Towards Performance Awareness in Java EE Development Environments

Alexandru Danciu[1], Andreas Brunnert[1], Helmut Krcmar[2]
[1]fortiss GmbH
Guerickestr. 25, 80805 München, Germany
{danciu, brunnert}@fortiss.org
[2]Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
krcmar@in.tum.de

Abstract: This paper presents an approach to introduce performance awareness in integrated development environments (IDE) for Java Enterprise Edition (EE) applications. The approach predicts the response time of EE component operations during implementation time and presents these predictions within an IDE. Source code is parsed and represented as an abstract syntax tree (AST). This structure is converted into a Palladio Component Model (PCM). Calls to other Java EE component operations are represented as external service calls in the model. These calls are parameterized with monitoring data acquired by Kieker from Java EE servers. Immediate predictions derived using analytical techniques are provided each time changes to the code are saved. The prediction results are always visible within the source code editor, to guide developers during the component development process. In addition to this immediate feedback mechanism, developers can explicitly trigger a more extensive response time prediction for the whole component using simulation. The paper covers the conceptional approach, the current state of the implementation and sketches for the user interface.

1 Introduction

Detecting performance problems of applications systems is essential before their release to the field. However, evaluating the performance of application systems in terms of response time, resource utilization and throughput is a difficult task. Performance tests require realistic environments comprising hardware, middleware, utilized components, test data and a test workload. Load testing is often one of the last steps performed during a development process, even though performance problems can be solved easier, the earlier they are detected.

Performance awareness represents the ability to detect performance problems and to react to them [T˚14]. One of the core targets of this concept is supporting developers with insights on the performance of code they are currently developing. This paper presents an approach to introduce performance awareness in integrated development environments (IDE) for Java Enterprise Edition (EE) applications. Java EE supports the implementa-

Proc. SOSP 2014, Nov. 26–28, 2014, Stuttgart, Germany

tion of component-based software systems. The Java EE specification defines application component types such as applets, servlets and Enterprise JavaBeans [LD13]. 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. Component developers reuse other components to implement the required functionality. The performance of a new component depends among others on the performance of reused components [Koz10]. Therefore 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?

Answering these questions is an increasingly difficult task, due to three main factors: application system architecture, system life cycle and IT governance [BVD$^+$14]. Complex architectures often lead to geographical, organizational, cultural and technical variety. Developers lack the knowledge about the structure and the deployment of reused components. Components are subject to a constant iteration between development and operation. It is difficult for developers to maintain an overview of the performance behavior of components. Additionally, components are assigned to different organizational units. Accessing monitoring data is thus more difficult for the developer. From a technical point of view, the developer needs experience in the performance engineering domain and in using corresponding tools. This article proposes an automated and integrated approach to answer these questions. The approach predicts the response time of component operations during implementation time and presents the results within the IDE of the developer.

The remainder of this paper is organized as follows. Section 2 provides an overview of the approach and describes the three main phases in detail. Section 3 describes existing research related to this approach. Section 4 concludes this article and describes future research directions.

2 Developer Performance Awareness Approach

The goal of the approach presented in this paper is to support developers with response time estimations for Java EE component operations. The main phases and the architecture of the proposed approach are shown in Figure 1. First, performance data of running Java EE components is collected from existing application deployments (1). Our approach is based on the assumption that new components will reuse existing ones to some extent. The collected data is then aggregated over different component instances, versions and user workloads (2). Response time estimations for new components are derived by using Palladio Component Models (PCM). These PCM models are generated based on the source code of the component operations representing the control flow with an emphasis

