
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

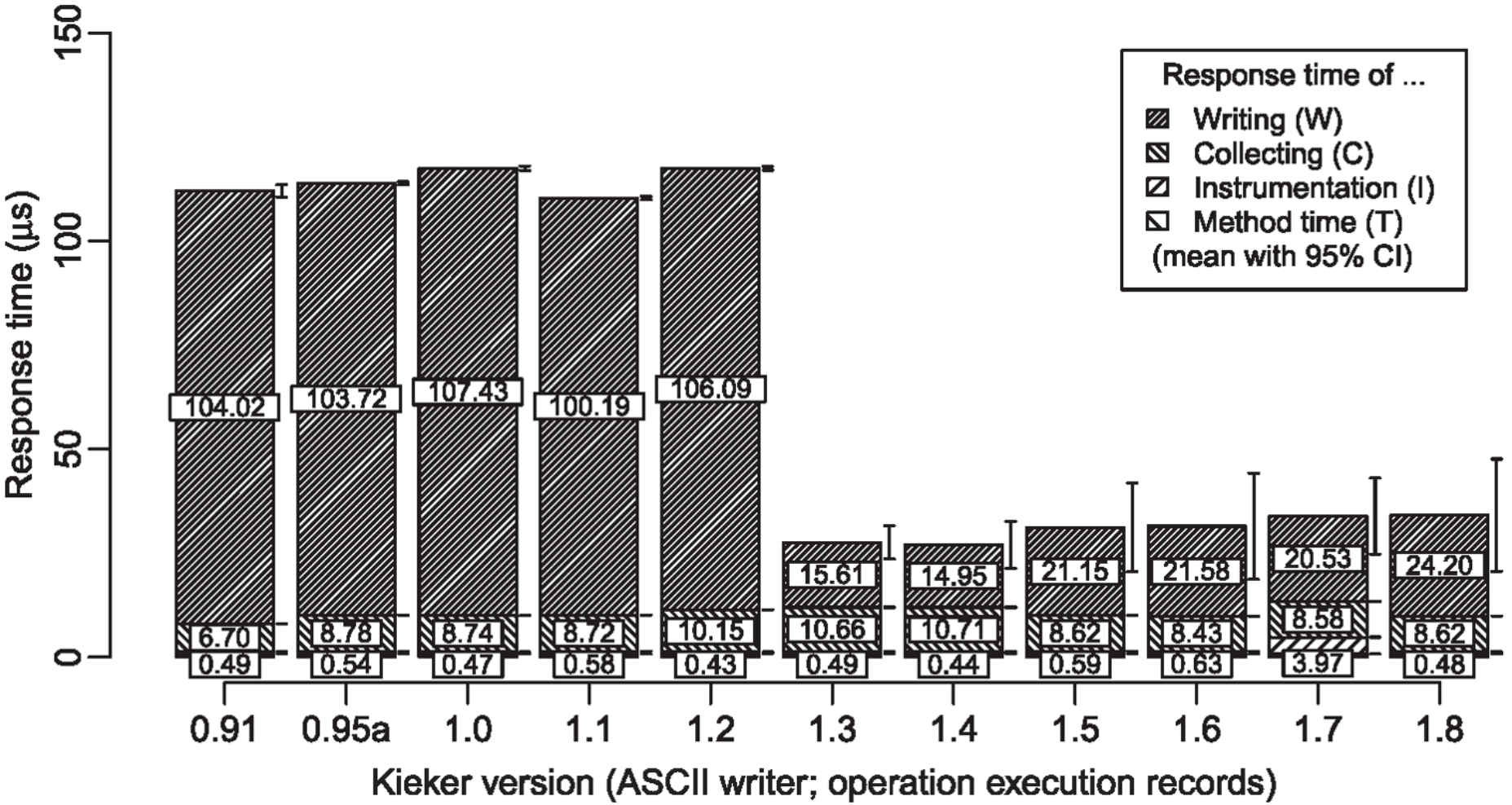
```
<!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN" "http://www.aspectj.org↩
    /dtd/aspectj_1_5_0.dtd">
<aspectj>
    <weaver options="">
        <include within="*"/>
    </weaver>
    <aspects>
      <aspect name="kieker.monitoring.probe.aspectj.operationExecution.↩
          OperationExecutionAspectFull"/>
    </aspects>
</aspectj>
```

Start the monitored application:

```
$ java -javaagent:lib/kieker-1.9_aspectj.jar BookstoreStarter
```

Monitoring Overhead

[Waller and Hasselbring 2013]

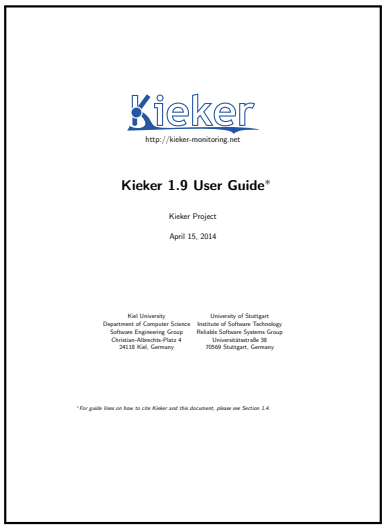


Agenda

Details on the Analysis Component

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)


Kieker
http://kieker-monitoring.net

Kieker 1.9 User Guide*

Kieker Project

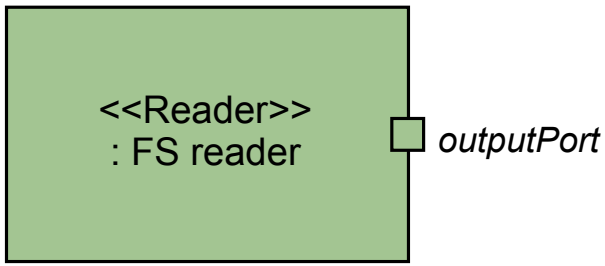
April 15, 2014

Kiel University
Department of Computer Science
Software Engineering Group
Christian-Albrechts-Platz 4
24118 Kiel, Germany

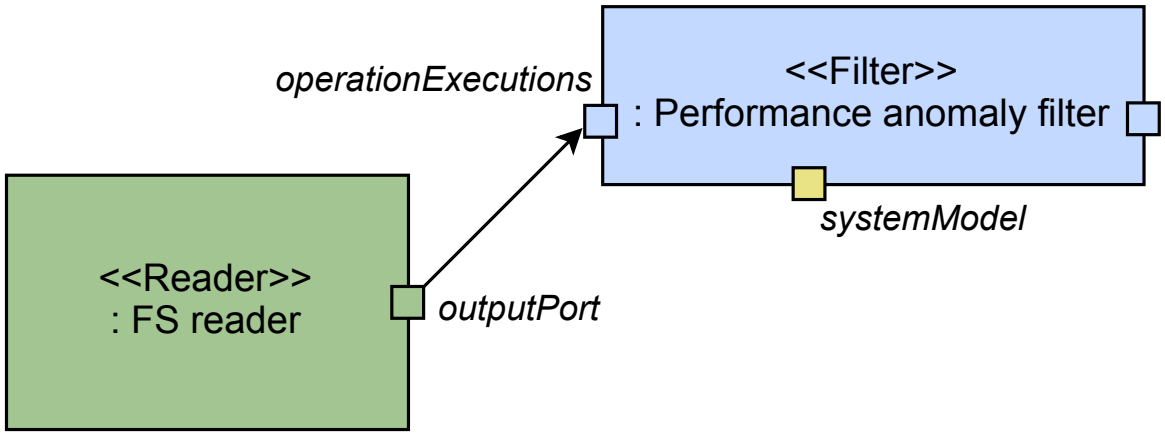
University of Stuttgart
Institute of Software Technology
Reliable Software Systems Group
Universitätsstraße 38
70569 Stuttgart, Germany

*For guide lines on how to cite Kieker and this document, please see Section 1.4.

