STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

A COMPARISON OF AOP BASED MONITORING TOOLS

GRIGORETA S. COJOCAR AND DAN COJOCAR

Abstract. The performance requirements of a software system are very important for the end user, especially today when everything gets faster and faster. In order to improve the performance of a software system, developers must first identify the parts that take a long time to execute. In this paper we describe how AOP is used for monitoring software systems performance and we present a comparison of the existing AOP based monitoring tools. The comparison is done using different criteria like: AOP extension used, UI support, performance metrics provided, etc.

1. Introduction

1.1. **AOP.** Separation of concerns is an important principle in software engineering [6]. It refers to the ability to identify, encapsulate and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose [12]. Even though separation of concerns seems simple, it is not. System concerns are of two types: core concerns that capture the central functionality of a module, and crosscutting concerns that capture system-level, peripheral requirements that cross multiple modules. Even though the most popular programming paradigms are good for designing and implementing core concerns, they do not provide the means of a clear separation for crosscutting concerns. From the various approaches that have proposed solutions for the design and implementation of crosscutting concerns [1, 5, 9, 11, 15], the aspect oriented programming (AOP) approach has known the greatest success both in industry and academia. AOP introduces four new notions in order to implement a crosscutting concern: *joinpoint*, *pointcut*, *advice*, and *aspect*.

- A *join point* is a well-defined point in the execution of a program. Join points consist of things like method calls, method executions, etc.

Received by the editors: April 10, 2011.

2000 *Mathematics Subject Classification.* 68M20, 68N19.

1998 *CR Categories and Descriptors.* D.1.m [**Programming Techniques**]: Miscellaneous – *Aspect oriented programming*; D.2.8 [**Software Engineering**]: Metrics – *Performance measures*.

Key words and phrases. aspect oriented programming, performance monitoring tools.

- A *pointcut* groups a set of join points, and exposes some of the values in the execution context of those join points.
- An *advice* is a piece of code that is executed at each join point in a pointcut. Usually, an AOP extension supports at least three types of advice: *before*, *around*, and *after*.
- An *aspect* is a crosscutting type that encapsulates pointcuts, advice, and static crosscutting features. An aspect is the modularization unit of AOP.

The aspects are integrated into the final system using a special tool called *weaver*. Nowadays, there are AOP extensions for well-known programming languages (eg. AspectJ for Java [2]) which are used in industry, too.

1.2. **Performance Analysis.** Performance of software systems is a very important topic. It refers to the response time or throughput as seen by the users of the software system. Many different things can impact the performance of a software system: network load, computation time, database query response time, etc.

There are two approaches to performance engineering: a *fix-it-later* approach and an *engineering* approach. The first approach advocates concentrating on correctness and deferring consideration of performance until the testing phase. The detected performance problems are then corrected by adding additional hardware, tuning the software, or both [3]. The second approach, called *software perfomance engineering* (SPE) [14], uses model predictions to evaluate trade-offs in software functions versus hardware costs.

Even though the SPE approach have obtained good results in developing software systems that meet their performance objectives from the beginning, this approach is not used very often. There are still software systems built without considering performance issues, and then, during testing, they are modified to meet the performance requirements. However, modifying the system to discover performance problems is a tedious task. The developers must modify different parts of the software system in order to identify those that cause performance problems.

The paper is structured as follows. In Section 2 we describe the AOP based approach for developing monitoring tools/frameworks and some of the freely available AOP based monitoring tools. A comparison of these tools is presented in Section 3. Conclusions and further work are given in Section 4.

2. The AOP Approach

The goal of AOP based monitoring tools is to develop an easy to use and easy to integrate tool in order to obtain performance data for different parts of a software system. Usually, when performance problems are encountered,

the developers must gathered different kinds of data in order to discover the parts of the software system that caused the problems. In order to do that, using object oriented programming for example, two steps must be performed:

(1) Some parts of the software system or all of it must be manually modified in order to gathered different kind of performance data, like the execution time of each unit of work (method, component, etc.).
(2) The gathered data must then be analyzed in order to discover the parts with performance problems.

If the second step can be automatically performed, the first one is a tedious one, that takes time to complete. Also, if new kind of data must be added for the analysis, all the previous modified parts of the system must be manually modified, again. This leads to decreased productivity, and lost time and effort.

The idea of the AOP based approach is to keep all the source code that gathers the data in one place, and then to integrate it with the software system whenever needed. Also, using AOP, the above described steps are merged. The analysis is performed while the data is gathered and when the execution of the system is finished, the results of the analysis are also ready.

The basic design of AOP based monitoring is as follows: pointcuts that match the chosen joint points are defined [4, 10]. Usually, methods executions or calls are chosen as joint points. Then, the around advice or before and after advices are written to update the performance data.

Some of the advantages of using AOP for monitoring software system performance are:

- The source code for the performance analysis (data gathering and analysis) is kept in one place: the performance analysis module (aspects). The analyzed software system has no reference to this module.
- The monitored system does not need to be modified in order to obtain performance data.
- The module can be easily plug-in and out of the system.
- The performance analysis aspects can be easily used for different software systems.

There are also some disadvantages of using AOP:

- The pointcut(s) definition is very dependent on the signature of the methods. If after a execution run, the signatures of some methods are changed, the developer must be careful to redefine the target pointcut, otherwise the methods may be missed in the following runs.
- In order to define the pointcut(s) the developer that uses the performance module must have an indepth knowledge of the software system. This may take time if he/she is not among the developers of the software system.

- This approach can be used only for programming languages for which an AOP extension exists.

2.1. **AOP based Monitoring Tools.** In the following we present some of the existing monitoring tools that use AOP.

