A Generic Software Monitoring Model and Features Analysis

Chang-Guo Guo
1. School of Computer Science, National University of Defense Technology, Changsha, China
2. China Electric Equipment and Systems Engineering Ltd., Beijing, China
Email: cgguo@163.net

Jun Zhu, Xiao-Ling Li
1. School of Computer Science, National University of Defense Technology, Changsha, China
Email: mail_zhujun@nudt.edu.cn, lxl_19851210@163.com

Abstract—**Software runtime monitoring has been used to increase the dependability of software. This paper focuses on software runtime monitoring techniques and tools. A generic software runtime monitoring model is presented, which consists of five basic elements, i.e., Monitored Object Features, Monitoring Access Methods, Execution Relationships, Runtime Monitor and Platform Dependencies. This model is an innovation in software monitoring fields. This paper gives some features of each element. Based on these features, researchers can use the model to comprehend and analyze runtime monitoring techniques and tools. The objective of this paper is to help researchers and users to identify the difference and the basic principles of software runtime monitoring techniques and tools. This paper also shows a result of relationship between techniques and features, through the result, we can understand the development trends of the techniques and tools, such as, what features are concerned more, and what features are concerned less.**

Index Terms—**software runtime monitoring, dependability, monitoring access methods, monitored object features, runtime monitor**

I. INTRODUCTION

Recent years, with the rapid development of information technology, software is increasingly presenting more and more important role in this information society. However, because of the growing scale and complexity of software and the growing dependence on software, software faults have a great impact on the information society.

On one hand, errors in safety-critical system have a huge impact. For example, the unsuccessful maiden launch of the Ariane-5 missile on July 1996 [1], the steep and off target landing of the Russian Soyuz-TMA1 April 5, 2003 [2], incorrectly aligning account numbers of A2LL software used by Germany's social services in 2004 [3] etc. On the other hand, apart from safety aspect, software errors can be very expensive. For instance, the error in Intel's Pentium floating-point division unit is estimated to have caused a loss about 500 million US dollars [4]. On June 28 2002, the National Institute of Standards and Technology of America published an inquiry report about software faults in which showed the average national loss of 59.5 billion dollars for the reason of software faults, equivalent to 0.6% of GDP in America [5]. In a word, constructing dependable software, this ensures that the software does what people expect it to do, is becoming an increasingly important activity.

Pointing to the safety of software, researchers have put forward a series of methods to solve this problem, such as model checking, software testing technology, and software monitoring [6]. However, model checking ordinary checks the model, not the software; even if the model is checked without problems, software faults may still exist in the software design and implementation phases [7]. Software testing technology may eliminates the software faults at a certain extent, but the test scenarios are limited, cannot test the software when it is running [8].Therefore, software monitoring is used more and more frequently to pledge the dependability of software, especially software runtime monitoring.

This paper summarizes common points and extracts common grounds in software runtime monitoring techniques, and presents a unified generic software runtime monitoring model, which comprises five basic elements. These elements are essential for constructing runtime monitoring systems. This paper also studies some features of the runtime monitoring model. Based on these features, this paper analyzes 40 existing monitoring techniques and tools. Through analyzing features of the techniques and tools, we can identify the difference among these different techniques and tools. This paper also shows a result of relationship between techniques and features, through the result, we can understand the development trends of the techniques and tools, such as, what features are concerned more, and what features are concerned less.

II. RELATED WORK

This work is supported by the National Natural Science Foundation of China under Grant No. 90818028, the National "Core electronic devices high-end general purpose chips and fundamental software" project under Grant No.2009ZX01043-002-004 and the National High-Tech Research and Development Plan of China under Grant No. 2007AA010301

© 2011 ACADEMY PUBLISHER
doi:10.4304/jsw.6.3.395-403

Nelly Delgado gives out a three components generic monitoring model, which contains software requirements, monitors, and event handlers [9]. Software requirements are implementation-independent descriptions of the external behavior of a computation. The definition of monitor which is widely accepted is, "A monitor is a system that observes the behavior of a system and determines if it is consistent with a given specification" [10]. Nelly Delgado considers that a monitor takes an execution trace and a software property specification and checks that the execution trace meets the property. The event-handler is the mechanism that captures and communicates the monitoring results to the system or user and possibly responds to a violation.