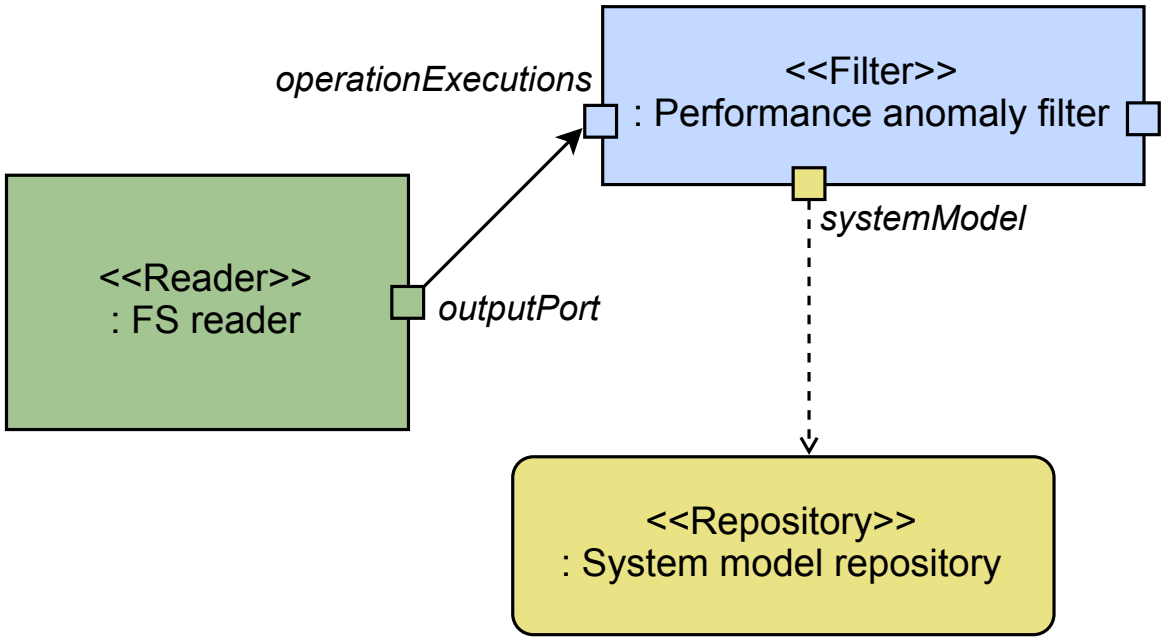
Example Pipe-and-Filter Configuration



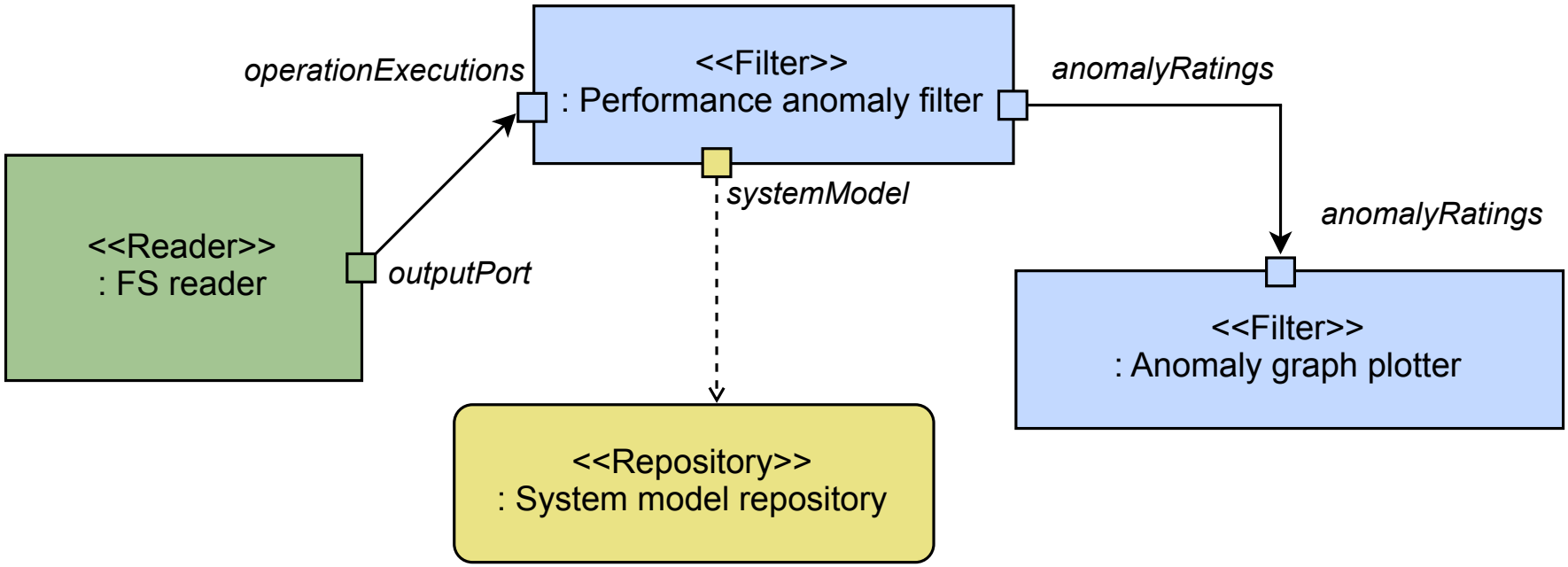
Example Pipe-and-Filter Configuration



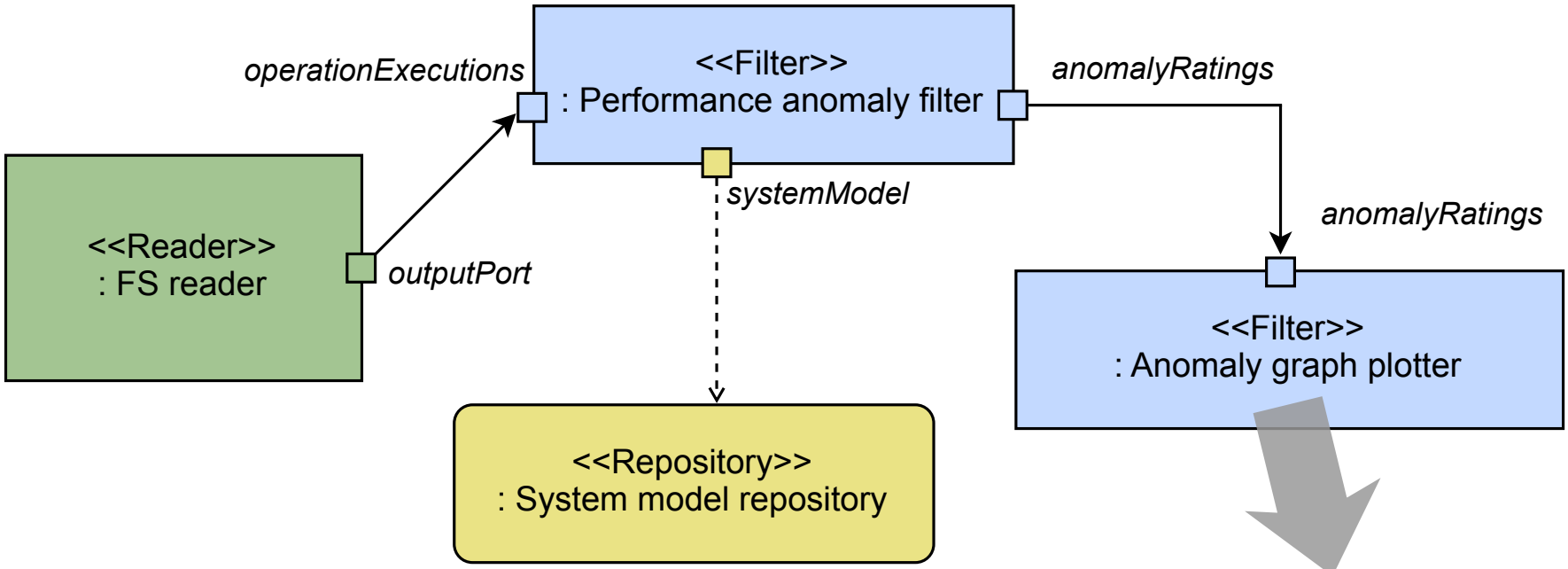
Example Pipe-and-Filter Configuration

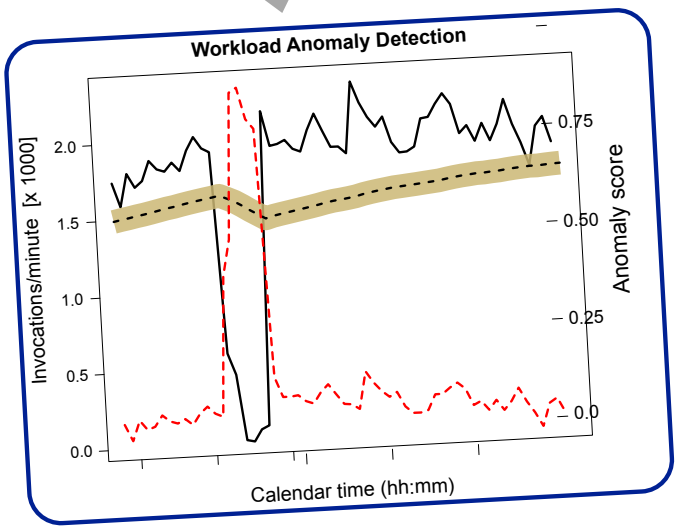


Example Pipe-and-Filter Configuration



Example Pipe-and-Filter Configuration





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

Programmatic Analysis Creation

```
/* 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"));
```

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

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

Own Analysis Components

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

Own Analysis Components (cont'd)

```
@Plugin(outputPorts = {
  @OutputPort(name = "eventCount", eventTypes = { Long.class })})
public final class CountingFilter extends AbstractFilterPlugin {
}
```

Own Analysis Components (cont'd)

```
@Plugin(outputPorts = {
  @OutputPort(name = "eventCount", eventTypes = { Long.class })})
public final class CountingFilter extends AbstractFilterPlugin {

 private final AtomicLong counter = new AtomicLong();

}
```

Own Analysis Components (cont'd)

```
@Plugin(outputPorts = {
  @OutputPort(name = "eventCount", eventTypes = { Long.class })})
public final class CountingFilter extends AbstractFilterPlugin {

 private final AtomicLong counter = new AtomicLong();

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

}
```

Own Analysis Components (cont'd)

```
@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)

```
@Plugin(outputPorts = {
  @OutputPort(name = "eventCount", eventTypes = { Long.class })})
public final class CountingFilter extends AbstractFilterPlugin {

 ...