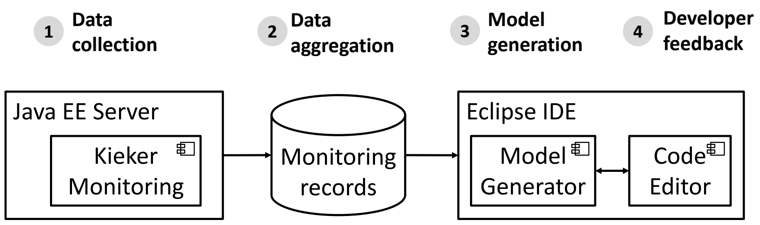


Figure 1: Overview of the approach (structure adapted from [BVK13])

on external calls (3). Two types of estimations can be triggered (4): an immediate estimation after the developer saves a change to the code and an explicitly triggered simulation. The immediate estimation is performed using an analytical solver and stochastic expressions representing the performance behavior of reused components. The prediction results are visible within the source code editor. Simulations are intended to provide more precise predictions and are performed using performance curves [WHW12]. Simulation results will be displayed in a separate view. The data collection and aggregation phases are executed periodically. The model generation and developer feedback phases are triggered by the developer. Each phase is described in detail below.

2.1 Data Collection

The response time behavior of running components within the application landscape is collected using the Kieker framework [Pro14]. Applications are instrumented using aspect-oriented programming. Using the adaptive monitoring feature [Pro14], data collection is activated for specific component operations. Monitoring records are passed by a new monitoring writer via a web-service call to a database application that serves as a backend for the approach presented in this work. This database application serves as a central repository for monitoring records. The web-service deserializes the record and stores it to a relational database. Monitoring records include response times of component operations and the current workload of the application in terms of queue length and resource utilization. The monitoring writer stores for each record an additional description of the deployment from which the measurement was obtained. This description includes information about the host and the application server instance. Information about the deployment is retrieved by the monitoring writer from local configuration files. The monitoring writer also stores meta-data of the application binaries provided by a central build tool so that performance measurements can be related to component versions.

2.2 Data Aggregation

The response time behavior of component operations can be collected from multiple application servers and for different workloads. Also, different versions of a component can be deployed within the application landscape. The aim of this step is to aggregate individual records to performance curves and stochastic expressions describing the response time behavior of operations in dependence on a set of input parameters such as workload characteristics. Aggregation is performed using statistical methods and measurements are aggregated for each component version separately. Java functionality is delivered as deployable units by assembling source code to Java Archives, Web Archives or Enterprise Archives. The current version of the code, identified by a revision number, is exported from a repository and assembled. The resulting archive is then deployed to an application server. However, neither the revision, the assembly nor the deployment suggest which component version is addressed by a performance measurement. The source code of a component might remain constant over multiple revisions, assemblies and deployments. The proposed approach stores for each component the revision numbers which contained changes to its source code. This revision number is then compared to the revision contained in the deployable unit. A central build tool stores the revision number of the assembled code in a property file within the deployable unit. Measurements can thus be assigned to specific component versions. The results of this step are provided over a web-service interface to clients (i.e., IDEs).

2.3 Model Generation

The source code of a new component and the performance data collected for any reused components are used to generate a component-based performance model. PCM is used as the meta-model for the generated models.

The main factors influencing the performance of reusable software components are the component implementation, required services, deployment platform, usage profile and resource contention [Koz10]. The focus of this approach are the component implementation and the required services. The component implementation is represented in terms of the control flow of individual component operations. Required services are modeled explicitly in the model as external calls. The deployment platform is assumed to be constant for all components. The usage profile is represented in terms of the response times of reused components in dependence on the workload. The impact of input parameters and resource contention are not addressed by the approach.

The proposed approach first generates a PCM repository model which specifies the currently investigated component. The model generation is based on the approaches of Kappler et al. [KKKR08] and Becker et al. [BHT$^+$10]. Source code of component operations is parsed and represented as an abstract syntax tree (AST). Each AST is then converted to a Resource Demanding Service Effect Specifications (RDSEFF). An RDSEFF specifies the behavior of a single component operation. Calls to reused components are modeled as external call actions. The response time behavior of external calls is modeled either using stochastic expressions or performance curves depending on whether the model is used as input for an analytical solver (stochastic expression) or a simulation engine (performance curve). This information is retrieved from the monitoring record database. Calls within the boundaries of the investigated component are modeled as internal action calls. Branches, loops and forks are also represented in the RDSEFF. Since the modeled component is under development, the probabilities of different execution flows are unknown. Therefore, branches are modeled having equal probabilities. Since binary code is not available for the investigated component, a dynamic analysis [KKKR08] of the component behavior can not be performed. Thus, resource demands of internal call actions are also not represented in the RDSEFF.