Nelly Delgado's monitoring model is applicable for most runtime monitoring tools and techniques, based on common elements of monitoring systems: specification language, monitor, and event-handler. However, with the increasing appearance of distributed computing systems, more and more application platforms are used to support these systems, for example, some systems are executing in certain operating system, some use middleware to exchange messages among different nodes, some must be applied in Virtual Machine, some use large database to store information, etc. Besides, the objective of monitoring is to guarantee software running as what people expect it to do, so if software running in some exceptions, monitoring techniques must have some measures to recover the software. Therefore, monitoring techniques must have the abilities to obtain the software runtime state information and do some response to the software. Previous monitoring models don't consider the platform dependencies and response mechanism between monitor and software.

This paper proposes another generic monitoring model, consider which not only comprises the components of previous monitoring model, but also comprises platform dependencies and response mechanism. Consequently, research this model is very significant.

III. Definitions and Illustrations

First of all, it is necessary to know what is software runtime monitoring. Various definitions for software runtime monitors exist. In paper [32], one of the most popular definitions is given out:"a monitor is a system that observes the behavior of a system and determines if it is consistent with a given specification". However, in our opinion the definition should contain five important aspects: (1). Observe the actual states and behaviors of executing software system, and express the monitoring information in a proper format; (2). Acquire the expected states and behaviors of executing software system; (3). Check the actual monitoring information with expected ones, and gain the analysis results, which can be used to analyze, diagnose and evaluate the healthiness of executing system; (4). Take specific measurements and control operations, in order to recover from errors; (5). The goal is to keep software system in correct states and to increase the dependability of software system.

For easy understanding of the definition, we raise several typical illustrations at first. Totally speaking, there are diverse approaches which can be used to implement software runtime monitoring. Software runtime monitoring can achieve the goal by creating an observer process to monitor the execution of software. During the execution of software, software can send state information to the observer process, and then the observer do necessary analysis on this information. There is another similar approach which uses event specifications. It associates state information with events in high abstract levels. One advantage of such an approach is to make modification of original source code as little as possible and try to decrease the impact on execution of software as much as possible. For example, BEE++ [17], DB-Rover [20], HiFi [27], Issos [29], Hy+ system [28] etc.

One approach of software runtime monitoring is to insert assertions to the program for the need of checking. These assertions will take effect during the execution of program. In usual ways, programmers write these assertions by hand which are developed together with programs, and insert them into the source code of programs. Assertions can also be written in the way of annotation of program, which will be automatically translated to constraints checking code at the point of annotation when compiling. For example, Design by Contract (DbC) [33] and Monitoring-Oriented Programming (MOP) [34].

There is another kind of software runtime monitoring, such as MaC (Monitoring and Checking) [31], JPaX (Java PathExplorer) [35] and so on. MaC and JPaX are two logic methods based monitoring tools, both of which generate monitoring systems from formal specifications. MaC uses a special interval temporal logic based language to specify the program behaviors, while JPaX supports just LTL. In order to send the application's states to the monitor, these systems need to instrument the Java bytecodes, which is hard to achieve in some other languages such as C++. Anyway, the examples mentioned above are very typical in software runtime monitoring. However, different software runtime monitoring techniques and tools exist in different forms, by employing different mechanisms. Thus, it is meaningful to extract a runtime monitoring model for comprehending and analyzing the related techniques and tools.

IV. Runtime Monitoring Model

Although software runtime monitoring techniques differ in thousands of ways, there is still something in common. This paper is trying to present a unified generic runtime monitoring model, which includes all common grounds of diverse techniques and tools. And then the basic elements of this model are described in details.

