Adaptive Instrumentation of Java-Applications for Experiment-Based Performance Analysis

Henning Schulz, Albert Flaig, Alexander Wert and André van Hoorn

Symposium on Software Performance 2014

28.11.2014
Motivation

Application

Performance?

Performance Engineer

Instrumentation

Introduction ➔ Kieker & AIM ➔ Experiment-Based Scenarios ➔ Integration of Kieker & AIM ➔ Conclusion

2/22  28-11-2014  Adaptive Instrumentation of Java-Applications for Experiment-Based Performance Analysis

KIT – Software Design and Quality Group
University of Stuttgart – Reliable Software Systems Group
Instrumentation Overhead

start

end (no instrumentation)

end (full instrumentation)

methodA()

methodB()

methodC()

methodD()

Kieker & AIM

Integration of Kieker & AIM

Conclusion
Adaptive Instrumentation

start \hspace{2cm} \text{end (no instrumentation)} \hspace{2cm} \text{end (full instrumentation)}

\text{methodA()}

\text{methodB()}

\text{methodC()}

\text{methodD()}

Kieker & AIM

Conclusion
Kieker vs. AIM

<table>
<thead>
<tr>
<th>Kieker (<a href="http://kieker-monitoring.net">http://kieker-monitoring.net</a>)</th>
<th>AIM (<a href="http://sopeco.github.io/AIM">http://sopeco.github.io/AIM</a>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrumentation at application start-up</td>
<td>Instrumentation on demand</td>
</tr>
<tr>
<td>Statically adaptive instrumentation</td>
<td>Dynamically adaptive</td>
</tr>
<tr>
<td>Extensible (writer, etc.)</td>
<td>Extendable (writer, etc.)</td>
</tr>
<tr>
<td>Analysis infrastructure</td>
<td>No analysis at all</td>
</tr>
<tr>
<td>Focus on production</td>
<td>Focus on experiments</td>
</tr>
<tr>
<td>Regexps for adaptive monitoring</td>
<td>Instrumentation description language</td>
</tr>
</tbody>
</table>

Use dynamic adaptive instrumentation from AIM
Use instrumentation description language

Use Kieker's monitoring & analysis infrastructure

Introduction  ➔  Kieker & AIM  ➔  Experiment-Based Scenarios  ➔  Integration of Kieker & AIM  ➔  Conclusion
AIM IN EXPERIMENT-BASED CONTEXT

Introduction  Kieker & AIM  Experiment-Based Scenarios  Integration of Kieker & AIM  Conclusion

AIM

:Instrumentation Description

instrumentation entity

:Instrumentation Entity

:Response Time Probe

how

:Method Scope

pattern = my.package.*(int)

where

JVM

Target Application

Injected Probes

AI Agent

Measurement Data

Java Process

Experiment-Based Scenarios

Conclusion
Evaluation

AIM is comparable to existing approaches

- Compare to Kieker and DiSL
- **Instrumentation process:**
  - fast for < 500 methods
  - comparatively slow for > 500 methods
- **Instrumentation overhead:** smaller than Kieker’s

Benefits of adaptable instrumentation

- Automated Palladio Resource Demand Estimation

Introduction – Kieker & AIM – Experiment-Based Scenarios – Integration of Kieker & AIM – Conclusion
Automated Estimation of Palladio Resource Demands

Derive Call Tree

Measure Response Times of All Methods in the Call Tree

Derive Resource Demands by Subtraction of Response Times

methodA()
methodB()
methodC()
methodD()

TPC-W Deployment
Palladio Model of TPC-W
Insert Resource Demands into PCM
Simulate
Compare Simulation Results

10%
100%
90%
80%
70%
60%
50%
40%
30%
20%

response time

methodA()
methodB()
methodC()
methodD()

Henning Schulz - AIM in Experiment-Based Context

Introduction ➔ Kieker & AIM ➔ Experiment-Based Scenarios ➔ Integration of Kieker & AIM ➔ Conclusion
Comparing PRD Qualities

<table>
<thead>
<tr>
<th></th>
<th>mean [ms]</th>
<th>mean error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>4.52</td>
<td>-</td>
</tr>
<tr>
<td>AIM (adaptive)</td>
<td>4.52</td>
<td>0.01</td>
</tr>
<tr>
<td>AIM (full)</td>
<td>6.21</td>
<td>37.3</td>
</tr>
<tr>
<td>Kieker (adaptive)</td>
<td>4.95</td>
<td>9.4</td>
</tr>
<tr>
<td>Kieker (full)</td>
<td>6.57</td>
<td>45.4</td>
</tr>
</tbody>
</table>

Legend:  
R: Reference  
A_a: AIM adaptive  
A_f: AIM full  
K_a: Kieker adaptive  
K_f: Kieker full

Measurement Simulation

Introduction  ➤ Kieker & AIM  ➤ Experiment-Based Scenarios  ➤ Integration of Kieker & AIM  ➤ Conclusion
INTEGRATION OF KIEKER & AIM

Statically Adaptive Monitoring in Kieker

Expensive
Checks all Instrumentation Patterns
Result is cached

Illustration based on Waller et al. (2012)
Dynamically Adaptive Monitoring in Kieker

Illustration based on Waller et al. (2012)
**AIM Integration in Kieker**

**Java Process**

**Kieker**
- Monitoring Controller
- Instrumentation Patterns
- Execution Probe
- Record
- Kieker Instrumentor
- BCInjector
- Methods to instrument
- Javassist
- Instrumented Classes

**JVM Instr. API**
- Pattern List
- Loaded Classes
- Instrumentation
- Target Application
- Hotswap
- Instrumented Classes

