Design Phase Analysis of Software Performance Using Aspect-Oriented Programming

Daesung Park and Sungwon Kang
Information and Communications University

Munji-ro 119, Yuseoung-gu,
Daejeon, 305-714, Korea

{grayger,kangsw}@icu.ac.kr

ABSTRACT

Apart from functionality, software system may have various non-functional requirements such as performance, security, reliability and schedulability. If we can predict how well the system will meet such requirements at an early phase of software development, we can significantly save the total development cost and time. Among non-functional requirements, performance is commonly required as the essential property of the system being developed. Therefore in the past, many analysis methods have been proposed but methods that can be practically performed in the design phase are rare. In this paper, we propose a simulation-based design-level performance analysis method based on aspect-oriented programming. We separate performance models from design models, and then inject performance requirements into skeleton code generated from design models. Our method has the advantages that (1) code for the simulation is generated automatically or semi-automatically, and (2) it is relatively easy to modify design models or performance models independently when they are changed.

Keywords

Performance analysis, Design stage software analysis, Aspect-oriented programming

1. INTRODUCTION

Software architecture and design are the artifacts that are produced early in the development process and reflect the early solution decisions for the given software requirements. They should be carefully designed because later discovery and fixing of problems would result in much higher cost of development. Performance is one of the most commonly required qualities and, in the past, various analysis methods were proposed for it but most of them have been rarely practiced in industry. It is because they require additional tasks such as modeling, implementation, evaluation, and so forth in order to perform analysis. We need more practical methods that reduce analysis overhead and can fill the gap between modeling and implementation.

This paper suggests a new method for the design phase performance analysis based on Aspect-Oriented Programming (AOP). AOP is a programming paradigm that realizes the principle of *separation of concerns* and lets the programmer focus on various aspects one at a time. When designing and implementing a software system, we usually modularize the system into small units such as objects, modules, procedures, and so forth. Decisions based on the *separation of concerns* principle tend to be confined to functionalities. Some non-functional aspects such as synchronization, component interaction, persistency or security control are hard to localize cleanly and hence generally are implemented as lines of code scattered over many software units. This is particularly so with the performance aspect. AOP helps modularization of software by allowing us to express various aspects independently from functionality and from other aspects. In particular, AOP enables us to model and analyze the performance aspect of a software system in the design phase as we will show in this paper.

The rest of the paper is organized as follows: Section 2 introduces current performance analysis methodologies, the UML performance profile and AOP. Section 3 presents our performance analysis approach from the aspect-oriented point of view and compares it with the traditional approach. In Section 4, we illustrate our approach by showing the modeling, implementation, simulation and resulting feedback of a case study application. Finally, Section 5 is the conclusion and outlines the future work.

2. RELATED WORKS

2.1 Software Performance Engineering

Software Performance Engineering (SPE) is a systematic, quantitative approach to develop a software system that meets performance requirements [3]. SPE pioneered model-based prediction of software performance. It evaluates the software system considering not only the relationship between tasks and resources but also resource requirements per each execution step of the whole system.

The SPE process [2] is well suited to Unified Process. For each use case, the key performance scenario, which is represented with augmented sequence diagrams, corresponds to a workload. The scenarios are represented by an execution graph. SPE uses two models: the software execution model and the system execution model. The software execution model represents key aspects of the software execution behavior. While the software execution model captures the resource requirements of the software alone, the system execution model is a more sophisticated model that captures workloads, multiple users or delays due to contention for resources. Like our approach to be presented later, SPE captures key performance scenarios and decomposes them into execution steps, and then describes the steps with appropriate performance models.