As Fig.1 shows, this software runtime monitoring model consists of five elements: Monitored Object, Monitoring Access Method, Execution Relationships, Runtime Monitor and Platform Dependencies. Besides these five elements, the Requirements and Evaluations are considered as the inputs and outputs of the monitoring

© 2011 ACADEMY PUBLISHER

model. So this paper doesn't consider them as element of the model. Requirements imply the demand of users, such as, the properties that users want to monitor. Evaluations imply the monitor information which can be deal with by user.

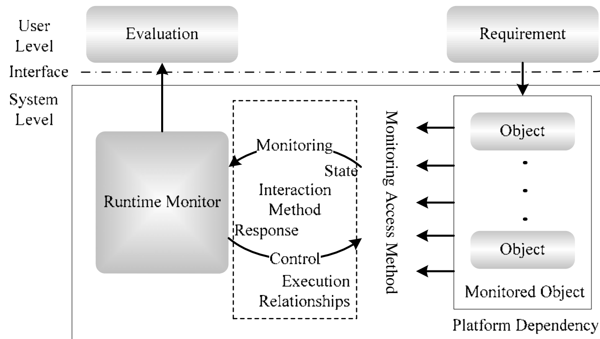

Figure 1. Software Runtime Monitoring Model

The five elements construct the Software Runtime Monitoring Model; each element is indispensable for the model. Function of each element is listed as follows.

- *Monitored Object*: Software entities which are focused on by users and need monitoring, such as, programs, components, service and distributed software systems. From the figure 1, we can know Monitored Object consists of one or more objects, this illuminates that the monitoring model can monitor distributed software system.
- *Monitoring Access Method*: Similar to a component interface between Runtime Monitor and Monitored Object. It consists of two aspects, On the one hand, Runtime Monitor invoke Monitoring Access Method to obtain the states and behaviors of executing programs or software systems. On the other hand, specific feedback or control operations would be sent back to monitored object.
- *Execution Relationships*: Execution relationship should tell us in which kind of interaction methods between Runtime Monitor and Monitored Object can be implemented. It also consists of two aspects, Runtime Monitor how to obtain the monitoring information and Runtime Monitor how to control the Monitored Object.
- *Runtime Monitor*: Takes responsibility of monitoring software objects and its process can be divided into three steps. Firstly, observing the states and behaviors of runtime software, and collecting the information of runtime software. Secondly, analyzing the collected monitoring information and checking the consistency between monitoring information and expected states of users, the judgments decide whether it is necessary to do some responds or control operations. Lastly, if the system appears some exceptions or failures, how to control the system in order to pledge the software recover to normal runtime states.
- *Platform Dependencies*: Some software, especially distributed software systems relies on some certain platforms. Such as, need certain operating system to support its running, running on Virtual Machines, delivers messages through Middleware, etc.

V. FEATURES OF THE RUNTIME MONITORING MODEL

As described above, Software Runtime Monitoring Model comprises five different elements, Monitored Object, Runtime Monitor, Monitor Access Method, Execution Relationships, and Platform Dependencies. In order to help researchers and users to identify the differences in software runtime monitoring techniques and tools, we need to define some features.

A. *Monitored Object Features*

There are so many features, which Monitored Object owns, such as performance, programming languages, architectures, platforms etc. However, this paper mainly focuses on distributed feature, which includes distributed software and non-distributed software.

Non-distributed. Some techniques and tools can only be used in monitoring non-distributed software systems. For example, Alamo [11], Annotation PreProcessor (APP) [13], DynaMICs [22], Java with Assertions (Jass) [36], Java PathExplorer [35], jMonitor [37], MOP [34] and so forth.

Distributed. The other techniques and tools can be used to monitor distributed software systems. BEE++ [17] is a typical example, that is used to dynamically analyze distributed programs. And it regards the execution of program as a stream of events. Meta monitoring system [38] is a collection of tools used for constructing distributed application management software in conjunction with a distributed toolkit. Meta enables management applications to observe and control functional behaviors of monitored programs.