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

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

}
```

AnalysisController API (Excerpt)

AnalysisController

Instantiation:

Persistence:

Controller State:

Pipes-and-Filters Configuration:

AnalysisController API (Excerpt)

AnalysisController

Instantiation:

+ **AnalysisController**()
+ **AnalysisController**(File)

Persistence:

Controller State:

Pipes-and-Filters Configuration:

AnalysisController API (Excerpt)

AnalysisController

Instantiation:

+ **AnalysisController**()
+ **AnalysisController**(File)

Persistence:

Controller State:

Pipes-and-Filters Configuration:

+ void: **connect**(AbstractPlugin, String, AbstractPlugin, String)
+ void: **connect**(AbstractPlugin, String, AbstractRepository)

AnalysisController API (Excerpt)

AnalysisController

Instantiation:

+ **AnalysisController**()
+ **AnalysisController**(File)

Persistence:

Controller State:

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

Pipes-and-Filters Configuration:

+ void: **connect**(AbstractPlugin, String, AbstractPlugin, String)
+ void: **connect**(AbstractPlugin, String, AbstractRepository)

AnalysisController API (Excerpt)

AnalysisController

Instantiation:

+ **AnalysisController**()
+ **AnalysisController**(File)

Persistence:

+ void: **saveToFile**(File)

Controller State:

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

Pipes-and-Filters Configuration:

+ void: **connect**(AbstractPlugin, String, AbstractPlugin, String)
+ void: **connect**(AbstractPlugin, String, AbstractRepository)

WebGUI Live Demo

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/`

Everything in Sight: Kieker's WebGUI in Action
– Tutorial Paper –

Nils Christian Ehmke
Department of Computer Science, Kiel University, Kiel, Germany
nie@informatik.uni-kiel.de

Abstract: In order to process monitoring data, the Kieker framework provides an API to define and execute pipe-and-filter oriented analysis networks. We develop a multi-user web application, which uses this API to allow graphical assembling and execution of Kieker analysis projects. Besides supporting programmers with the configuration of pipe-and-filter networks, the application helps to assemble so-called cockpits. Cockpits visualize results from running analyses and provide a live view on software systems within the web application using widgets. Such cockpits can be a useful tool for administrators to observe a system.

In this paper, we outline Kieker's pipe-and-filter API and introduce the web application. We put a main focus on its features, current development state, and planned future. Using an exemplary analysis, we show the tool and its potentials in action.

1 Introduction

Kieker[1] [vHWH12] is a Java-based framework which provides functionality to monitor and analyze, amongst others, Java, .NET, and COM applications. Various interception technologies, such as AspectJ[2] and Spring[3] AOP, can be used to weave monitoring code into the software. Pipe-and-filter analysis networks can be created, configured, and executed using a Java API.

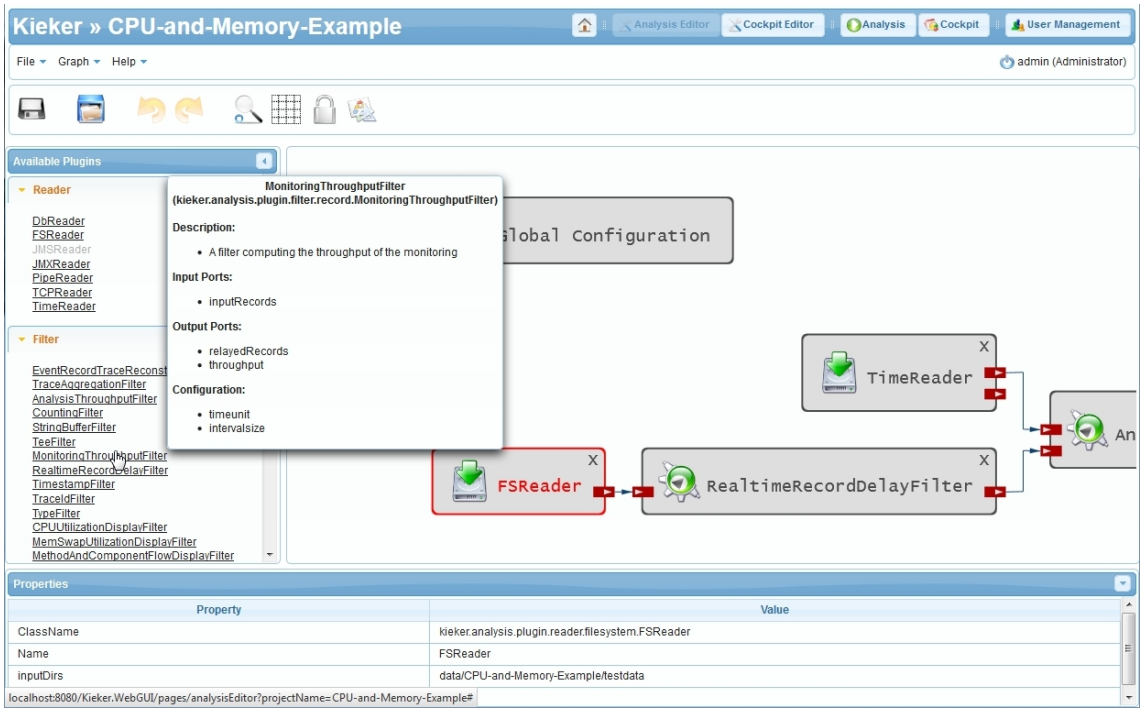
Larger analysis networks, consisting of tens or even hundreds of plugins, can be difficult to handle though. Using the API to define and configure such networks is possible, but is usually cumbersome and error-prone. Especially maintenance tasks and later modification requests require a suitable graphical editor. Our approach provides a web application for this issue.

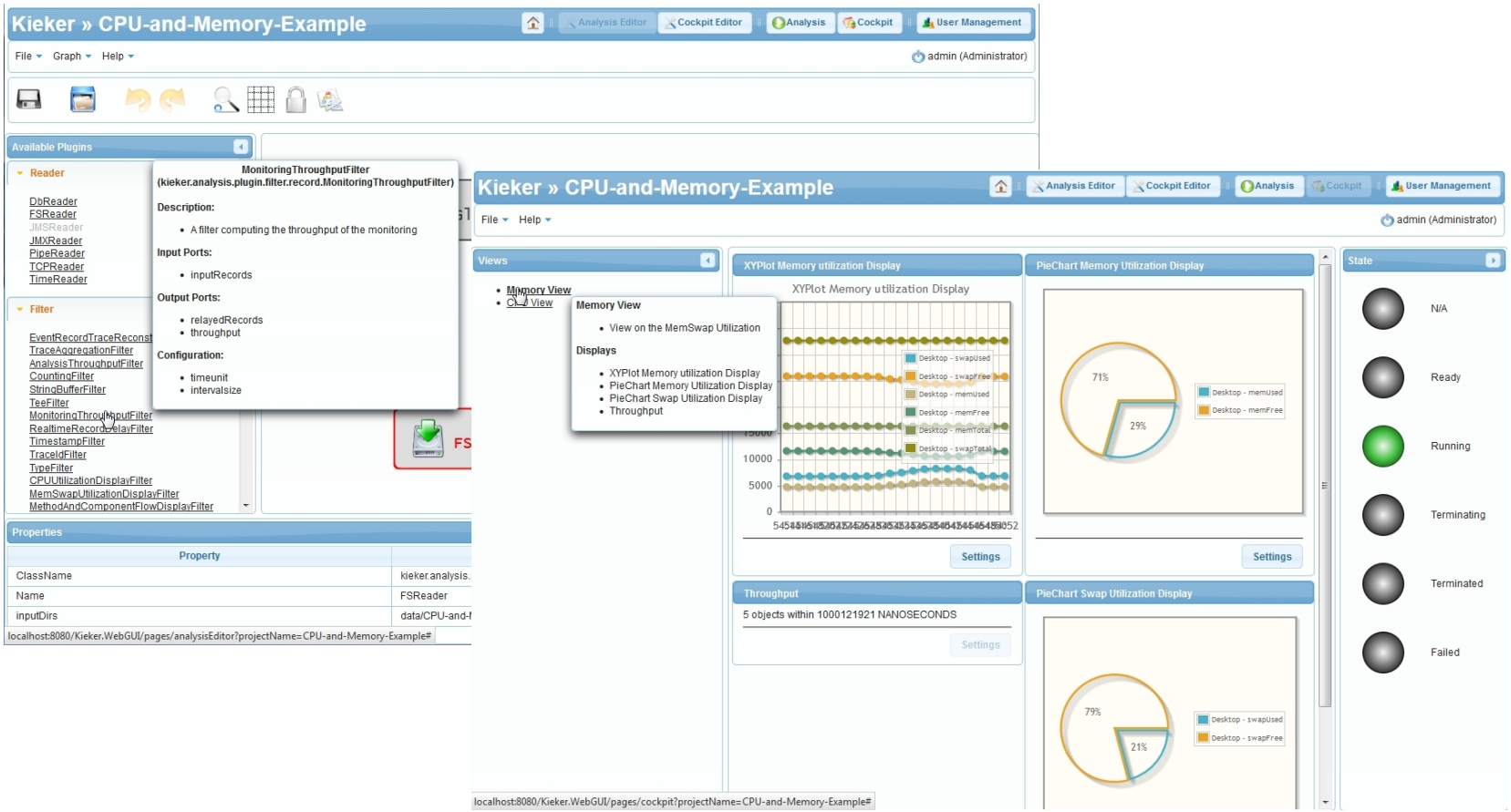
However, the assembly and configuration of analysis networks is only one part of the motivation for the web application. The other part is the live visualization of running analyses using widgets, which we designate as displays. Using the application, administrators can observe the behavior of software systems. In combination with anomaly detection [CBK09] filters, it is possible to detect and diagnose failures (e.g., long response times) within a system early.

[1] http://kieker-monitoring.net/
[2] http://eclipse.org/aspectj/
[3] http://spring.io/

Proc. Kieker/Palladio Days 2013, Nov. 27–29, Karlsruhe, Germany
Available online: http://ceur-ws.org/Vol-1083/