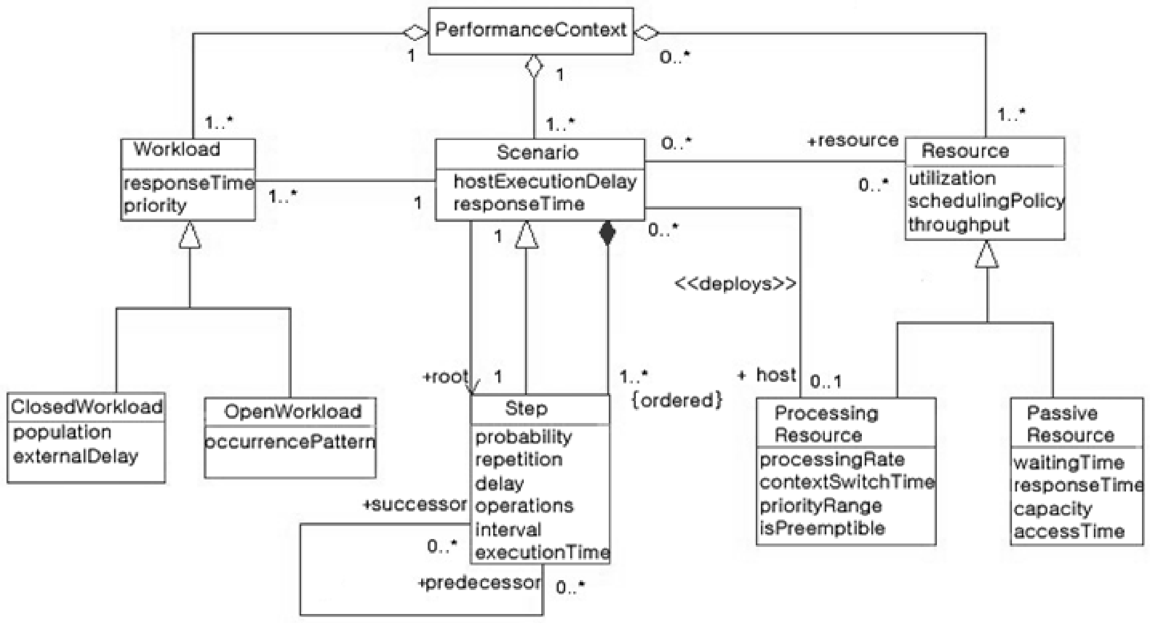


Figure 2-1. The performance analysis domain model [13]

2.2 Performance Analysis Techniques

In the early stage of development, for software performance analysis we have to use models that capture dynamic characteristics of the system because the phase has no concrete implementations. The paper [15] reviews various model-based software performance analysis methods in requirement specification, software architecture, and design phase. It classifies software performance analysis models into software dynamics specification models and performance models. Automata, process algebra, Petri-nets, message sequence charts, UML diagrams, and use case map are software dynamics specification models which describe the behavior of a software system. Markov processes, Queuing networks, stochastic timed Petri nets, stochastic process algebras, and simulation models are performance models which quantify the performance of system.

In our approach of this paper, we use annotated sequence diagrams for a software dynamics specification model and a simulation model for a performance model.

2.3 UML Performance Profile

The UML performance profile [13] specifies software execution parameters which can be used in performance analysis. Figure 2-1 shows main classes and their relationships in the performance analysis domain model. Main classes consist of workload, scenario, resource, step and so forth. The scenario defines system execution path, and has QoS requirements such as response time or throughput. The workload specifies the intensity of demand for the execution of a specific scenario. The resource is classified into a processing resource and a passive resource. The processing resource is a device (e.g., processor, interface device, storage device) that has processing steps. The passive resource is protected by an access mechanism such as a semaphore.

Even though the performance model based on class diagrams can specify performance attributes effectively, it addresses only static information about performance. AOP can capture dynamic aspects of a system using join point, pointcut and advice.

2.4 Aspect-Oriented Programming

In traditional programming paradigms, some design concerns such as synchronization, component interaction, persistency or security control tend to be expressed scattered and tangled across the system code. AOP [7] enables modular implementation of crosscutting concerns. By using it, software qualities including performance can be modularized as separate modules. We use AspectJ, a general-purpose AOP extension to Java, to build an executable for a simulation.