With the birth of Web services and their compositions, some monitoring tools and techniques, which aim at Web services, come out. For example, Li et al. [39] proposed a Runtime Monitoring and Validation Framework for Web Service Interactions (called RMVF4WSI in this paper) . This framework can monitor the runtime interaction behaviors of Web service and validating the behavior against pre-defined interaction constraints. Qianxiang Wang et al. [40] introduced another Web Service Online Monitoring Framework (WSOMF) that collects quality sensitive events by multiple kinds of probes and agents. The framework, which focuses on quality of Web services, can also do some analysis according to the pre-specified constraints, so as to evaluate the quality of Web service. Paper [41] focuses on Web service compositions. It proposed a solution to monitor Web services implemented in BPEL, and devised an architecture that separates the business logic of a Web service from its monitoring functionality.

B. *Monitoring Access Methods Features*

Monitoring Access Method is presented in the following two aspects: Monitoring Code Instrument Methods (MCIM) and Response Mechanisms (RM).

© 2011 ACADEMY PUBLISHER

MCIM always through inserting aspect codes into the software to obtain the monitoring information. Due to the different strategies of inserting aspect codes, it consists of None (without inserting the code), Manual (insert by hand) and Automatic (inserting the code by specific tools). Automatic is also classified into two kinds, Dynamic and Static, which based on the state of the software when inserting the codes. If the software is running, it belongs to Dynamic, otherwise it belongs to Static.

None. Only a few runtime monitoring techniques and tools can acquire monitoring information without any instrument methods. Thus, in order to implement monitoring information collecting mechanisms, they rely upon some other specific tools. For example, a program debugger is used to obtain required monitoring information about the dynamic behaviors of actual software system. Meanwhile, interceptor mechanism, which has the capability of intercepting messages, can be used as monitoring probes. For example, JVMTI in Java Virtual Machine, Handler in AXIS, and Interceptor in CORBA etc. To monitor the behaviors of Web service, a SOAP Monitor utility in the Apache Axis1.2 toolkit is used to intercept the SOAP messages going in and out of the service, without requiring any special instrumentation on Web service itself, in RMVF4WSI [39].

Manual. Manual instrumentation is the easiest method in common use, which depends on programmers who manually instrument monitoring code into programs or software which needs monitoring. This method has many drawbacks, such as low effectiveness, high randomness and so on. For example, the Annalyzer [42] uses ANNotated Ada to specify properties as annotations, which are instrumented by hands by programmers. In addition, early MOP [34] technique and DbC [33] belong to this type.

Automatic(Static). Before execution of programs and software, monitoring code can be automatically instrumented without any intervention of programmers. Once program is compiled and executed, the source code cannot be inserted or changed again. A part of techniques and tools utilize AOP (Aspect-Oriented Programming) as the instrument methods, such as J-LO [30].

Automatic(Dynamic). During runtime and execution, monitoring code can still be instrumented into software without any interrupt on execution. This type of methods can be categorized to runtime instrumentation, interpreter instrumentation, instrumenting compilers, and virtual machine (VM) etc. Runtime instrumentation refers to the modification of the monitored program code immediately prior to or during execution. For example, Valgrind [43] is a framework for dynamic binary instrumentation, which can be used to build dynamic binary analysis tools including runtime monitors.

RM reflects the way how Runtime Monitor affects the states and behaviors of Monitor Object, especially when a violation or exceptions happens. It consists of None, User Customized, and Automated. None, means this techniques and tools cannot change the states and behaviors of software. User Customized, the states and behaviors are changed by users. Automated, the monitor cannot only analyze the states and behaviors of software, but also can control and change the states and behaviors.

None. Although these techniques or tools can not affect the execution of software apart from execution effectiveness, they have many ways to help users analyze the states of monitored objects by warning reports, trace records etc. Post-mortem analysis is such a typical approach. In post-mortem analysis, sequences of states (execution trace) from a particular execution of software are examined and stored. After the program completes, the execution trace will be analyzed. The advantages are that performance degradation is minimized, and some temporal constraints can be verified at any state in the execution trace. The disadvantage is that failures are not prevented from occurring.