After creating the PCM repository model, the remaining PCM models that are required for a simulation, are also generated automatically. The PCM system model describes how components are combined and which interfaces they provide. The investigated component is represented as a single instance in the PCM system model. Each public component operation is modeled as an externally accessible interface. The PCM usage model describes a closed workload where all operations of the investigated component are called equally often. The hardware resources such as central processing units (CPU) available to the investigated component are specified within the PCM resource environment model. A resource environment model containing one server and a single CPU is generated. The investigated component is then mapped to this server within the PCM allocation model.

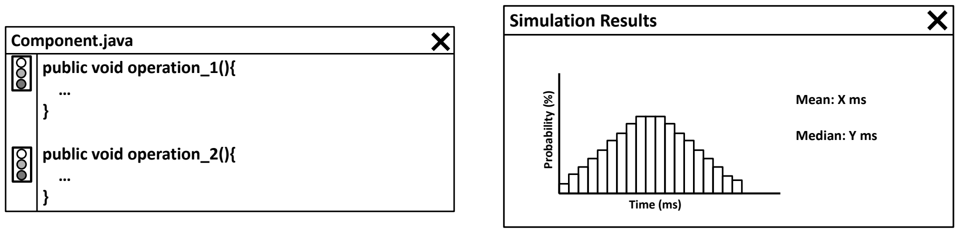
2.4 Developer Feedback

The main goal of the proposed approach is to integrate estimations of the expected response time of component operations in the development environment. Two types of feedback are envisioned:

1. An implicit estimation of the response time of component operations is automatically performed each time changes to the code are saved.
2. A detailed response time estimation of all component operations triggered by the developer.

Since changes to code are frequently saved, the immediate estimation must be executed very fast. The estimation is therefore performed using an analytical solver. Figure 2(a) shows how this type of feedback could look in the IDE. 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. Notifications are displayed as yellow and red traffic lights and are associated to the corresponding operation signatures.

