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

Performance Engineer

Application

Performance?

Instrumentation

Kieker & AIM

Experiment-Based Scenarios

Integration of Kieker & AIM

Conclusion
Instrumentation Overhead

start

methodA()

methodB()

methodC()

methodD()

end (no instrumentation)

time

end (full instrumentation)

Conclusion

Integration of Kieker & AIM

Experiment-Based Scenarios

Kieker & AIM

Introduction

3/22 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
Adaptive Instrumentation

**Method Call**: methodA() → methodB() → methodC() → methodD() → methodD()

**Time Line**:
- start
- end (no instrumentation)
- end (full instrumentation)

**Conclusion**
Integration of Kieker & AIM
Experiment-Based Scenarios
Integration of Kieker & AIM
Conclusion
## Kieker vs. AIM

<table>
<thead>
<tr>
<th></th>
<th></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**

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

**KIT – Software Design and Quality Group**
University of Stuttgart – Reliable Software Systems Group
AIM IN EXPERIMENT-BASED CONTEXT

Henning Schulz - AIM in Experiment-Based Context

**Experiment-Based Scenarios**

- **Instrumentation Description**
  - Instrumentation entity
  - :Method Scope
    - pattern = my.package.*(int)
  - :Response Time Probe
- **Measurement Data**
- **Target Application**
  - Injected Probes

**Java Process**

**Introduction**

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

1. Derive Call Tree
2. Measure Response Times of All Methods in the Call Tree
3. Derive Resource Demands by Subtraction of Response Times

Method A
Method B
Method C
Method D

Response Times

\[ T_{\text{res}} = T_1 - T_2 \]

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<sub>a</sub>: AIM adaptive
- A<sub>f</sub>: AIM full
- K<sub>a</sub>: Kieker adaptive
- K<sub>f</sub>: Kieker full

Measurement Simulation

Introduktion | 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

JMX Client

Process of External Tool

Data Flow
Component
Piece of Data

Java Process

Kieker
Monitoring Controller
Instrumentation Patterns
Execution Probe
Record

AIM
Probe Snippet
BCInjector

JVM Instr. API
Pattern List
Loaded Classes

Instrumentation

Target Application
Hotswap
Instrumented Classes

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

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?

MooBench

Introduction ➔ Kieker & AIM ➔ Experiment-Based Scenarios ➔ Integration of Kieker & AIM ➔ Conclusion
Evaluation 1: Overhead Comparison

![Bar chart comparing execution time (µs) between Manual Instrumentation and Bytecode Instrumentation for different probes and scenarios.]

- **Manual Instrumentation**
  - No Probe
  - Deactivated Probe
  - Collecting Data
  - Writer

- **Bytecode Instrumentation**
  - No Probe
  - Deactivated Probe
  - Collecting Data
  - Writer

**Conclusion**

Integration of Kieker & AIM
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]

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

**Introduction**
**Kieker & AIM**
**Experiment-Based Scenarios**
**Integration of Kieker & AIM**
**Conclusion**
Evaluation 2b: Scalability Comparison

**Bytecode Instrumentation**

- Turnaround Time (ms)
- # of classes

**Manual Instrumentation**

- Turnaround Time (ms)
- # of classes

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

19/22 28-11-2014 Albert Flaig - Integration AIM/Kieker

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

JMX Client → Process of External Tool → Data Flow

Component

Piece of Data

Kieker

Monitoring Controller → JMX Controller → Probe Controller

Instrumentation Pattern

Class Loaded

Kieker Class Loader

JVM Instr. API

Instrumentation

Target Application

Hotswap

AI M

Probe Snippet → BCInjector → Methods to instrument

Classes to instrument

Probe Snippets

Javassist

Instrumented Classes

Java Process

Execution Probe → Record → Kieker Instrumentor

Pattern List

Loaded Classes

Predicted Classes
Evaluation 1: Overhead Comparison

Manual Instrumentation

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

Bytecode Instrumentation

Mean execution time of ...
- Writer
- Collecting Data
- No Probe
Instrumentation Duration (TPC-W)

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

- servlet (8 methods)
- public (251 methods)
- full (380 methods)
Instrumentation Duration (Liferay)

Legend:

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

- 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

Response Time [μs]
Response Time [ms]

Single Method
Servlet
Public
Full

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

Legend:  

Response Time [ms]

N  A  K
Servlet (6 methods)

A  K
Public (2810 methods)

A  K
Full (3270 methods)