AOP

AOP enables us to decompose problems into not only functional components but also aspects which crosscut functional components, and then implement them by composing these components and aspects. In AOP, an aspect weaver is a code generator that is in charge of the composing. The paper [7] shows the benefits of AOP with the example of an image processor program. The paper finds the tradeoff between understandable code and optimized code in memory usage. By using AOP, they can achieve both understandability and efficiency in memory usage.

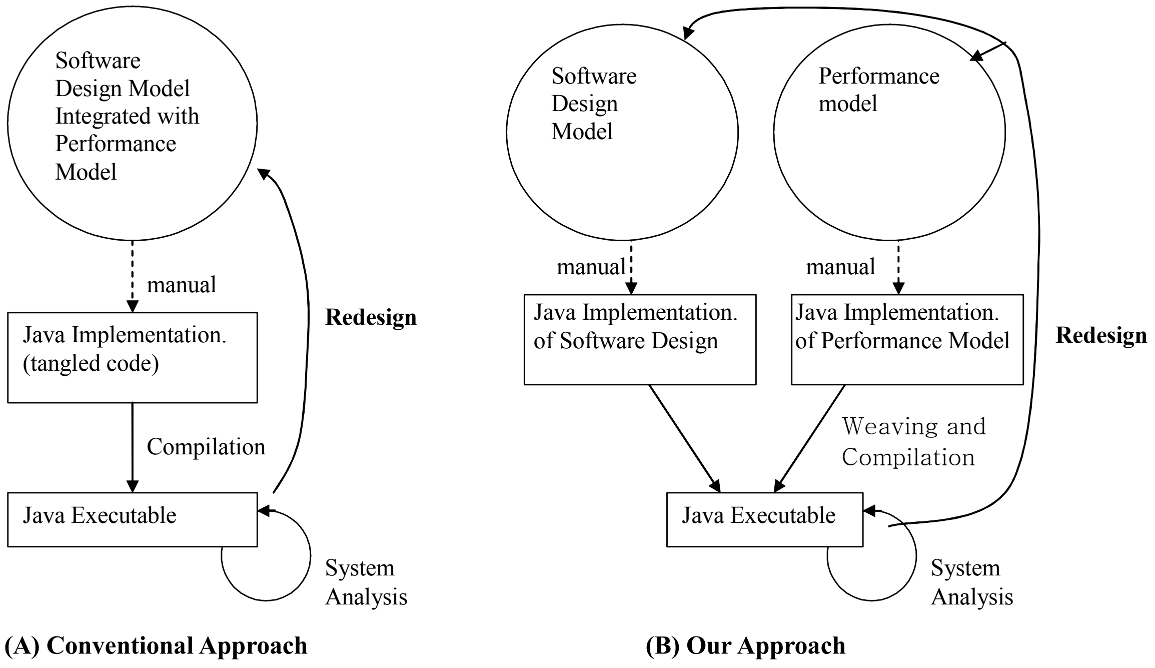
AspectJ

AspectJ [6] is a language that extends Java to support AOP. It has new concepts such as join point, pointcut, advice, and aspect. Join point is an identifiable point of program execution such as method/constructor calls, method/constructor execution, field get and set, exception handler execution, and static and dynamic initialization. Pointcut is a set of join points selected by a Boolean operation such as "and" or "or". Advice is used to define additional code to be executed before, after, or around join points. Aspect is a modular unit of crosscutting implementation. The

AspectJ compiler merges Java code with AspectJ code to achieve the weaving of crosscutting concerns. Current IDE tools that support AspectJ enable us to edit, compile, and debug Java code with AspectJ code.

Aspects for quality Attributes

Software qualities are desired attributes of software system such as performance, reliability, modifiability and reusability. Software requirement specifications describe different software qualities in different ways. In addition, software qualities have different models, tools, or metrics for analysis. For example, Markov model was used for reliability analysis, and Wright [14] was used for deadlock checking. However they were intermingled in the implementation phase because functionality was the only factor for modularization. Actually, it is difficult to set qualities apart from functionalities in the design phase. As the paper [12] indicates, systematic design methods and solution catalogues are not sufficient. While AOP is successful in modularizing code views with the help of tools such as AspectJ, researches on design modularization ([8], [9], [11], [12]) are on going. This paper handles how to modularize the performance concern in the design phase and how to implement it with AOP techniques.



