Towards Performance Awareness in Java EE Development Environments
Symposium on Software Performance (SOSP) 2014

Alexandru Danciu\textsuperscript{1}, Andreas Brunnert\textsuperscript{1}, Helmut Krcmar\textsuperscript{2}
\textsuperscript{1}fortiss GmbH, \textsuperscript{2}Technische Universität München
Agenda

- Motivation
- Performance Awareness Approach
  - Data Collection
  - Data Aggregation
  - Model Generation
  - Developer Feedback
- Related Work
- Limitations
- Conclusion
- Future Work
Agenda

• Motivation
• Performance Awareness Approach
  – Data Collection
  – Data Aggregation
  – Model Generation
  – Developer Feedback
• Related Work
• Limitations
• Conclusion
• Future Work
Motivation

- The performance of a new components depends among others on the performance of reused components.

- Reused components are assigned to different organizational units and are subject to constant change.

- For evaluating the performance impact of component reuse developers require access to monitoring data and experience in the performance engineering domain.

- To introduce performance awareness in integrated development environments (IDE) for Java Enterprise Edition (EE) applications.
Motivation
Component-based Software Engineering

- The emphasis of component-based development is on the specification of loosely coupled independent components to enable separation of concerns and reuse across the system.
Motivation

Implications for Developers

• Component developers reuse other components to implement the required functionality.

• Component developers are facing questions such as:
  – Are the service-level agreements (SLA) imposed to my component violated by reusing a specific component?
  
  – Can the SLAs imposed to my component be achieved with the current component implementation?
  
  – Does a particular change in the control flow of my component lead to an SLA violation?
  
  – How is the performance of my component changing for varying workloads?
Motivation

Technical and Organizational Challenges

System Lifecycle
- Development
- Operations

DevOps

IT Governance
- Unilateral
- Multilateral

System Architecture
- Monolithic
- System-of-Systems

Technical and Organizational Challenges (Brunnert et al. 2014)
Motivation

Technical and Organizational Challenges

- Identification of reused component's lifecycle status
- Identification of reused component's deployment information
- Identification of organizational unit managing the component
- Acquisition of monitoring data
- Employment of performance management tool

System Lifecycle

IT Governance

System Architecture

Technical and Organizational Challenges (Brunnert et al. 2014)
Agenda

• Motivation
• Performance Awareness Approach
  – Data Collection
  – Data Aggregation
  – Model Generation
  – Developer Feedback
• Related Work
• Limitations
• Conclusion
• Future Work
Proposed Performance Awareness Approach

- Collection of performance data for running Java EE components.
- Aggregation of monitoring data over different component instances, versions and user workloads.
- Estimation of the response time for component operations under development.
- Support feedback to component developers within the IDE.

Performance Awareness Workflow Adapted from Brunnert et al. (2013)
Data Collection

- Response time behavior of running components within the application landscape is collected using the Kieker framework.
- Applications are instrumented using aspect-oriented programming.
- Using the adaptive monitoring feature, data collection is activated for specific component operations.
- Monitoring records are passed by a custom monitoring writer via a web-service call to a database application that serves as a backend.
- The web-service deserializes the record and stores it to a relational database.

Kieker Data Collection (Kieker Project, 2014)
Data Aggregation

• Aggregation of individual records to performance curves (Wert et al. 2012) and stochastic expressions describing the response time behavior of operations in dependence on a set of input parameters such as workload characteristics.
Factors Influencing Component Performance (Koziolek, 2010)
Model Generation

- The model generation is based on the approaches of Kappler et al. (2008) and Becker et al. (2010).
Implicit estimation of the response time of component operations is automatically performed each time changes to the code are saved:

- The estimation is performed using an analytical solver.
- If the estimated response time exceeds a specific threshold, a notification will be displayed within the code editor.
- Thresholds are configured with default values and can be adjusted by the developer.

```
Component.java

public void operation_1(){
    ...
}

public void operation_2(){
    ...
}
```
Developer Feedback

- Detailed response time estimation of all component operations triggered by the developer:
  - A simulator is used to process the PCM model.
  - The response time behavior of reused components is modeled using performance curves.
  - Results are displayed as probability density function in a separate view.

<table>
<thead>
<tr>
<th>Simulation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean: X ms</td>
</tr>
<tr>
<td>Median: Y ms</td>
</tr>
</tbody>
</table>
Agenda

• Motivation
• Performance Awareness Approach
  – Data Collection
  – Data Aggregation
  – Model Generation
  – Developer Feedback
• Related Work
• Limitations
• Conclusion
• Future Work
Weiss et al. (2013) propose an approach for evaluating the performance of JPA persistence service operations based on tailored benchmarks applications during the implementation phase.

Heger et al. (2013) present an approach to integrate performance regression root cause analysis into development environments.

Bureš et al. (2014) propose the integration of performance awareness in the development life cycle of autonomic component ensembles.
Related Work

Automatic Performance Model Generation

• Brosig et al. (2011) present a semi-automatic approach for extracting PCM models from Java EE applications based on monitoring data collected at run-time using WebLogic-specific monitoring tools.

• Brunnert et al. (2013) present a similar approach that is applicable for all Java EE server products based on data collected from custom Servlet filters and EJB interceptors.

• Krogmann et al. (2010) use static and dynamic analysis to reverse engineer Java applications.
Agenda

• Motivation
• Performance Awareness Approach
  – Data Collection
  – Data Aggregation
  – Model Generation
  – Developer Feedback
• Related Work
• Limitations
• Conclusion
• Future Work
Limitations

• The application of this approach is only useful if component reuse exists.

• The prediction of response times in dependence on the workload is only possible if the behavior of reused components was measured under various workloads.
Agenda

• Motivation
• Performance Awareness Approach
  – Data Collection
  – Data Aggregation
  – Model Generation
  – Developer Feedback
• Related Work
• Limitations
• Conclusion
• Future Work
Conclusion

• Approach to provide feedback on the estimated response time of Java EE components to developers within the IDE:
  
  – Approach does not require any knowledge about reused components or experience in the performance engineering domain.
  
  – Developers are not required to make efforts for obtaining access to performance data and do not have to employ additional tools for processing these data.
  
  – Feedback is provided automatically to the developer and requires no additional effort.
Agenda

- Motivation
- Performance Awareness Approach
  - Data Collection
  - Data Aggregation
  - Model Generation
  - Developer Feedback
- Related Work
- Limitations
- Conclusion
- Future Work
Future Work

Evaluation

• Evaluation of the approach within a SPECjEnterprise 2010 deployment.
Future Work

Enhancements

• Integration of the approach with a Performance Model Management Repository.

• Support developer in comparing the performance of different component versions.

• Enable the developer to refine generated PCM models, for example by specifying probabilities for branches within an RDSEFF (e.g., by annotating code).
References


Alexandru Danciu, Andreas Brunnert
performancegroup@fortiss.org
pmw.fortiss.org