11

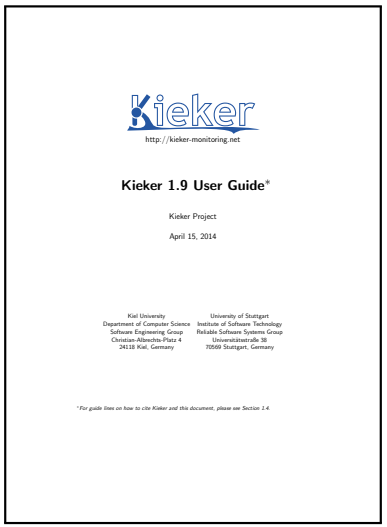

Kieker » CPU-and-Memory-Example
Analysis Editor
Cockpit Editor
Analysis
Cockpit
User Management
File
Graph
Help
admin (Administrator)
Available Plugins
Reader
DbReader
FSReader
JMSReader
JMXReader
PipeReader
TCPReader
TimeReader
Filter
EventRecordTraceReconst
TraceAggregationFilter
AnalysisThroughputFilter
CountingFilter
StringBufferFilter
TeeFilter
MonitoringThroughputFilter
RealtimeRecordDelayFilter
TimestampFilter
TraceIdFilter
TypeFilter
CPUUtilizationDisplayFilter
MemSwapUtilizationDisplayFilter
MethodAndComponentFlowDisplayFilter
MonitoringThroughputFilter
(kieker.analysis.plugin.filter.record.MonitoringThroughputFilter)
Description:
A filter computing the throughput of the monitoring
Input Ports:
inputRecords
Output Ports:
relayedRecords
throughput
Configuration:
timeunit
intervalsize
Global Configuration
TimeReader
FSReader
RealtimeRecordDelayFilter
Properties
Property
Value
ClassName
kieker.analysis.plugin.reader.filesystem.FSReader
Name
FSReader
inputDirs
data/CPU-and-Memory-Example/testdata
localhost:8080/Kieker.WebGUI/pages/analysisEditor?projectName=CPU-and-Memory-Example#


Kieker » CPU-and-Memory-Example
Analysis Editor
Cockpit Editor
Analysis
Cockpit
User Management
File
Graph
Help
admin (Administrator)
Available Plugins
Reader
DbReader
FSReader
JMSReader
JMXReader
PipeReader
TCPReader
TimeReader
Filter
EventRecordTraceReconst
TraceAggregationFilter
AnalysisThroughputFilter
CountingFilter
StringBufferFilter
TeeFilter
MonitoringThroughputFilter
RealtimeRecordDelayFilter
TimestampFilter
TraceIdFilter
TypeFilter
CPUUtilizationDisplayFilter
MemSwapUtilizationDisplayFilter
MethodAndComponentFlowDisplayFilter
MonitoringThroughputFilter
(kieker.analysis.plugin.filter.record.MonitoringThroughputFilter)
Description:
A filter computing the throughput of the monitoring
Input Ports:
inputRecords
Output Ports:
relayedRecords
throughput
Configuration:
timeunit
intervalsize
Properties
Property
ClassName
kieker.analysis.
Name
FSReader
inputDirs
data/CPU-and-
localhost:8080/Kieker.WebGUI/pages/analysisEditor?projectName=CPU-and-Memory-Example#
Kieker » CPU-and-Memory-Example
Analysis Editor
Cockpit Editor
Analysis
Cockpit
User Management
File
Help
admin (Administrator)
Views
Memory View
CPU View
Memory View
View on the MemSwap Utilization
Displays
XYPlot Memory utilization Display
PieChart Memory Utilization Display
PieChart Swap Utilization Display
Throughput
Throughput
XYPlot Memory utilization Display
Desktop - swapUsed
Desktop - swapFree
Desktop - memUsed
Desktop - memFree
Desktop - memTotal
Desktop - swapTotal
15000
10000
5000
0
Settings
PieChart Memory Utilization Display
71%
29%
Desktop - memUsed
Desktop - memFree
Settings
Throughput
5 objects within 1000121921 NANOSECONDS
Settings
PieChart Swap Utilization Display
79%
21%
Desktop - swapUsed
Desktop - swapFree
State
N/A
Ready
Running
Terminating
Terminated
Failed
localhost:8080/Kieker.WebGUI/pages/cockpit?projectName=CPU-and-Memory-Example#

Agenda

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)


Kieker
http://kieker-monitoring.net

Kieker 1.9 User Guide*

Kieker Project

April 15, 2014

Kiel University
Department of Computer Science
Software Engineering Group
Christian-Albrechts-Platz 4
24118 Kiel, Germany

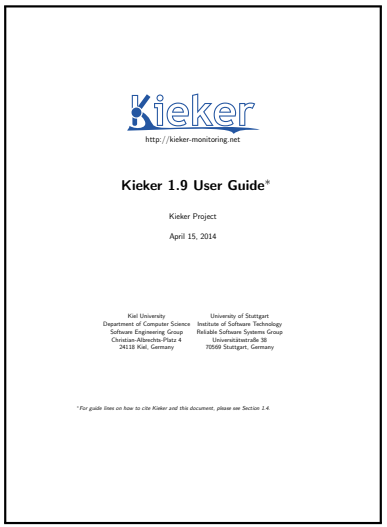
University of Stuttgart
Institute of Software Technology
Reliable Software Systems Group
Universitätsstraße 38
70569 Stuttgart, Germany

*For guide lines on how to cite Kieker and this document, please see Section 1.4.

Agenda

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]


Kieker
http://kieker-monitoring.net

Kieker 1.9 User Guide*

Kieker Project

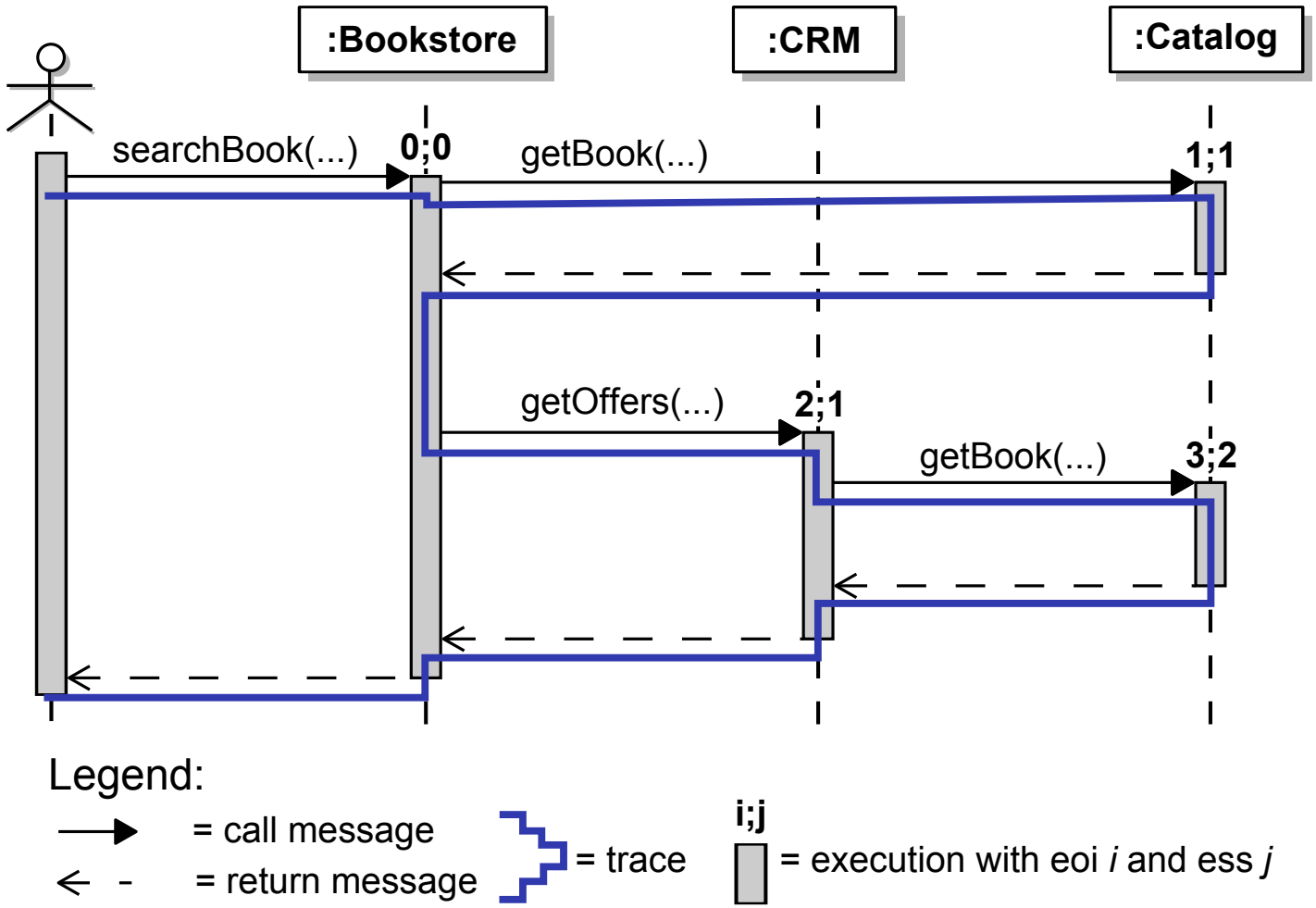
April 15, 2014

Kiel University
Department of Computer Science
Software Engineering Group
Christian-Albrechts-Platz 4
24118 Kiel, Germany

University of Stuttgart
Institute of Software Technology
Reliable Software Systems Group
Universitätsstraße 38
70569 Stuttgart, Germany

*For guide lines on how to cite Kieker and this document, please see Section 1.4.

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*

Anatomy of a Monitoring Record

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