**AIM**
- Probe Snippet

**Data Flow**
- Component
- Piece of Data

**Integration of Kieker & AIM**

**Conclusion**

**Introduction**
- Kieker & AIM
- Experiment-Based Scenarios

**14/22 28-11-2014 Albert Flaig - Integration AIM/Kieker**

**KIT – Software Design and Quality Group**
University of Stuttgart – Reliable Software Systems Group
Evaluation

- Bytecode instrumentation vs. Manual

1. Overhead
   - RQ1: How does it fare in terms of performance?

2. Turnaround
   - RQ2a: How does it fare in terms of turnaround time?
   - RQ2b: Does it scale well?

3. Lost Transactions
   - RQ3: Is it reliable?
Evaluation 1: Overhead Comparison

Introduction

Kieker & AIM

Experiment-Based Scenarios

Integration of Kieker & AIM

Conclusion

Execution Time (µs)

0 2 4 6 8 10 12 14

Manual Instrumentation

Bytecode Instrumentation

No Probe

Deactivated Probe

Collecting Data

Writer

16/22 28-11-2014 Albert Flaig - Integration AIM/Kieker

KIT – Software Design and Quality Group
University of Stuttgart – Reliable Software Systems Group
Evaluation 2: Turnaround (Setting)

- Bytecode Instrumentation
  - Turnaround = Activating probe + first execution of probe
  - Activating probe invokes BCI

- Manual Instrumentation
  - Turnaround = Activating probe + first execution of probe
  - First execution of probe invokes caching

- Instrumenting 1,000 dummy classes

- Comparison runtime BCI vs. caching
Evaluation 2a: Turnaround Time

![Graph showing turnaround time comparison between Bytecode Instrumentation and Manual Instrumentation.](Graph.png)

**Introduction**

- **Kieker & AIM**
- **Experiment-Based Scenarios**
- **Integration of Kieker & AIM**
- **Conclusion**

**Evaluation 2a: Turnaround Time**

- **Bytecode Instrumentation**
- **Manual Instrumentation**

**Graph Details**

- **Axes:**
  - X-axis: Buckets of Instrumentations (each 10)
  - Y-axis: Turnaround Time (ms)

**Data Points**

- Bytecode Instrumentation:
  - Smooth trend indicative of consistent performance.
- Manual Instrumentation:
  - Fluctuations indicating varying performance.

**Conclusion**

- Integration of Kieker & AIM is discussed in detail with experiment-based scenarios.
Evaluation 2b: Scalability Comparison

**Bytecode Instrumentation**

**Manual Instrumentation**

### Turnaround Time (ms)

<table>
<thead>
<tr>
<th># of classes</th>
<th>5</th>
<th>15</th>
<th>25</th>
<th>35</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytecode Instrumentation</td>
<td>500</td>
<td>1000</td>
<td>1500</td>
<td>2500</td>
<td>5000</td>
</tr>
<tr>
<td>Manual Instrumentation</td>
<td>0.35</td>
<td>0.5</td>
<td>0.7</td>
<td>1.0</td>
<td>1.5</td>
</tr>
</tbody>
</table>

### Conclusion

Integration of Kieker & AIM

Experiment-Based Scenarios

Introduction
Conclusion

Core Results

- Kieker & AIM synergies
- Small overhead reduction and higher flexibility
- High turnaround time
  → relevant for production
  → needs optimization

Future Work

- Kieker supporting the instrumentation description language
- Optimize Performance of AIM’s instrumentation step
- Usage in bigger context (e.g., performance problem diagnostics)
- Integration into Kieker release
References

[Dmitriev, 2004]
M. Dmitriev, “Profiling java applications using code hotswapping and dynamic call graph revelation”

[Ehlers et al, 2011]

[Moret et al, 2011]
P. Moret, W. Binder, and E. Tanter, “Polymorphic bytecode instrumentation”

[Marek et al, 2012]

[van Hoorn et al, 2009]
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”
AIM Integration in Kieker Detailed

Process of External Tool

Data Flow

Component

Piece of Data

Java Process

Kieker

Monitoring Controller

Instrumentation Patterns

JMX Controller

Probe Controller

Pattern List

Loaded Classes

Kieker Class Loader

JVM Instr. API

Instrumentation

Target Application

Hotswap

Instrumented Classes

AIM

Probe Snippet

BCInjector

Methods to instrument

Classes to instrument

Probe Snippets

Javassist
Evaluation 1: Overhead Comparison

Manual Instrumentation

Bytecode Instrumentation

Mean execution time of ...
- Writer
- Deactivated Probe
- Collecting Data
- No Probe

Execution Time (μs)

500 750 1000

500 750 1000
Instrumentation Duration (TPC-W)

Legend:  
A: Instr. with AIM  
K: Instr. with Kieker  
D: Instr. With DiSL

(8 methods)  
(251 methods)  
(380 methods)
Instrumentation Duration (Liferay)

Legend:

A: Instr. with AIM
K: Instr. with Kieker
D: Instr. With DiSL

Response Time [s]

servlet
(6 methods)

public
(2810 methods)

full
(3270 methods)
Overhead (MooBench & TPC-W)

Legend:  
N: No Instr.  
A: Instr. with AIM  
K: Instr. with Kieker

(8 methods) (251 methods) (380 methods)
Overhead (Liferay)

Legend:  
N: No Instr.  
A: Instr. with AIM  
K: Instr. with Kieker

Response Time [ms]

0 2000 4000 6000 8000

Servlet (6 methods)  
Public (2810 methods)  
Full (3270 methods)