User Customized. On one hand, as have been mentioned above, the monitoring techniques which adopt DbC approach [33], such as JML [23], Jass [36] and so on, can define exception handling mechanism by users. In addition, MOFRM [44] is an extension of DbC approach. On the other hand, user-customized method can also be implemented as a program debugging systems, in which breakpoints can be set and users can interaction with the executing software, so as to ensure normal executing state of the system.

Automated. In order to fully control the execution of a program, part of runtime monitoring techniques and tools are endowed with automated control and steering mechanisms. For example, exception handling, rollback, recovery operations or termination and so forth can be used as automated response mechanisms. Falcon provides online monitoring and steering of distributed software. Its steering system help users to implement online control software systems. Each steering server can read monitoring events and then decide what actions to take [25]. Issos System provides a lot of hooks for action specifications, which can be used for exception handling [29].

C. Execution Relationships Features

The execution relationship Features consists of two parts: interaction methods (IM) and monitoring execution models (MEM).

The interaction method defines the interaction or communication ways between runtime monitor and monitored object. It shows how runtime monitor and monitored object can get in touch with each other. It should be said that the selection of interaction methods depends on the execution models to some extent. For example, if single-process model is employed, monitoring information can be acquired through utilizing procedures or function libraries. However, when monitor is a separate process, it may need the support of IPC techniques, or other methods. According to different techniques, interaction mechanisms can be classified into the following groups:

Shared Variables. Monitoring systems, which adopt single-process execution model, usually depend upon shared variables. Because the runtime monitoring information can be obtained by directly accessing shared

© 2011 ACADEMY PUBLISHER

variables. In addition, different threads in the same process share the same execution space. As a result, thread model can get monitoring information through shared variables. For example, Alamo owns an access library that allows monitors to directly manipulate target program states [11].

InterProcess Communication. Monitoring systems using multi-process execution model usually utilize InterProcess Communication mechanism to realize the communication needs between monitor and monitored object. IPC consists of pipe, semaphore, shared memory, socket and so on. For example, in JPaX, checking module can execute on different machines, and concerned monitoring events can be transmitted by socket interface [35]. Meta monitoring system [38] makes use of the ISIS distributed toolkit to construct distributed application management software.

Middleware. Strictly speaking, middleware is one of IPC mechanisms. However, in this paper, middleware is excluded from IPC, for its particularity. Middleware can be utilized as an interaction method. Programmers do not need to pay any attention to the distributed features of monitored objects, no matter whether monitor and monitored object locate in the same machine or in a distributed environment. This implementation can greatly decrease workload and take communication and execution effectiveness of monitoring system into consideration. Paper [45] describes another typical example, called Model-based Runtime Monitoring of End-to-End Deadlines (RMoEED), in which RT CORBA is applied for interaction mechanism.

At first, monitoring process is implemented by using procedures or function libraries. In this way, runtime monitor and monitored software are in the same process. And then, runtime monitor is peeled off from process of monitored object, forming a separate observer or monitor process. After thread techniques appear, runtime monitor can exist as a thread, which can greatly decrease the system expenditure of running runtime monitor. Among the possible relationships between monitor and the program being monitored, three are commonly used: one-process model, multi-process model, and thread model.

Single-Process Model. Runtime monitor and monitored object are in the same process space. Monitor can use procedure library to get the monitoring information from monitored object. A monitor is a library of procedures linked to the program being monitored or integrated into the runtime system. The one-process model has good performance and access characteristics, but it does not prevent the target program and monitor code from affecting each other in critical ways. In addition, the control flow logic within the monitor is somewhat inverted, since the monitor is activated through callbacks. For example, Annalyzer [42], APP [13], Jass [36], JML [23], jMonitor [37] etc.

Multi-Process Model. In the multi-process model, the monitor is a separate process from the program being monitored, reducing the problem of intrusion at the expense of complicating monitor access and reducing performance. The communication methods between monitor and program are usually implemented through IPC. One of the design goals in BEE++ is the support of dynamic program analysis for distributed heterogeneous target applications at runtime. The design is based on a symmetric peer-peer architecture [17]. Falcon [25], JPaX [35], Issos [29], Meta [38], Fabio Barbon et al.'s monitoring system [16], ComPol [19], GAMMA system [26], WSOMF [40] and so on use multi-process model too.