```
$0; 128315654562336560 8; public void kieker.examples..CRM.getOffers();
<no-session-id>; 6488138950668976129; 1283156498817823953;
1283156498820007367; Osterinsel; 2; 1
```

The meaning of this record:

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`)

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)

Anatomy of a Monitoring Record

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

```
$0; 128315654562336560 8; 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)

Anatomy of a Monitoring Record

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

```
$0; 128315654562336560 8; 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; 128315654562336560 8; public void kieker.examples..CRM.getOffers();
<no-session-id>; 6488138950668976129; 128315649881782395 3;
128315649882000736 7; 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)

Anatomy of a Monitoring Record

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

```
$0; 128315654562336560 8; 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)

Anatomy of a Monitoring Record

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

```
$0; 128315654562336560 8; public void kieker.examples..CRM.getOffers();
<no-session-id>; 6488138950668976129; 128315649881782395 3;
128315649882000736 7; 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; 128315654562336560 8; public void kieker.examples..CRM.getOffers();
<no-session-id>; 6488138950668976129; 128315649881782395 3;
128315649882000736 7; 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)

Anatomy of a Monitoring Record

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

```
$0; 128315654562336560 8; 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)

Anatomy of a Monitoring Record

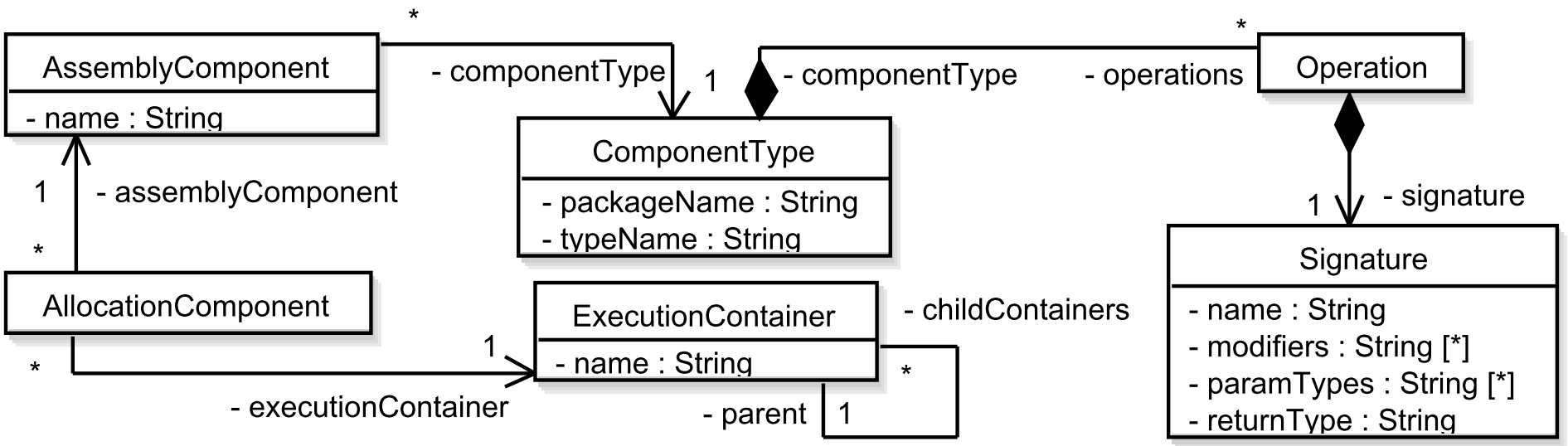
A single line of the monitoring log – a single *monitoring record*

```
$0; 128315654562336560 8; public void kieker.examples..CRM.getOffers();
<no-session-id>; 6488138950668976129; 128315649881782395 3;
128315649882000736 7; Osterinsel; 2; 1
```

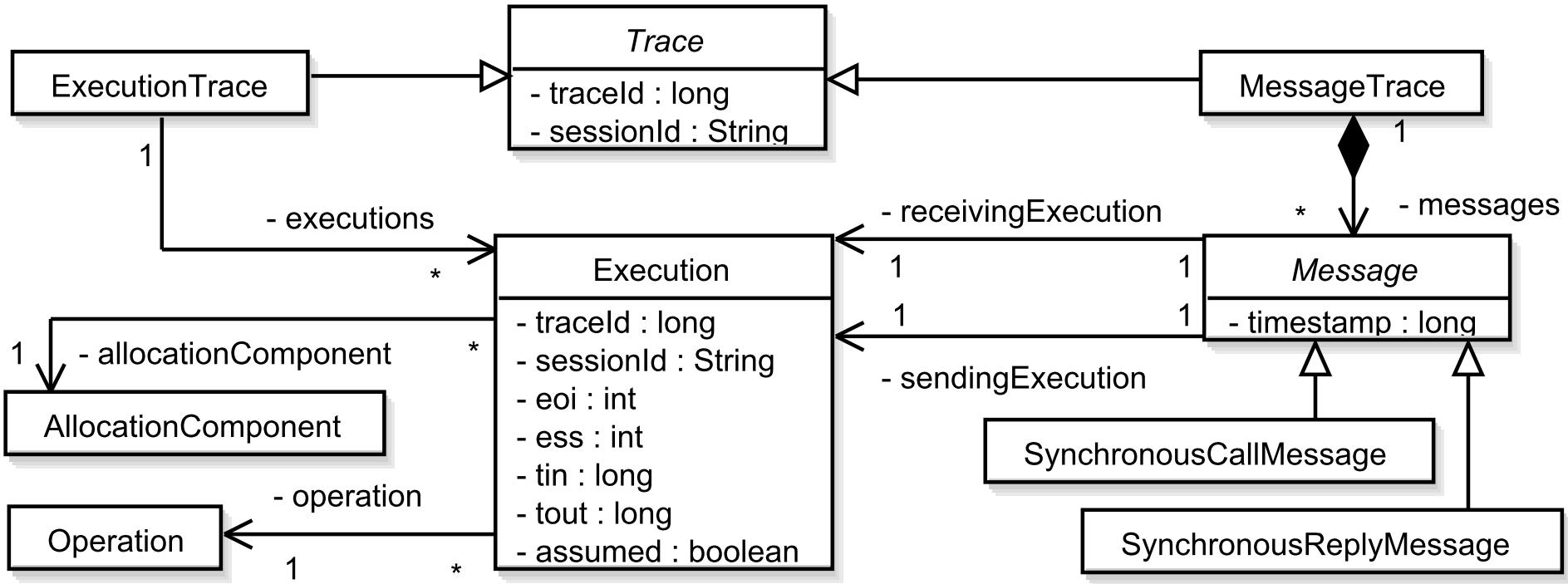
The meaning of this record:

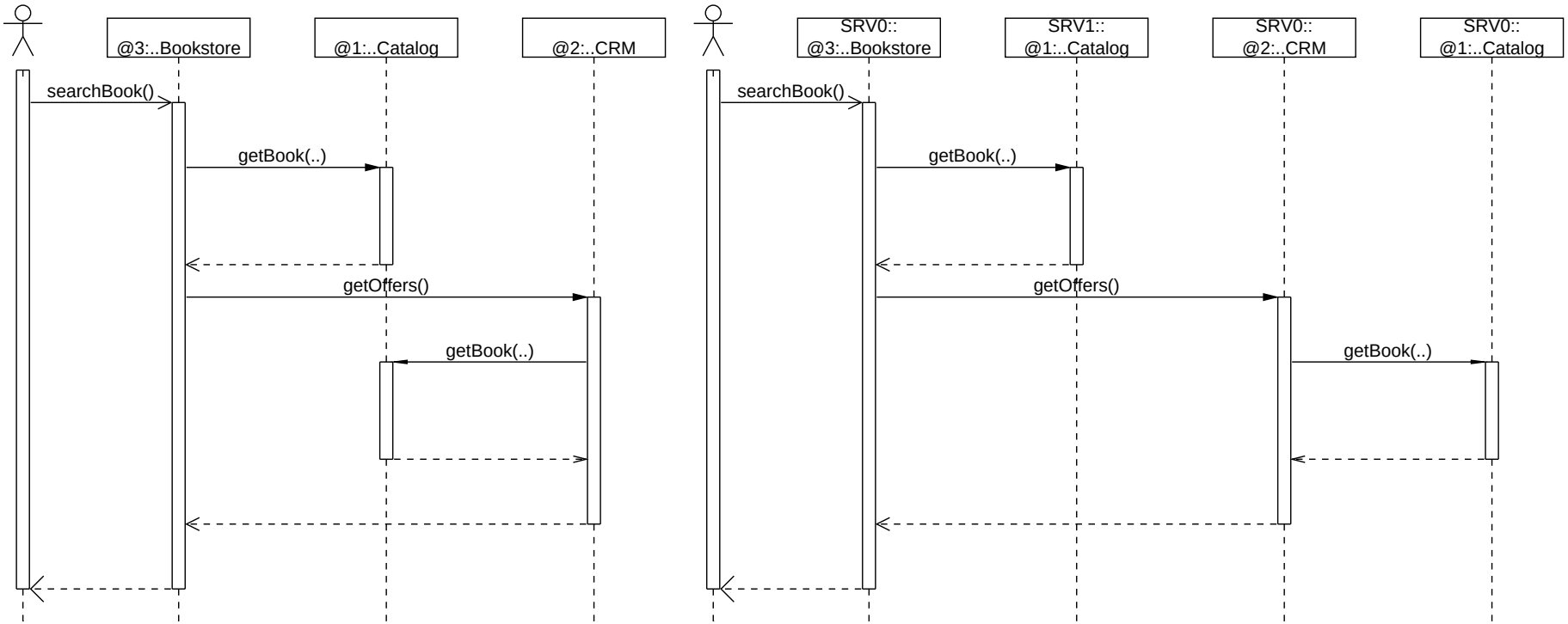
1. Type of monitoring record (see `kieker.map`; here: `OperationExecutionRecord`)
2. Logging timestamp (time in ns)
3. Operation signature (fully qualified)
4. Session id (only with web applications)
5. Trace id (unique id of the trace)
6. t_{in} (start time of execution)
7. t_{out} (end time of execution)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