InfraRED is a tool for monitoring performance of a J2EE application and diagnosing performance problems [8]. It collects metrics about various aspects of an application's performance and makes it available for quantitative analysis of the application. It uses AOP to weave the performance monitoring code into the application.

Glassbox is a troubleshooting agent for Java applications that automatically diagnoses common problems [7]. It offers ready to use aspects and tools to help developers get started with application-level monitoring and identify potential problems.

Perf4J is a toolset for calculating and displaying perfomance statistics for Java code [13]. It adds Java server-side code timing statements and it logs, analyzes, and monitors the results.

SpringSource AMS is designed to manage and monitor Spring-based applications, the Spring runtime, and a variety of platforms and application servers [16]. It focuses on application-level monitoring.

3. Monitoring Tools Comparison

In this section we present a brief comparison of the previously described tools. The comparison is performed using the following criteria: language dependecy, weaving approach, source code modification, extendability, computed metrics, and UI support.

Language dependecy. All the previously described AOP based monitoring tools can be used to monitor only Java based software systems. Also, all tools use AspectJ as the AOP extension for Java. However, some of them can be configured to use a different AOP extension, i.e., InfraRED with Aspectwerkz or JBoss AOP, SpringSource AMS with Spring AOP, etc.

Weaving approach. AspectJ supports three different approaches for weaving aspects into the software system:

- *Compile-time weaving (CTW)* where the AspectJ compiler takes as input the software system source code and the aspects source code and produces woven class files as output.
- *Binary weaving* is used to weave existing class files and JAR files.
- *Load-time weaving (LTW)* is binary weaving defered until the point that a class loader loads a class file and defines the class to the JVM.

All the previously presented monitoring tools support compile time weaving, and most of them also support load-time weaving. The latter one is preferred by the developers as it does not add extra step to the building process, it is easy to switch between the original version of the software system and the instrumented one, and the extra time required for load time weaving does not take too long.

Source code modification. Does the user of the tool need to modify the source code of the analyzed software system? Most tools do not require the manual modification of the source code. The only tool that requires source code modification is Perf4J because it uses AspectJ5 annotation facility. The *@Profiled* annotation provided by Perf4J is used to select method calls for monitoring. The annotation step must be performed only once, even if new data is required for performance analysis. However, if new methods are included in the analysis, they must be first annotated, and it is the user responsability to perform the annotation.

Extendability. How easy is for a developer/tool user to define/include new performance metrics using these tools? In order to define a new performance metric using the AOP approach, all the user has to do is to define a new aspect in which he specifies the methods to be monitored and how to compute the new metrics. As such, most tools allow the user to define new performance metrics. The exception is Perf4J, that uses AOP only for gathering performance data.

User Interface (UI) Support. All the tools provide a UI for viewing the performance metrics and for analysis. The UI provided is either a Web UI or a JMX client. Java Management Extensions (JMX) is a standard API for managing Java applications by viewing attributes of managed objects.

Performance metrics. All tools compute at least the following metrics: the number of times a method was executed in a run, and the average execution time for each monitored method. Some tools provide additional performance statistics like the mininum and maximum execution time, and the standard deviation (Perf4J), or the accumulated and the maximum execution time (Glassbox).

4. Conclusions and Further Work

We have presented in this paper the AOP approach for developing performance monitoring tools and a comparison of some AOP based performance monitoring tools. The comparison was performed using different criteria like language dependency, weaving approach, source code modification, and UI

support. In the future we intend to compare the AOP based performance monitoring tools using different case studies.

References

1. Mehmet Aksit, *On the design of the object oriented language sina*, Ph.D. thesis, Department of Computer Science, University of Twente, The Netherlands, 1989.
2. *AspectJ Project*, http://eclipse.org/aspectj/.
3. Ken Auer and Kent Beck, *Pattern languages of program design 2*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996, pp. 19–42.
4. Ron Bodkin, *AOP@Work: Performance Monitoring with AspectJ, Part 1 and Part 2*, http://www.ibm.com/developerworks/java/library/j-aopwork10/, http://www.ibm.com/developerworks/java/library/j-aopwork12/, 2005.
5. Krzysztof Czarnecki and Ulrich Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.
6. Edsger Wybe Dijkstra, *A Discipline of Programming*, Prentice Hall, 1976.
7. *Glassbox.com*, http://glassbox.sourceforge.net/glassbox/Home.html.
8. *InfraRED: Opensource J2EE Performance Monitoring Tool*, http://infrared.sourceforge.net/versions/latest/.
9. Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin, *Aspect-Oriented Programming*, Proceedings European Conference on Object-Oriented Programming, vol. LNCS 1241, Springer-Verlag, 1997, pp. 220–242.
10. Ramnivas Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming (2nd edition)*, Manning Publications Co., 2010.
11. Karl J. Lieberherr, *Component Enhancement: An Adaptive Reusability Mechanism for Groups of Collaborating Classes*, Information Processing '92, 12th World Computer Congress (Madrid, Spain) (J. van Leeuwen, ed.), Elsevier, 1992, pp. 179–185.
12. David L. Parnas, *On the Criteria To Be Used in Decomposing Systems into Modules*, Communications of the ACM **15** (1972), no. 12, 1053–1058.
13. *Perf4J home*, http://perf4j.codehaus.org/.
14. Connie U. Smith, *Performance engineering of software systems*, Addison-Wesley, 1990.
15. *Subject oriented programming*, http://www.research.ibm.com/sop/.
16. *Virtualized Java Application Monitoring*, http://www.springsource.com/products/systems-management, 2009.

(1) Babes-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogălniceanu str., Cluj-Napoca 400084, Romania

E-mail address: grigo@cs.ubbcluj.ro

E-mail address: dan@cs.ubbcluj.ro