Thread Model. In the thread model, the monitor is a separate thread in a shared address space occupied by the program and possibly other monitors, providing a reasonable compromise between the characteristics of the one-process and multi-process models for many monitoring applications. For example, Alamo adopts a model called coroutine, which is exact thread model in fact [11]. Alamo provides an execution model in which a target program and the execution monitor are coroutines executing within a single address space. And the context switches within a single address space are lightweight. With the development of multi-core techniques in micro-electronics, more and more computers will use multi-core CPU, and the runtime monitoring techniques, which adopt thread model, will be much more popular.

D. Runtime Monitor Features

This paper mainly pays close attention to the method how to implement the monitoring mechanism, includes Algebra, Automata, Logic, Policy Rule, and Statistics. Due to article space reasons, the concrete realization of these methods is not detailed here.

Algebra. Algebra is utilized as their basic mechanism of runtime monitors by Jass [36], JPaX [35], MOP [34] and some other monitoring techniques.

Automata. MOP can use a standard CFGto-pushdown-automata algorithm which will be implemented as an MOP logic-plugin. Therefore, MOP can also support CFG specifications that cannot be expressed using parametric extended regular expressions or temporal logics [34]. Meanwhile, ComPol is based on specifications, and its specifications are expressed in communicating finite state automata (FSA) based formalism [19]. And so is RMVF4WSI [39], which adopts FSA as the representation methods of interaction constraints, and executes automatic consistency checking of interaction against these constraints.

Logic. By counting the number of monitoring techniques which adopt logic as their runtime monitoring mechanism implementation, it is concluded that logic is the most popular method in this research field. For example, Evolvable System employs a revision-based first order logical framework as its monitoring and evolution mechanism [24]. EAGLE is a rule-based framework, which is capable of defining and implementing finite trace monitoring logics, including future and past time temporal logic, extended regular expressions, real-time and metric temporal logics (MTL), internal logics, linear temporal logic and so on [12]. J-LO also specifies its properties in linear-time temporal logic [30]. MOP [34], DB-Rover [20], DynaMICs [22], JPaX [35], jMonitor [37].

© 2011 ACADEMY PUBLISHER

Policy Rule. In Meta monitoring system, programmers write a set of policy rules in data model to specify the desired system behaviors, by using a language called Lomita. Therefore, the programmers may make direct calls to sensors, actuators or other functions defined in data model [38]. Control flow checking (CFC) unit is one of basic blocks of EASIS. CFC utilized a lookup table, which stored all the possible predecessor/successor relationships of the monitored components, to compare real executed successors with the possible successor set of the predecessors.

Statistics. One of the typical techniques which employ statistics as its monitoring mechanism implementations is Artemis [15] that is a practical runtime monitoring mechanism for execution anomalies. The Artemis framework can guide baseline monitoring techniques toward regions of the program where bugs are likely to occur, yielding a low asymptotic monitoring overhead. Artemis also facilitates system-load aware runtime monitoring that allows the monitoring coverage to be dynamically scaled. Argus [14] is another typical example, which is an online statistical bug detection and monitoring tool. Argus that constructs statistics at runtime using a sliding window over the program execution, is capable of detecting bugs in a single execution and can raise an alert at runtime when bug symptoms occur. SOBER [46] is another statistical model-based approach, which can localize software bugs without any prior knowledge of program semantics, by modeling evaluation patterns of predicates in both correct and incorrect runs respectively.

E. Platform Dependencies Features

Platform dependencies mean that the execution of monitoring system must rely on the support of some specific platforms, in order to implement software runtime monitoring. It contains two meanings: on one hand, the implementation of software monitoring function itself needs platform. On the other hand, the whole monitoring system execute on the platform, which provides system with runtime supports. There are four categories: operating system, virtual machine, database, and middleware.

