Analysis of Monitoring Tools for Java Applications

Ramya Dharam [1], Sajjan G. Shiva [1]

Department of Computer Science

[1] University of Memphis, Memphis, TN, USA

Abstract. Runtime Monitoring is performed during the execution of software to detect anomalies in them. Currently several tools are available that help in developing the monitors. We analyze the prominent monitoring tools available for Java applications based on two features, the properties that can be monitored using these tools and the specification language used to specify the monitorable properties. The analysis performed will help the users and developers better evaluate the characteristics of different monitoring tools available in order to select the one suitable for their application.

Keywords: Runtime Monitors, Software Properties, Specification, Specification Language, Software.

1. Introduction

Many software systems have been deployed in critical areas like avionics, medical electronics, automobiles and for other activities like online banking, online shopping and social networking etc. In these areas we need the software systems to behave reliably i.e. produce expected results and should not reveal confidential information. Many a times due to internal faults and external intrusions these systems do not behave reliably i.e. deviate from their expected behavior, produce unexpected results and reveal confidential information. This creates the need for continuously monitoring a running system to detect the anomalies and to possibly prevent the systems from critical breakdowns.

A monitor is a software program that continuously observes the behavior of an executing target program and currently available tools help in the construction of monitors. These tools will take the specification of software properties to be monitored in the target program as input, and construct the monitors automatically.

The major contribution of this work is comparative analysis of prominent runtime monitoring tools available for Java applications. This analysis will help both developers and users to make a better choice of tools they would like to utilize for monitoring their applications based on their needs. Section 2 presents the comparative analysis of the monitoring tools. Section 3 discusses related work and conclusion is presented in Section 4.

2. Runtime Monitoring Tools

Different runtime monitoring tools available for applications developed using different programming languages is surveyed in [8], which distinguishes the tools based on the features like specification language, monitor type, event handler and the operational issues. Since today most of the applications are built using Java, in this section we present comparative analysis of the prominent tools available for monitoring Java applications. We analyze the tools to identify two main features: the properties that can be monitored in the application and the specification language used to specify the identified properties.

+ Corresponding author. Tel.: + (901-678-5465); fax: + (901-678-1506).
E-mail address: (rdharam@memphis.edu, sshiva@memphis.edu).

2.1. Jass

Java with assertions (Jass) [3] is a pre-compiler. It allows Java programs to be annotated with specifications using different kinds of assertions. Assertions are Boolean expressions written as comments into the Java code. Kinds of assertions that can be inserted into Java source code include method pre and post conditions, class invariants, loop invariants and variants, beginning/ending of method call and order of method executions. Jass then translates from the annotated program a pure Java program and the compliance of the program with its specification is dynamically tested during the runtime. Violation of the specification will be indicated by throwing a Java exception.

Jass allows the monitoring points to be manually identified by the user and specification at the identified points will be directly written into the program in the form of assertions. This avoids the usage of any external tool needed for identifying the monitoring points and instrumentation of the monitoring code with the source code. The syntax of assertions is close to programming language and easy for Java users thus, avoiding the need to learn a new specification language. Thus Jass tool is suitable for monitoring concurrent and reactive systems.

2.2. Java Mac

Java MaC [1] consists of two phases: the static phase and the runtime phase. The static phase is before the execution of the target program in which runtime components a filter, an event recognizer and a runtime checker are automatically generated from the formal requirement specification. During the runtime phase, information about the current execution is collected and checked against the formal requirement specification. Two separate languages are used to describe the requirement specification. The low-level and high-level requirement specifications are written in Primitive Event Definition Language (PEDL) and Meta Event Definition Language (MEDL) respectively. These two specifications help in separating low-level implementation specific details of monitoring from high-level requirements checking. The PEDL specification defines about the information that will be sent from the filter to the event recognizer and contains all the implementation specific details of monitoring. Execution points like beginning/endings of methods are declared to be monitored.

Java MaC provides a clear separation between monitoring implementation dependent low-level behavior and high-level behavior checking against the formal requirement specification. This characteristic of Java MaC distinguishes it from others. It supports automatic instrumentation of executable codes and real time systems built in Java can be monitored using this tool.