Kieker's System Meta-Model



Kieker's Trace Meta-Model

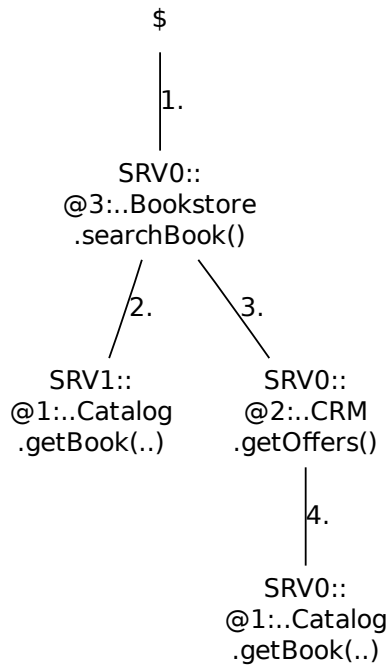




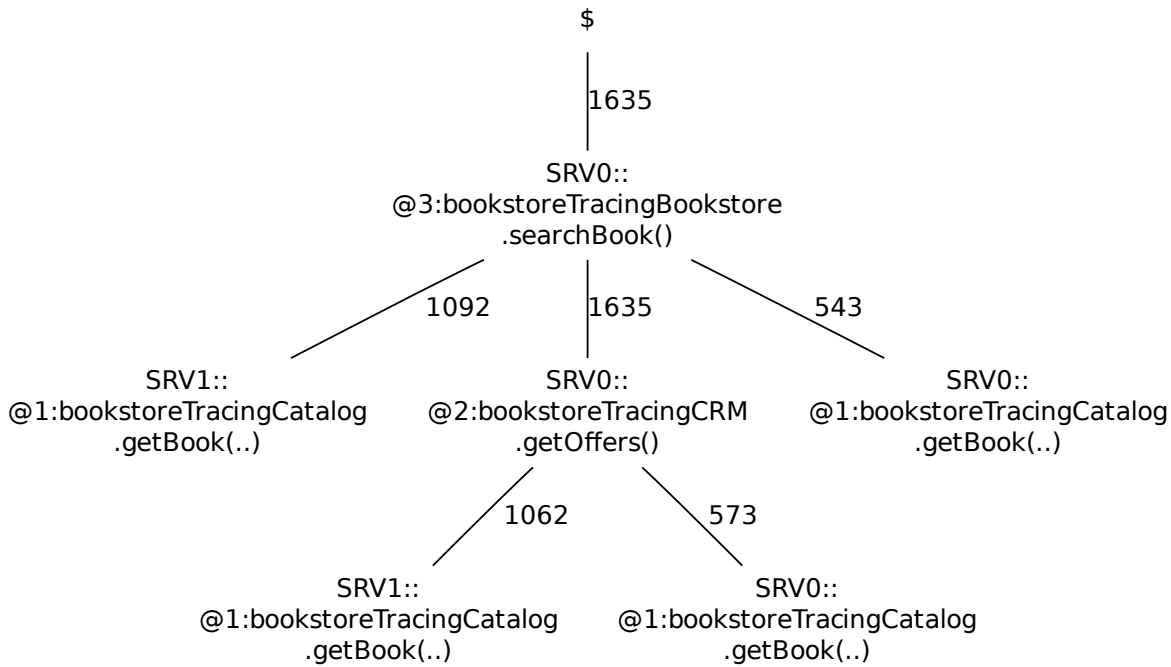
(a) Assembly-level view

(b) Deployment-level view

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

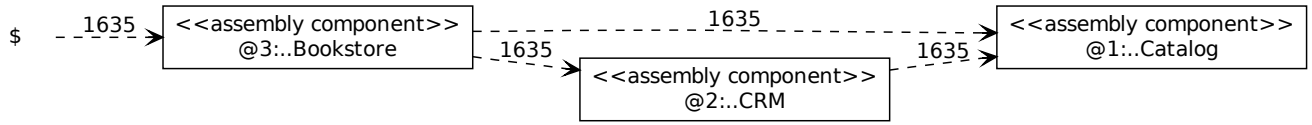


(a) Dynamic call tree **(single trace)**

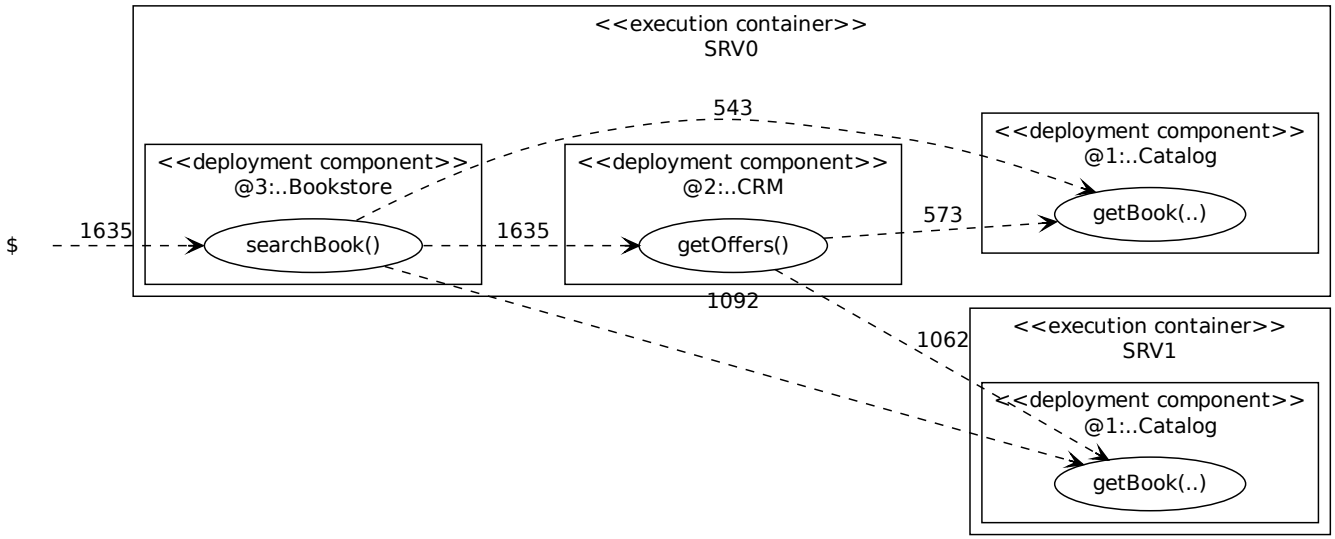


(b) **Aggregated** deployment-level call tree

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

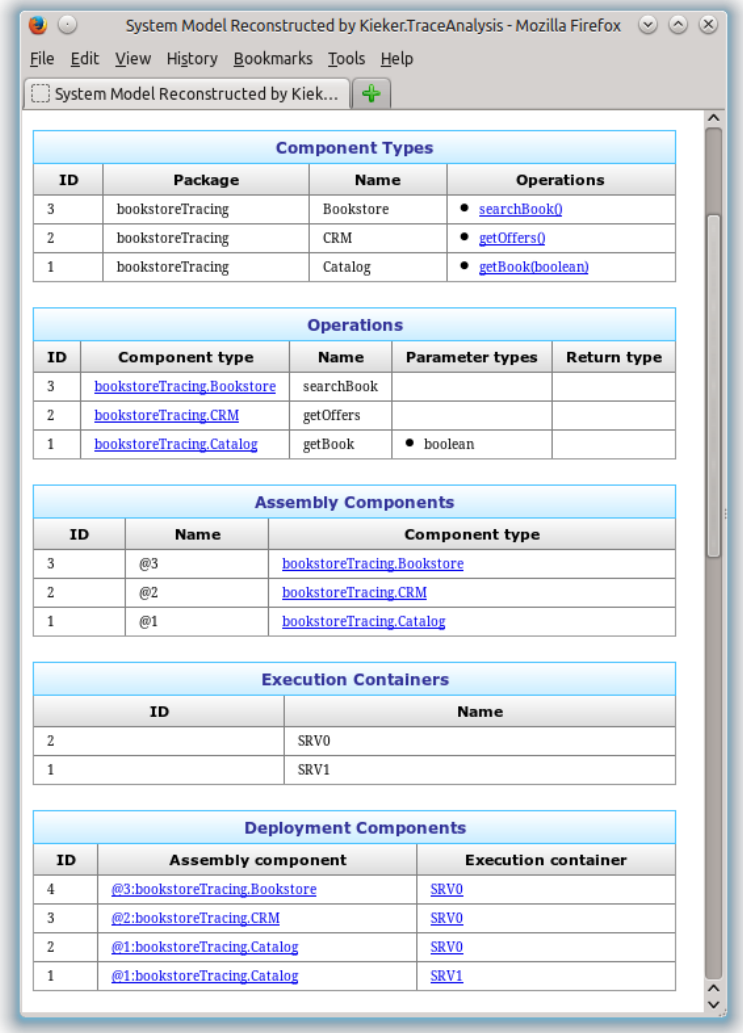


(a) Assembly-level **component** dependency graph



(b) Deployment-level **operation** dependency graph

1. Sequence diagrams
2. Dynamic call trees
3. Hierarchical calling dependency graphs
4. **System model** (here: HTML representation)



Component Types

ID	Package	Name	Operations
3	bookstoreTracing	Bookstore	• searchBook()
2	bookstoreTracing	CRM	• getOffers()
1	bookstoreTracing	Catalog	• getBook(boolean)

Operations

ID	Component type	Name	Parameter types	Return type
3	bookstoreTracing.Bookstore	searchBook		
2	bookstoreTracing.CRM	getOffers		
1	bookstoreTracing.Catalog	getBook	• boolean	

Assembly Components

ID	Name	Component type
3	@3	bookstoreTracing.Bookstore
2	@2	bookstoreTracing.CRM
1	@1	bookstoreTracing.Catalog