Operating System. Most of runtime monitoring techniques cannot execute without the support from operating system. A performance and reliability monitor is integrated into the operating system of Microsoft Vista. It can collect runtime information, which will be simply provided to users in a specific and legible format. This tool can help user monitor memory access, disk request, CPU time, running process and other related information, which is embedded in Windows operating system. By fully surveying, every runtime monitoring technique and tool needs the support of OS platform.

Virtual Machine. In Section 4.2, it is mentioned that virtual machine can be used to monitor the program which is running on this VM. Besides, some instrument methods also can utilize Java byte instrumentation in Java virtual machine. For example, JPaX can instruments Java byte code to transmit a stream of relevant events to the observation module which can perform logic-based monitoring [35]. Moreover, jMonitor [37], J-LO [30], EAGLE [12] etc. also depend upon VM.

Database. In DB-Rover, it is possible to capture monitoring information data in a database. These data can be used for analysis at a later time. So DB-Rover relies very much on the platform of database [20]. In addition, the Hy+ system [28] and GAMMA system [26] are very similar to the database approach in which monitoring information is stored in a database and manipulated after collection.

Middleware. Middleware can be used to shield from the differences of platforms. As for RMoEED [45], the authors create an infrastructure for validation based on RT CORBA. So it can provide a distributed monitoring facility, which can observe interaction deadlines. In [39], a runtime validation tool of RMVF4WSI is based on CORBA. Moreover, there are more typical examples, such as WSOMF [40].

Fig. 2 shows the features which are considered in this paper of Software Runtime Monitoring.

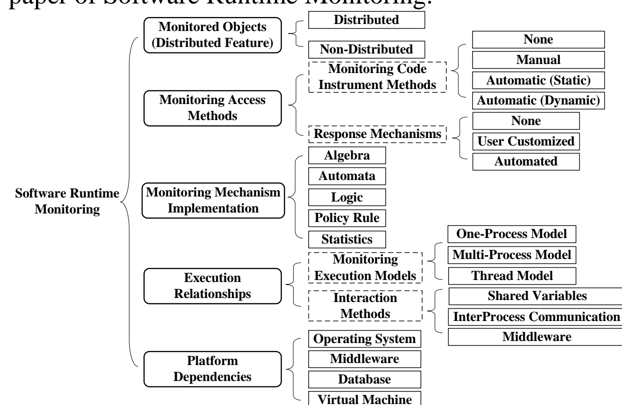

Figure 2. Features of Software Runtime Monitoring

VI. A CASE STUDY

This section analyzes the features of Software Runtime Monitoring through 40 monitoring techniques and tools. This case would show the features of some techniques and tools. Results would help us realize the characteristics of the modern monitoring techniques and tools.

The 40 techniques and tools are, Alamo[11], EAGLE [12], Jass[36], APP[13], Meta[38], Argus[14], MOP[34], Artemis[15], Barbon[16], RMVF4WSI[39], BEE++[17], CBI[18], ComPol[19], SOBER[46], JPaX[35], DB-Rover[20], Deep Space[21], DynaMICs[22], JML[23], Evolvable System[24], Falcon[25], GAMMA[26], HiFi[27], Hy+[28], Issos[29], J-LO[30], EASIS, jMonitor/jContractor[37], MaC[31], MOFRM[44], RMoEED[45], WSOMF[40].TABLE I shows the features of some common techniques and tools, such as Argus, ComPol, GAMMA, MaC, and JPaX. From this table, we can know clarity that, the types of software that the techniques and tools can monitor. For example, Argus can monitor the software that has these features, including Non-Distributed, Dynamic, Shared Variables, and Statistics.