Software Design Model Integrated with Performance Model
manual
Java Implementation. (tangled code)
Compilation
Java Executable
System Analysis
Redesign
(A) Conventional Approach
Software Design Model
Performance model
manual
manual
Java Implementation. of Software Design
Java Implementation. of Performance Model
Weaving and Compilation
Java Executable
System Analysis
Redesign
(B) Our Approach

Figure 3-1. Our approach compared with the conventional one

3. OUR APPROACH

In this section we explain our approach to a design phase analysis of software performance. Figure 3-1(A) shows the conventional approach. In Figure 3-1(A), software design includes both a functional design and a performance model and therefore code for the design model and code for the performance model get mixed together when they are implemented. It is assumed that we manually obtain code from the design model. Then the source code is compiled, executed and analyzed. If necessary, the system is redesigned and the whole process is repeated. The disadvantages of the conventional approaches are that (1) it is difficult to develop the design model because the designer should consider both the software design and the performance model at the same time, and (2) it is difficult to understand and maintain the program code because several concerns are intermingled in the implementation code.

Our approach in Figure 3-1(B) solves these problems. In the design phase, we clearly set the performance model apart from the design model and maintain the separation to the implementation phase. In other words, the design model has its own Java implementation code, and the performance model has its own AspectJ implementation. Later these two are woven together using the AspectJ compiler.

The approach of Figure 3-1(B) has many advantages over the conventional approach. Each of the design model and the performance model is transformed into the corresponding code clearly. Even though feedback may induce redesign of either the design model or the performance model, direct code generation speeds up the development cycle. Strict modularization in the implemented code enhances understandability and maintainability. We don't have to take the trouble to integrate the separated source code files for ourselves because it can be automatically done by the AspectJ compiler.

3.1 Design model

The UML [5] provides elaborate notations required for documenting requirement, architecture, design and so forth. Design of software system was usually modeled with class diagrams and sequence diagrams.

Performance analysis starts from constructing a design model based on system's functionalities. To make class diagrams, we extract main objects and their operations which are likely to affect performance critically. It is easy to generate program code from class diagrams. Several tools such as Rational Rose® can generate class, attribute, signature of operation, and relationship from class diagrams.

A sequence diagram can show the order of operations being executed. It is lacking in expressing timing information such as start time or end time of an operation and the duration of the operation. That means that as such it is not adequate for performance analysis. We show that aspect oriented mechanism can fill a gap in the sequence diagram by inserting time constraints to the section of AspectJ code that corresponds to operations of the sequence diagram.

3.2 Performance Model

A model should represent behaviors of system and performance requirements in static and dynamic ways, and provide appropriate format that can be easily utilized for the analysis phase[1].

We use XML, the flexible and extensible text format, to represent the performance model[2]. XML, originally designed to interchange documents over different application programs, is appropriate to be used in exchange performance-related data between modeling and simulation. XML can store not only content of data but also their structure. Moreover it is intuitive for a person to read and write data, and easy for computer to manipulate data (e.g., store, update, parse, or delete) in the programming by using DOM (Document Object Model) or SAX (Simple API for XML).

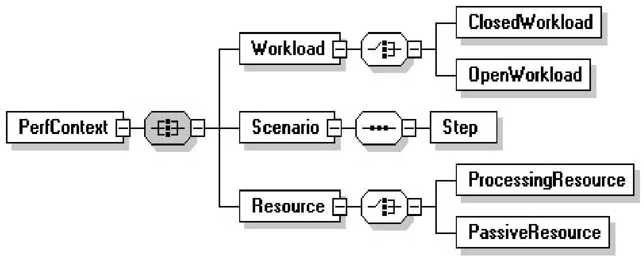


Figure 3-2. UML performance profile in XML format

For a simulation, we transform the performance model in Figure 2-1 to the XML format. Figure 3-2 shows the XML schema corresponding to the performance model. Each class in the model is mapped to an element in the XML schema and attributes of the class are mapped to the attributes of the element in XML. As *performance context* class in the model aggregates *workload*, *scenario*, and *resource* class, the *performance context* node in XML format has three child nodes: *workload, scenario,* and *resource*. The input of the simulation is a valid XML file containing actual performance parameters originating from the UML performance profile. The output of the simulation, response time or throughput, is also stored in the XML sharing the same schema. Table 3-1 explains performance parameters appeared to attributes of the XML nodes.