2.3. Java MOP

Java MOP [4] uses Monitoring Oriented Programming Framework. Specifications provided in separate files by users define different properties to be monitored like class invariants, interface constraints, method pre/post conditions and order of method calls. These specifications are compiled into AspectJ code consisting of AspectJ aspects. The generated AspectJ code is the monitoring code and is weaved into the program one wishes to monitor using any AspectJ compiler. Once weaved simply running the program as normal results in a monitored run of the program. Upon violations detection suitable actions like intimation, recovery and restoration are performed. Currently Java MOP tool supports only properties within the scope of the class; therefore each Java MOP specification file corresponds to a Java class containing all properties concerning that class. Each property to be monitored must be specified using any of the following specification languages: Extended Regular Expressions (ERE), Past Time Linear Temporal Logic (PTLTL) and Future Time Linear Temporal Logic.

Software projects whose requirements can be translated into monitorable properties can be used with Java MOP. The tool allows the usage of different specification language and integrates monitors at the program level.

2.4. AMNESIA

Analysis and Monitoring for Neutralizing SQL Injection Attacks (AMNESIA) [6] is a tool that detects and prevents SQL injection attacks in Java based Web applications. Combination of static analysis and run time monitoring technique is used by this tool which is composed of three modules namely Analysis module, Instrumentation module and Runtime Monitoring module.

The Analysis module takes Java Web Application as input and outputs a list of hotspots and a SQL Query model for each hotspot. Hotspots are the points in the application code that issue SQL queries to the underlying database. For each hotspot, an SQL-query model is built that represents all of the possible SQL queries that may be generated at that hotspot. The Instrumentation Module takes Java application and a list of hotspots as input and instruments each hotspot with lines of code to make a call to the runtime monitor before a call to the database. Third, the runtime monitoring module takes a query string and the ID of the hotspot that generated the query as input and retrieves the SQL query model for that hotspot. At runtime the application executes normally until it reaches a hotspot, at this point the monitor checks whether the query violates the hotspot's SQL query model by checking whether the model accepts the sequence of tokens in the query string. If the model does not accept the sequence of token, the query is identified as an attack and the query is then blocked and reported.

Monitoring Web applications for SQL injection attacks is the main focus of AMNESIA which distinguishes this tool from the rest.

2.5. LIME

The LIME [5] interface monitoring tool determine at runtime if software component violates the given specifications. Interface Specification Language (ISL) is used for specifying the interaction between the software components in a manner that can be monitored at runtime. The interactions are specified either by call specifications which specify the allowed call sequences to a Java object instance or by return specifications which specify the allowed behaviors of the Java object instance. Both the call and return specifications are expressed as Java annotations to Java interfaces and classes. The call and return specifications use atomic propositions to describe the expected properties of software interface components and are written in several different ways as past time LTL formulas, as future time LTL formulas, as regular expressions, and as non deterministic finite automata.

The LIME runtime monitoring tool translates the annotations in Java source file into deterministic finite state automata which functions as observers. The automaton is then translated into Java code and AspectJ will be used to weave the code into the original source code. This eventually produces an instrumented runtime environment where observers are executed at the time points. If a call specification is violated, the incorrect calling component is detected and if the return specification is violated the incorrect called component is detected.

The LIME runtime monitoring tool can be extensively used to monitor the behavior of Java interfaces. The LIME interface specification language allows the specifications to be written as annotations to Java interfaces.

2.6. LARVA

LARVA [7] performs runtime verification of properties of Java programs including real time properties. The five components present in LARVA consists of a system, a formal notated specification, a stream of events extracted from the system, a monitoring system which receives the events and verifies them according to the specification and a feedback loop. The input to LARVA is a Java program that is monitored and a LARVA script consisting of a set of specifications written using DATE specification language. LARVA compiler transforms the specification into the monitoring code that also contains a number of aspects that extracts the events from the system. The different properties that can be successfully monitored using LARVA include invariants, count the occurrence of number of events, duration of an event to occur i.e. real time.

A real-life financial system responsible for handling credit card transactions was successfully monitored using LARVA tool.

The analysis performed in this section is illustrated using a table described in Appendix A.