Execution Containers

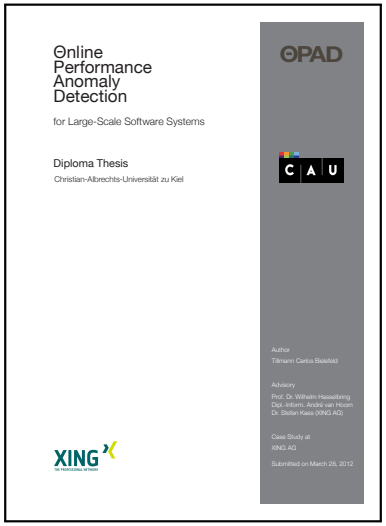
ID	Name
2	SRV0
1	SRV1

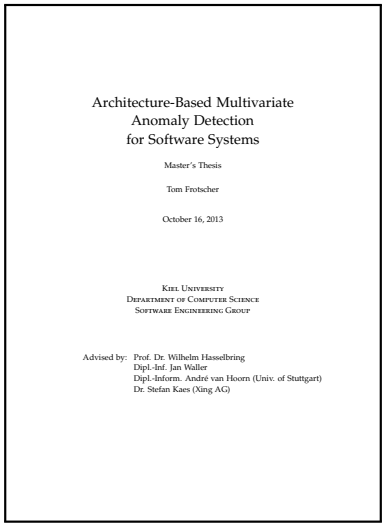
Deployment Components

ID	Assembly component	Execution container
4	@3:bookstoreTracing.Bookstore	SRV0
3	@2:bookstoreTracing.CRM	SRV0
2	@1:bookstoreTracing.Catalog	SRV0
1	@1:bookstoreTracing.Catalog	SRV1

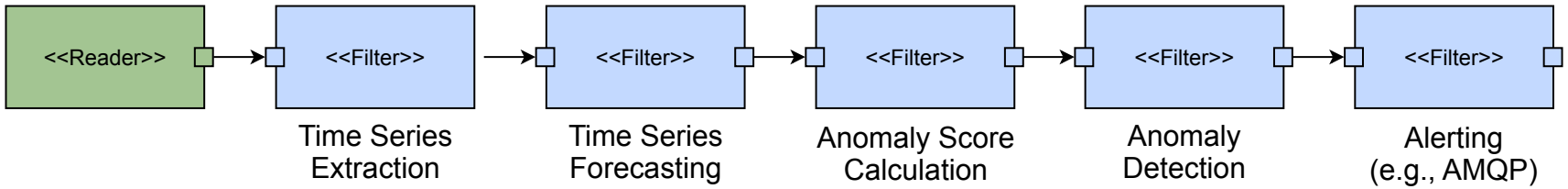
Details to be found in

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



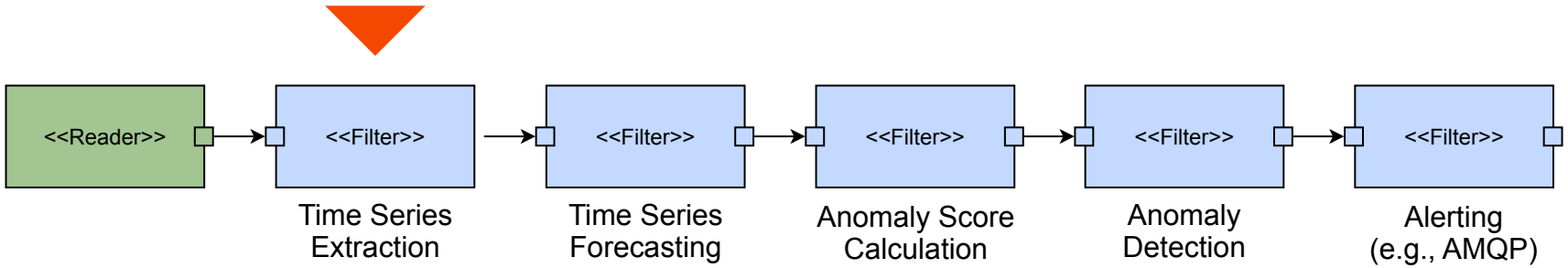


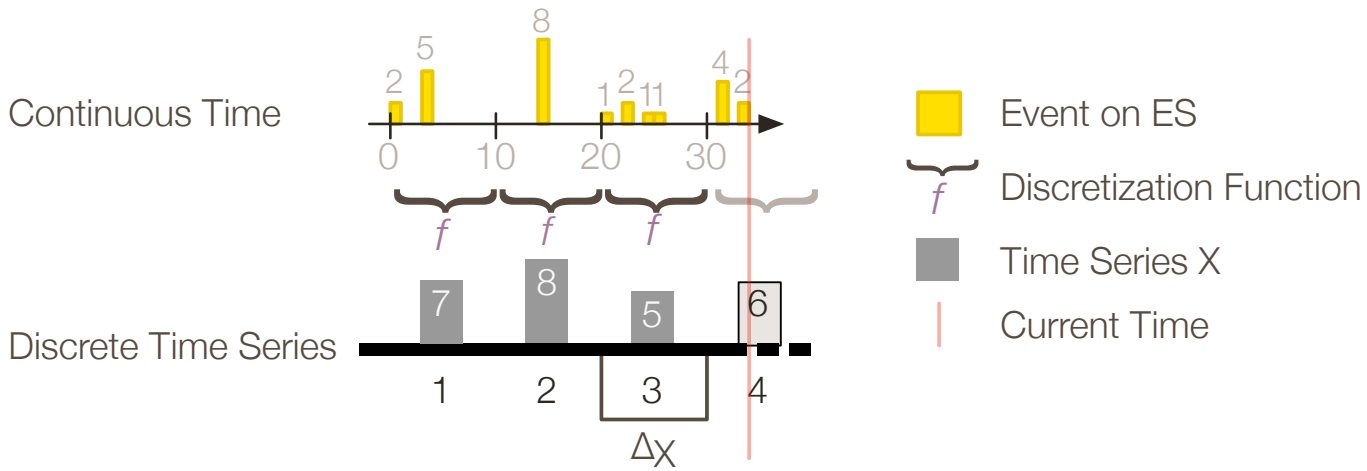
ΘPAD Processing Steps



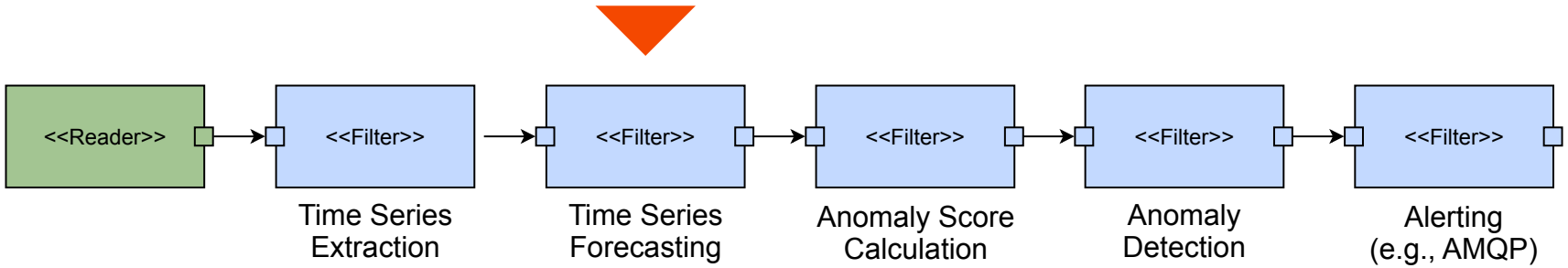
Step 1: Time Series Extraction

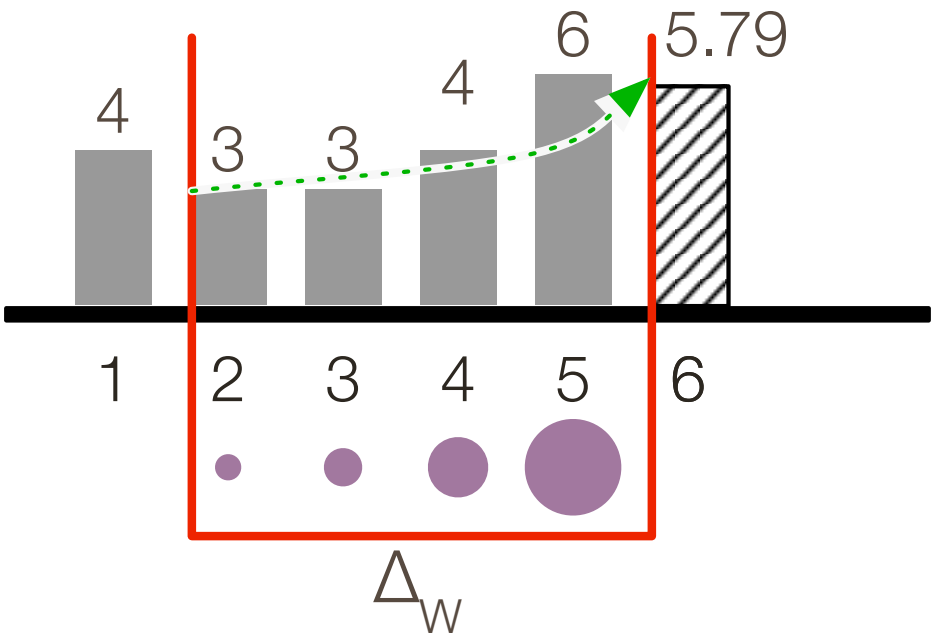
ΘPAD Processing Steps (cont'd)