Table 3-1. Performance parameters [13]

Node	Attribute	Explanation
Workload	population	the size of the workload (number of system users).
Scenario	response time	the total time required to execute the scenario, including all resource waiting, synchronization delays and execution times.
Step	response time	the total delay to execute the step including all resource waiting and all execution times.
	interval	the time interval between successive repetitions of this step, when it is repeated within a scenario.
Resource	capacity	the number of permissible concurrent users.
	throughput	the rate at which the resource performs its function.

4. APPLICATION EXAMPLE

In this section, we demonstrate our approach using a case study, namely a map viewer system. The map viewer system is a web-based application that allows users to view a detailed map of a location. When users select the area that they want to see, the map viewer system finds the information of the area and then shows the map image on the web. The system consists of three tiers: *Presentation*, *Business logic*, and *Data manager*. The sequential steps of "showing map image" scenario are as below;

(1) *Presentation* tier gets input from users and requests a map image to *Business logic*.
(2) *Business logic* tier queries map data to *Data manager* tier.
(3) *Data manager* tier finds map data and returns it to *Business logic* tier.
(4) *Business logic* tier makes a map image from map data and returns it to *Presentation* tier.
(5) *Presentation* tier draws the map image and shows it to users.

For the sake of simplicity, this scenario considers neither branching nor alternative flow.

4.1 Functionality concern

As we mentioned, class diagrams and sequence diagrams describe the functionality of a system. Figure 4-1 shows objects and their operations, and Figure 4-2 shows the caller and the callee of operations and the sequence of these operations. These can be translated to Java code as in Table 4-1.

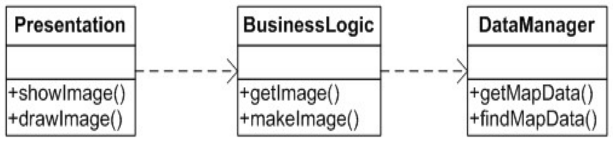


Figure 4-1. Class diagram

[1] In order to describe performance information for the analysis tool, the [1] introduces PMIF (Performance Model Interchange Format). However it is lacking in generality due to its special purpose.

[2] The papers [4] and [10] also try to map the UML performance profile to the XML format.

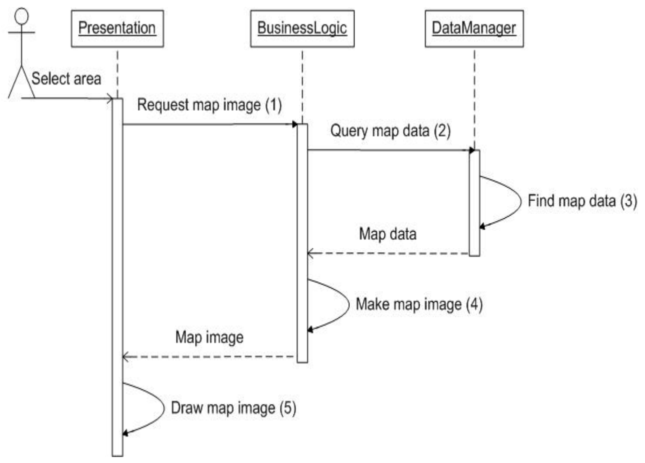


Figure 4-2. Sequence diagram

Table 4-1. Java code for functionality

```
 1:  public class Presentation {
 2:          BusinessLogic bl;
 3:          public void showImage( ) {
 4:                  bl.getImage( );
 5:                  drawImage();
 6:          }
 7:          public void drawImage( ) {
 8:          }
 9:  }
10:
11:  public class BusinessLogic {
12:          Presentation pre;
13:          DataManager dm;
14:          public void getImage() {
15:                  dm.getMapData( );
16:                  makeImage();
17:          }
18:          public void makeImage() {
19:          }
20:  }
21:
22:  public class DataManager {
23:          BusinessLogic bl;
24:          public void getMapData() {
25:                  findMapData();
26:          }
27:          public void findMapData() {
28:          }
29:  }
```