3. Related Work

Delgado et al., in [8] surveyed the prominent monitoring tools available by providing a taxonomy based on languages used to specify properties, types of monitors constructed, event handlers, and the implementation issues. They present only the class of properties which can be monitored using the monitoring tools as well as the specification paradigms for those classes of properties and give the general outline of the state-of-art technology of monitoring tools a decade ago. We in this work focus only on the monitoring tools currently available for systems built in Java. We specify both the classes and the typical properties that can be suitably monitored using each tool. The specification languages used to specify the properties are discussed and analyzed to provide an insight as to how the property specifications and the instrumentation points defined by the developer are implemented in constructing a monitor.

4. Conclusion

We provide a comparative analysis of the prominent monitoring tools available for Java applications. The analysis is performed based on two features i) the properties that can be monitored by the tool ii) the specification language used by the tool to specify the monitorable properties.

The analysis performed clearly illustrates that there exists different monitoring tools to monitor different software properties and choosing a tool depends on the user's needs. In this direction, the analysis performed in this paper will help a user understand the characteristics of different monitoring tools available for Java applications and better choose a monitoring tool suitable to their needs according to properties they intend to monitor in their application and ease of use of the specification language.

5. References

[1] Moonjoo Kim, Sampath Kannan, Insup Lee, Oleg Sokolsky, and Mahesh Vishwanathan, "Computational analysis of run-time monitoring- fundamentals of Java-Mac," in Elsevier Science B. V., Electronic Notes in Theoretical Computer Science 70 No. 4, 2002.

[2] A.K. Mok and G. Liu, "Efficient Runtime Monitoring of Timing Constraints", Proceedings of RTAS'97, June 9

[3] D. Bartetzko, "Jass-Java with Assertions", Proc. First Workshop Runtime Verification (RV '01), July 2001.

[4] F. Chen and G. Rosu, "Towards Monitoring-Oriented Programming: A Paradigm Combining Specification and Implementation", Electronic Notes in Theoretical Computer Science, vol.89, no. 2, 2003.

[5] K. Kahkonen, J. Lampinen, K. Heljanko and I. Niemela, "The Lime Interface Specification Language and Runtime Monitoring Tool", 9th International Workshop, RV 2009.

[6] William G.J. Halfond and Alessamdro Orso, "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks", ASE '05 Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, 2005

[7] C. Colombo, G. J. Pace, G. Schneider, "LARVA – Safer Monitoring of Real – Time Java Program", Seventh International Conference on Software Engineering and Formal Methods, 2009.

[8] Nelly Delgado, Ann Quiroz Gates, Steve Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," IEEE Transactions on Software Engineering, Vol. 30, No. 12, December 2004.

Appendix A

The table below consists of three columns namely: Tool, Specification Language, and Monitored Properties. Below are the definitions taken into consideration for completing the table:

A monitor is a system that observes the behavior of a system and determines if it is consistent with the given specification. A monitor takes an executing software system and a specification of software properties and checks that the execution meets the properties. [8]

Software properties are relations within and among states of computation. Software properties can be defined as a set of sequence of states. The two types of properties considered are Safety and Temporal properties. Safety properties include invariants, properties that check values of variables, properties that define sequence of events. Temporal properties include timing properties. [8]

Specification language is a language used to specify the properties associated with software behavior. [8]

Monitoring Tool	**Specification Language**	**Monitored Properties**										
		Global Primitive Variables	Local Primitive Variables	Class Invariants	Loop Variants and Invariants	Beginning/Ending of Method Calls	Order/Sequence of Method Calls	Method Pre/Post Conditions	Interface Constraints	Object Life Cycles	Realtime (Duration of an Event to occur)	Count of the occurrence of No. of Events	Hotspots (Points in Application code that issue SQL queries to the underlying database)
Jass	Java Assertions (Assertions are Boolean expressions.)			✓	✓	✓	✓	✓					
Java MaC	Primitive Event Definition Language (PEDL) Meta Event Definition Language (MEDL)	✓	✓			✓							
Java MOP	Future Time Linear Temporal Logic (FTLTL) Past Time Linear Temporal Logic (PTLTL) Extended Regular Expressions (ERE)			✓			✓	✓	✓				
AMNESIA	-												✓
LIME	LIME Interface Specification Language (ISL)								✓				
LARVA	DATE, LUSTRE									✓	✓	✓	