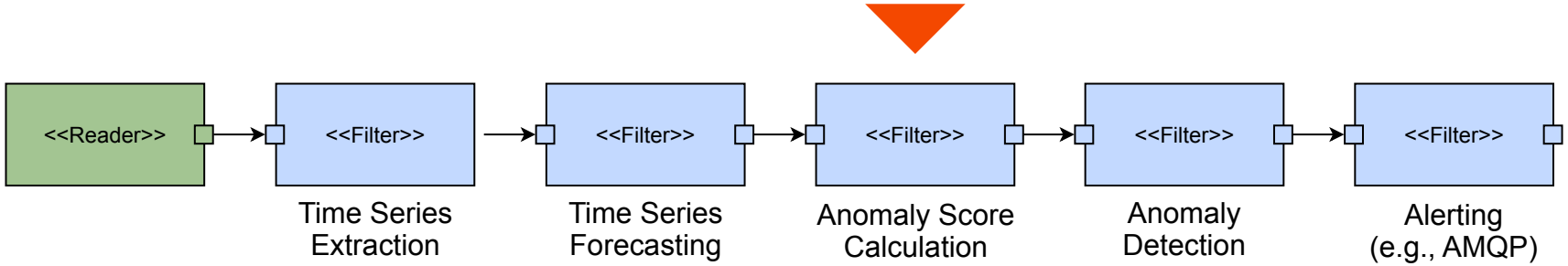

```
select sum(value) as aggregation
from MeasureEvent.win:time_batch( 1000 msec )
```

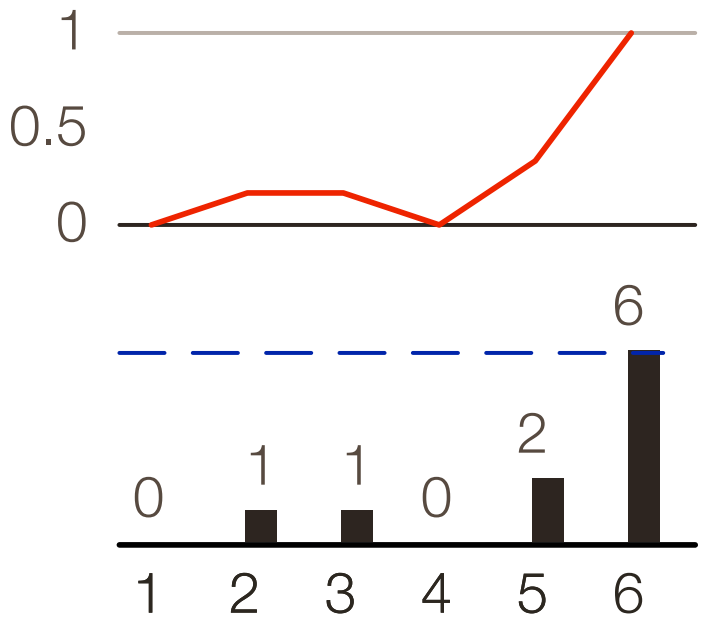

<<Reader>>
<<Filter>>
<<Filter>>
<<Filter>>
<<Filter>>
<<Filter>>
Time Series Extraction
Time Series Forecasting
Anomaly Score Calculation
Anomaly Detection
Alerting (e.g., AMQP)


4
3
3
4
6
5.79
1
2
3
4
5
6
Δ_W

Step 3: Anomaly Score Calculation

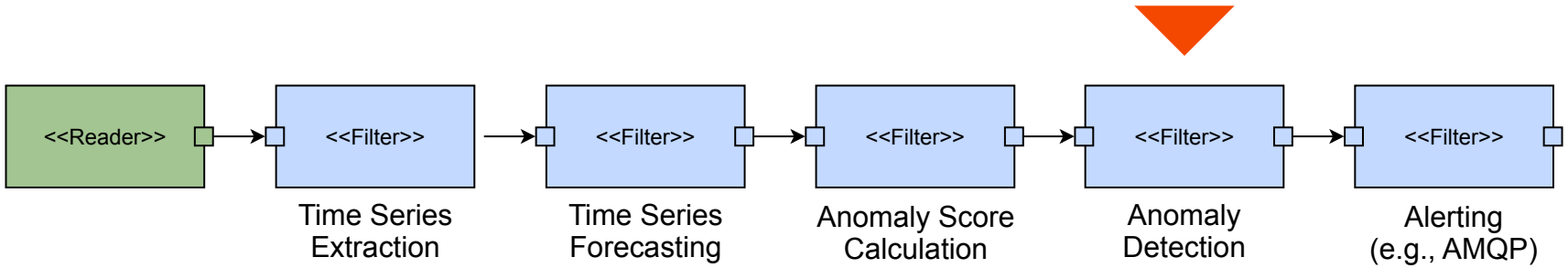
ΘPAD Processing Steps (cont'd)

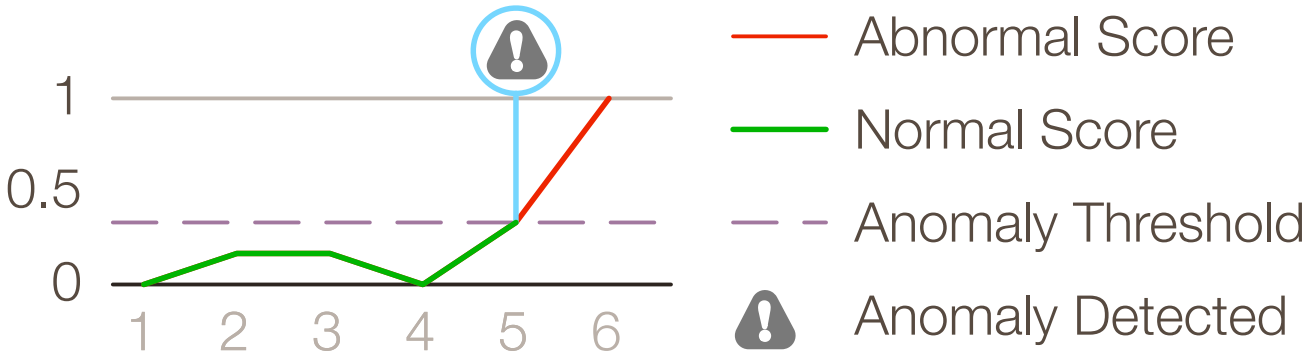




Step 4: Anomaly Detection

ΘPAD Processing Steps (cont'd)





Framework Features & Extension Points

Kieker

- Modular, flexible, and extensible architecture (Probes, records, readers, writers, filters etc.)
- Pipes-and-filters framework for analysis configuration
- Distributed tracing (logging, reconstruction, visualization)
- Low overhead (designed for continuous operation)
- Evaluated in lab and industrial case studies

b+m business IT management · cewe color · dataport · EWE · HSH NORDBANK · XING

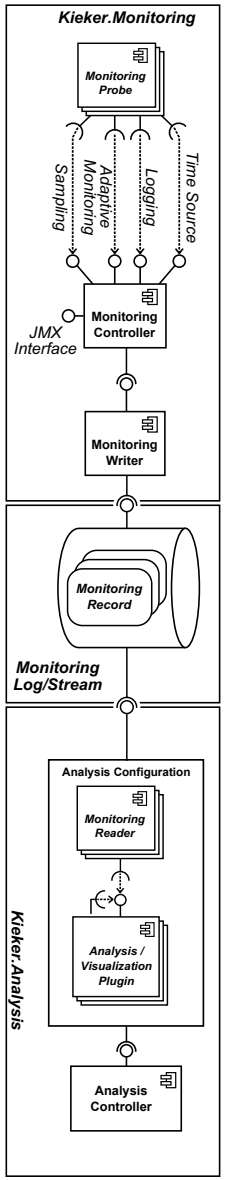
Kieker is open-source software (Apache License, V. 2.0)

`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`

spec Research



Monitoring Probes/Samplers	Control-flow tracing	Manual	AspectJ	Servlet	Spring
		CXF/SOAP	VB6, COBOL, ...		
		<custom interception technology>			
	Resource monitoring	Servlet	Sigar	CPU utilization	
				Memory usage	
		<custom technology>			
	<custom monitoring probe>				
Time Sources	System time				
	<custom time source>				
Monitoring Writers	File system	Database (SQL)	TCP/IP		
	Java Messaging Service (JMS)				
	Java Management Ext. (JMX)				
	Named pipe	Output stream			
	<custom monitoring writer>				
Monitoring Records	Operation execution				
	Control-flow events				
	CPU utilization	Memory/swap usage			
	Resource utilization				
	Current time				
	<custom monitoring record type>				
Monitoring Readers	File system				
	Java Messaging Service (JMS)				
	Java Management Ext. (JMX)				
	Database (SQL)	TCP/IP			
	Named pipe	Real-time replayer			
	<custom monitoring reader>				
Analysis/Visualization Plug-ins	Pipes and filters framework	Performance Analysis			
		Architecture reconstr.	Trace analysis	Visualization	Dependency graphs
					Sequence diagrams
					Call graphs
					<custom visualization>
				<custom trace analysis>	
			<custom reconstruction plug-in>		
		<custom analysis plug-in/tool>			

Literature

T. C. Bielefeld. Online performance anomaly detection for large-scale software systems, Mar. 2012. Diploma Thesis, Kiel University.

B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, 35(5):684–702, 2009. ISSN 0098-5589. doi: http://doi.ieeecomputersociety.org/10.1109/TSE.2009.28.

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 *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days (KPDAYS '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.

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.

G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. Lopes, C. Maeda, and A. Mendhekar. Aspect-oriented programming. Position paper from the Xerox PARC Aspect-Oriented Programming project, Xerox Paolo Alto Research Center, 1996.

Kieker Project. Kieker 1.9 user guide. `http://kieker-monitoring.net/documentation/`, Apr. 2014a.

Kieker Project. Kieker web site. `http://kieker-monitoring.net/`, 2014b.

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.

Literature (cont'd)

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. *Model-Driven Online Capacity Management for Component-Based Software Systems*. Number 2014/<++NUMBER> in Kiel Computer Science Series. Department of Computer Science, Kiel University, Kiel, Germany, 2014. Dissertation (under review), Faculty of Engineering, Kiel University.

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.

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/