Except for the implementation lines of code (lines 4, 5, 15, 16, 25) of each operation, lines of skeleton code can be generated from class diagrams automatically [16]. The implementation of each operation is manually written from sequence diagrams straightforwardly. Later, the operations in classes will be the join points of AspectJ code reflecting performance characteristics and will be used to describe the behaviors of execution steps.

4.2 Performance concern

As we mentioned in Section 3, the performance concern is represented in the XML format that captures the UML performance profile. As the functionality concern is implemented in Java language, the performance concern is also implemented in Java language. We simulate dynamic behavior of the system with the executable compiled from functionality-based skeleton code and the performance model implementation.

The workloads are generated with Java threads. The number of threads is the same as the population of workloads, or *Workload.population* in Table 4-2. Each thread tries to obtain resources to execute the given operation. However, each resource has limited capacity, or *Resource.capacity*. When a thread representing a workload fails to obtain the resource, it waits for an instance and retries to do. *Step.interval* represents the interval between trials. If the thread succeeds in obtaining the resource, it executes the operation for *Step.responseTime*. The consumed time for execution is emulated using the *sleep()* method of *java.lang.Thread* class.

4.3 Weaving

The functionality concern and the performance concern are weaved by the AspectJ compiler. Figure 4-3 shows sequence diagrams overlaid with AspectJ elements. Lines of Code for performance analysis are inserted before or after appropriate pointcuts.

With regard to the UML performance profile, sequence diagrams represent the scenario. Objects in the sequence diagrams can represent resources required for workloads. Methods or messages in the sequence diagrams represent steps of the scenario.

4.4 Simulation

In this section, we show how to calculate performance metrics using the simulation and how to apply the AOP techniques for checking timing information of operations and counting the number of service completions.

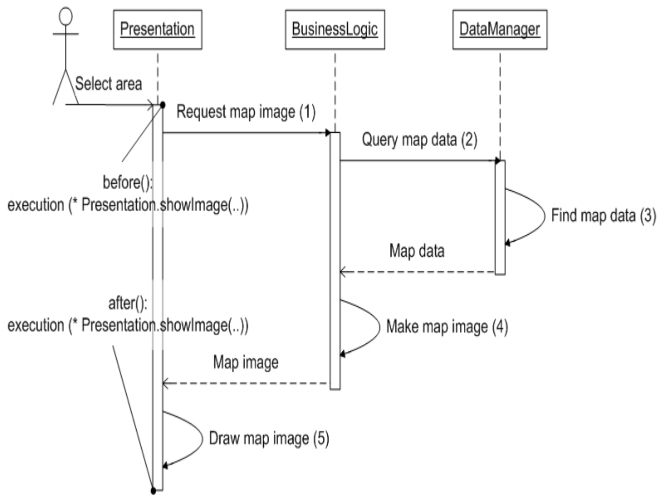


Figure 4-3. Sequence diagram with pointcuts and advices

Table 4-2. Input and output of the simulation

Input	Output
Step.responseTime	Resource.throughput
Step.interval	Scenario.responseTime
Resource.capacity	
Workload.population	

The purpose of the simulation is to get the response time of the scenario and the throughput of resource from performance parameters of step, resource, and workload. We can get the response time of scenario, or *Scenario.responseTime* using join points, pointcuts, and advices of AOP.