The explicitly triggered response time estimation aims at providing more precise results. Therefore a simulator is used to process the PCM model. The response time behavior of reused components is modeled using performance curves. Figure 2(b) shows how this type of feedback could look in the IDE. Results are displayed as probability density function in



```
public void operation_1(){
  ...
}

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

(a) Immediate feedback mockup (b) Simulation results mockup

Figure 2: User interface mockups

a separate view. Important metrics such as mean and median values are also displayed.

3 Related Work

The presented approach is related to existing research on developer performance awareness and performance model generation. In the following, we discuss existing work that addresses these research areas.

Developer Feedback

Several approaches aim at providing feedback to developers. Weiss et al. [WWHM13] propose an approach for evaluating the performance of software artifacts based on tailored benchmarks applications during the implementation phase. The authors describe a scenario, how immediate feedback could be provided to developers in the IDE. Developers can either track the performance impact of changes to the implementation or compare different design alternatives. Performance estimations are displayed as numerical values and bar charts and in a separate view. The presented approach is applicable only to Java Persistence API services. Instructions on how to design benchmark applications and how to apply the approach to other components are provided.

Heger et al. [HHF13] present an approach to integrate performance regression root cause analysis into development environments. The change in performance between two revisions is displayed graphically as a function. Methods causing the regression are presented to the developer as a graph. The approach employs unit tests to gather performance measurements and thus provides no feedback on the performance expected in realistic environments.

Bureš et al. [BHK+14] propose the integration of performance awareness in the development life cycle of autonomic component ensembles. The authors describe how the design and operations phases can be augmented with performance-related activities such as formulating performance goals or collecting performance data. One of these activities aims at providing feedback to developers. The authors envision the presentation of performance measurements to the developer directly in the IDE. Feedback is provided graphically within a pop-up window. The approach is suitable for the presentation of historical data and doesn't support a real-time interaction with the developer.

Automatic Performance Model Generation

Brosig et al. [BHK11] 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. Call path tracing is employed to determine the control flow. Only executed paths are identified by the approach. Brunnert et al. [BVK13] present a similar approach that is applicable for all Java EE server products based on data collected from custom Servlet filters and EJB interceptors. Neither of the approaches support evaluating the performance of single components during the implementation phase.

Becker et al. [BHT$^+$10] present an approach for reverse engineering Java applications based on static source code analysis. Similar to the approach proposed by Kappler et al. [KKKR08], source code is parsed to an AST which is then converted to an RDSEFFs. Krogmann et al. [KKR10] use static and dynamic analysis to reverse engineer Java applications. This approach requires the evaluated components to be executed in testbeds. Parametric dependencies between input parameters and the control flow are derived using genetic search.

4 Conclusion and Future Work

This paper proposed an approach to provide feedback on the estimated response time of Java EE components to developers within the IDE. Using the 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. The approach focuses on the performance impact of component reuse and ignores the resource demand of the investigated component. Therefore, 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.

Future research will evaluate this approach within a case study. Additionally, the approach needs to enable the developer to refine generated PCM models, for example by specifying probabilities for branches within an RDSEFF. Refinements could be performed by annotating code.

References

[BHK11] F. Brosig, N. Huber, and S. Kounev. Automated extraction of architecture-level performance models of distributed component-based systems. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 183–192, Nov 2011.

[BHK$^+$14] Tomáš Bureš, Vojtěch Horký, Michał Kit, Lukáš Marek, and Petr Tůma. Towards Performance-Aware Engineering of Autonomic Component Ensembles. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014.

[BHT$^+$10] Steffen Becker, Michael Hauck, Mircea Trifu, Klaus Krogmann, and Jan Kofron. Reverse Engineering Component Models for Quality Predictions. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering, European Projects Track*, pages 199–202. IEEE, 2010.

[BVD$^+$14] Andreas Brunnert, Christian Vögele, Alexandru Danciu, Matthias Pfaff, Manuel Mayer, and Helmut Krcmar. Performance Management Work. *Business & Information Systems Engineering*, 6(3):177–179, 2014.

[BVK13] Andreas Brunnert, Christian Vögele, and Helmut Krcmar. Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications. In MariaSimonetta Balsamo, WilliamJ. Knottenbelt, and Andrea Marin, editors, *Computer Performance Engineering*, volume 8168 of *Lecture Notes in Computer Science*, pages 74–88. Springer Berlin Heidelberg, 2013.

[HHF13] Christoph Heger, Jens Happe, and Roozbeh Farahbod. Automated Root Cause Isolation of Performance Regressions During Software Development. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 27–38, New York, NY, USA, 2013. ACM.

[KKKR08] Thomas Kappler, Heiko Koziolek, Klaus Krogmann, and Ralf H. Reussner. Towards Automatic Construction of Reusable Prediction Models for Component-Based Performance Engineering. In *Software Engineering 2008*, volume 121 of *Lecture Notes in Informatics*, pages 140–154, Munich, Germany, February 18–22 2008. Bonner Köllen Verlag.

[KKR10] K. Krogmann, M. Kuperberg, and R. Reussner. Using Genetic Search for Reverse Engineering of Parametric Behavior Models for Performance Prediction. *IEEE Transactions on Software Engineering*, 36(6):865–877, Nov 2010.

[Koz10] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634 – 658, 2010. Special Issue on Software and Performance.

[LD13] Bill Shannon Linda DeMichiel. Java Platform, Enterprise Edition (Java EE) Specification, v7. 2013.

[Pro14] Kieker Project. Kieker User Guide. Research report, April 2014.

[T̊14] Petr Tůma. Performance Awareness: Keynote Abstract. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 135–136, New York, NY, USA, 2014. ACM.

[WHW12] Alexander Wert, Jens Happe, and Dennis Westermann. Integrating Software Performance Curves with the Palladio Component Model. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 283–286, New York, NY, USA, 2012. ACM.

[WWHM13] Christian Weiss, Dennis Westermann, Christoph Heger, and Martin Moser. Systematic Performance Evaluation Based on Tailored Benchmark Applications. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 411–420, New York, NY, USA, 2013. ACM.