Table 4-3. Join points, pointcut, and advices

```
<J-1> Presentation.showImage() ;
<J-2> BusinessLogic.getImage();
<J-3> DataManager.getMapData() ;
<P-1> pointcut pShowImage() :
            execution(*  Presentation.showImage(..));
<A-1> before() : pShowImage();
<A-2> after() returning : pShowImage();
```

In Table 4-3, the lines <J-1>, <J-2>, <J-3> are the operations that appeared in the functionality implementation. The line <P-1> is the pointcut appeared in the performance implementation. When <J-1> calls <J-2>, <J-2> calls <J-3>. The response time of scenario is the time taken to execute <J-1>. To get the response time, time checking should be done in before/after the pointcut <P-1> corresponding to <J-1>. The advices <A-1> and <A-2> mean the very time before <P-1> is executed and the very time after <P-1> is returned respectively. Therefore, we can get the response time by subtracting the moment of <A-1> from the moment of <A-2> because the difference is the elapsed time executing the operation step.

To get the throughput of the resource, or *Resource.throughput*, we use a simple formula. Let T be the length of time in the observation period, and let C be total number of service completions in the observation period. Then the throughput of system is C divided by T. We can get T by observing simulation run time and can get C by setting counters before or after appropriate pointcuts.

4.5 Feedback

In performance analysis process, feedback means redesign of software architecture or reorganization of resource demands when analysis result does not meet requirements. The Feedback (1) of Figure 4-4 stands for modification of the software architecture or design. We can add new components for load balancing or can modify relationships between components.

The Feedback (2) of Figure 4-4 means modification of performance parameters. For example, when the result of performance analysis is worse than expected, we can increase the capacity of the resources. On the other hand, when the result is better than expected, we can increase the workload. That can be applied to enhance scalability of the system.

Initially, performance parameters used in simulation input are usually predicted values or assumed values. Through the feedback, simulation results can be used as input, therefore it makes the performance prediction more precise.

As Figure 4-4 shows, the Feedback (1) and the Feedback (2) are independent of each other because the model and implementation for the performance concern are separated from those for the functionality concern. Modification of the design or code for the functionality concern will hardly influence that for the performance concern, and vice versa. Therefore, our approach enables us to minimize the coupling between functionality-related modules (i.e., *Presentation.java, BusinessLogic.java,* and *DataManager.java*) and performance-related module (i.e., *Perf_Aspect.java*) in Figure 4-4.

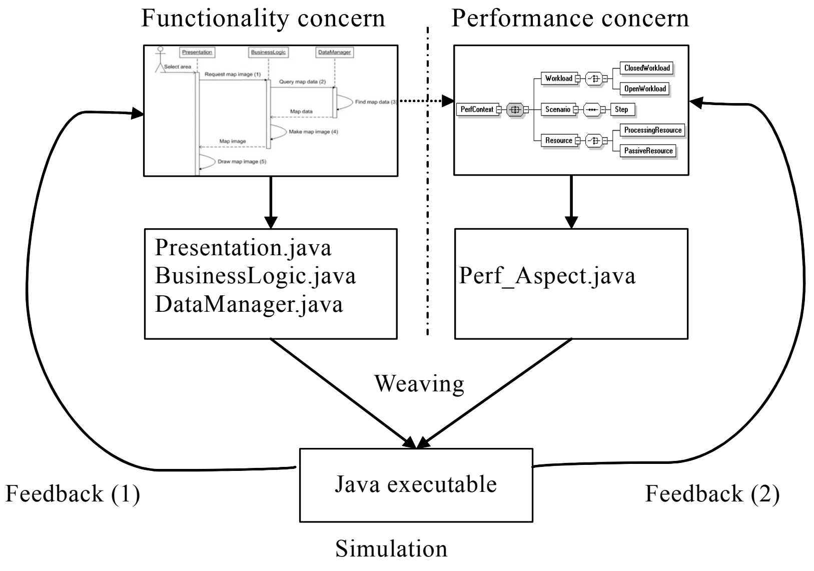


Figure 4-4. Feedback process applying the *separation of concerns* principal

5. CONCLUSIONS

We proposed a performance analysis method for the design phase using AOP. We also demonstrated the analysis process using an application example. The benefits of our method are as follows:

Firstly, it minimizes analysis overhead. Analysis using the simulation requires the working executable solely for the simulation purpose. Designing the model and building the executable from the model puts a lot of overhead on the software development process. We showed how we can separate the performance model from the design model. In our method, each isolated model can be easily transformed to the implementation in Java or AspectJ language and these two implementations are automatically integrated by the AspectJ compiler.

Secondly, it allows rapid feedback. Feedback can result in modification of design or performance requirements and even rewriting of the implementation code for the simulation. If we just have tangled code, we have to undergo the process of untangling, modifying, and re-tangling whenever a change is made regardless of how small the change may be. That makes feedback process slow and expensive. Well modularized code as our approach produces would enable rapid feedback.

Thirdly, it fits well into the modern software development process where design is often expressed predominantly with sequence diagrams and class diagrams, which are the basis for our analysis method. Our analysis method does not require learning complex, special purpose models, notations, or tools. Our choice, AspectJ, is the simple extension to Java and is not specific to any quality analysis.

In this paper, we showed that AOP can be applied to the simulation-based analysis in the design phase. We will expand the work to analysis of other software qualities such as reliability, security, deadlock freedom and so on.

6. REFERENCES

[1] C. U. Smith and L. Williams, “A Performance Model Interchange Format,” Journal of Systems and Software, Vol. 49, No. 1, 1999.

[2] C. U. Smith and L. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[3] C. U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley Longman Publishing Co., Inc., 1990.

[4] G. P. Gu and D. C. Petriu, “Early Evaluation of Software Performance based on the UML Performance Profile,” Proc. 2003 Conf. of the Centre for Advanced Studies Conf. on Collaborative research, Toronto, Ontario, Canada, pp. 66 – 79, 2003.

[5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language user guide*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 1999.

[6] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, “An Overview of AspectJ,” Proc. of the 15th European Conf. on Object-Oriented Programming, pp.327-353, June 18-22, 2001.

[7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, et. al., “Aspect-Oriented Programming,” Proc. ECOOP, Springer-Verlag, 1997.

[8] J. Gray, T. Bapty , S. Neema, D. C. Schmidt, A. Gokhale, and B. Natarajan, “An Approach for Supporting Apect-Oriented Domain Modeling,” Proc. 2nd Int’l Conf. on Generative programming and component engineering, pp.151-168, Erfurt, Germany, Sept. 22-25, 2003.

[9] J. Suzuki and Y. Yamamoto, “Extending UML with Aspects: Aspect Support in the Design Phase,” Proc. Workshop on OO Technology, pp.299-300, 1999.

[10] L. B. Arief and N. A. Speirs, “A UML Tool for an Automatic Generation of Simulation Programs,” Proc. 2nd Int,l Workshop on Software and Performance, pp.71-76, Ottawa, Ontario, Canada, Sept. 2000.

[11] M. Katara and S. Katz, “Architectural Views of Aspects,” Proc. 2nd Int’l Conf. on Aspect-oriented software development, pp.1-10, March 17-21, 2003

[12] N. Noda and T. Kishi, “On Aspect-Oriented Design: An Approach to Designing Quality Attributes”, APSEC 1999.

[13] Object Management Group, “UML Profile for Schedulability, Performance, and Time Specification,” OMG Adopted Specification Version 1.0, formal/03-09-01, Sept. 2003.

[14] R. Allen and D. Garlan, “A Formal Basis For Architectural Connection,”, A revised version of the paper that appeared in ACM Trans. on Software Engineering and Methodology, July 1997.

[15] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, “Model-based Performance Prediction in Software Development: A Survey,” IEEE Trans. SE, Vol.30, No.5, May 2004.

[16] W. Harrison, C. Barton, and M. Raghavachari, “Mapping UML Designs to Java,” Proc. Conf. on Object-Oriented programming, systems, languages, and applications, pp.178-187, Oct. 2000.