TABLE I. Typical Examples of Software Monitoring Techniques and Tools

Name	*Monitor Object Features*	*Monitoring Access Methods*		*Execution Relationships*		*Runtime Monitor Features*	*Platform Dependency*
		MCIM	*RM*	*MEM*	*IM*		
Alamo	Non-Distributed	Static	User Customized	Thread	Shared Variables	Other	None
Argus	Non-Distributed	Dynamic	None	Single-Process	Shared Variables	Statistics	None
Artemis	Non-Distributed	Dynamic	None	Single-Process	Shared Variables	Statistics	None
ComPol	Distributed	Static	None	Multi-Process	IPC	Automata	None
CBI	Non-Distributed	Manual	None	Single-Process	Shared Variables	Statistics	None
EAGLE	Non-Distributed	Manual	None	Single-Process/Thread	Shared Variables	Logic	VM
HiFi	Distributed	Static	User Customized	Multi-Process	IPC	Logic	None
Issos	Distributed	Manual	User Customized	Multi-Process	IPC	Automata	None
GAMMA	Non-Distributed	Dynamic	User Customized	Multi-Process	IPC	Logic	Database
MaC	Non-Distributed	Static	User Customized	Single-Process	Shared Variables	Logic	VM
Meta	Distributed	Manual	User Customized	Multi-Process	IPC	Policy	None
MOFRM	Non-Distributed	Static	User Customized	Multi-Process	IPC	Logic/ Statistics	None
jMonitor	Non-Distributed	Dynamic	None	Single-Process	Shared Variables	Logic	VM
JPaX	Non-Distributed	Dynamic	None	Multi-Process	Shared Variables	Algebra	VM
RMoEED	Distributed	Static	None	Multi-Process	IPC	Automata	Middleware
WSOMF	Distributed	Manual	None	Multi-Process	IPC	Other	Middleware

As Fig.3 shows, Consider 15 different features, x-coordinate represents different features, and y-coordinate represents the number of techniques and tools that possess the relevant feature. From Fig.3, we can know something about the existing runtime monitoring techniques and tools. For example, we can know the number which support distributed software is 17, and the number which support non-distributed software is 27. Thereby, the techniques and tools support non-distributed software is more than that support distributed software.

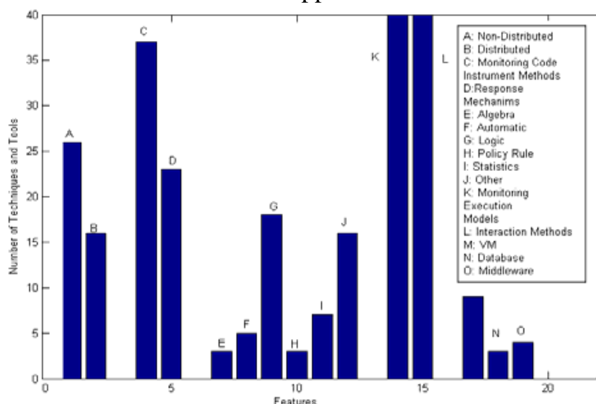

Figure 3. Relations between Features and Techniques

VII. CONCLUSIONS

Owing to the increases in scale and complexity of software, software systems become much more difficult to ensure the constant correctness in long time running. As a result, software runtime monitoring has been entered the vision of researchers again. It becomes very meaningful to propose a generic monitoring model for software runtime monitoring techniques and tools. This paper can help us to lay solid foundation for future research work.

In conclusion, this paper completes the following contributions: proposes a unified generic runtime monitoring model, which can be used to comprehend monitoring systems and their principles by researchers, through analyzing related techniques and tools; according to the five elements of runtime monitoring model; gives out the application scopes of the runtime monitoring techniques and tools, which can clearly tell the differences between these techniques and tools; discusses the obstacles and prospects of runtime monitoring as open issues, so as to indicate the developing trends and prompt the development of the research field.

ACKNOWLEDGMENT

The authors wish to thank their past and present colleagues, Tao Wang, Yue-Peng Yin etc., who contributed to the research described in this paper. They are also very thankful to their current research sponsors.

REFERENCES

[1] Inquiry Board, "Ariane 5 Flight 105 Inquiry Board Report," tech. rep., European Space Agency Press, July 1996.

[2] "http://space.kursknet.ru/cosmos/english/machines/stma1.sht."

[3] W. Hasselbring and R. Reussner, "Toward Trustworthy Software Systems," Computer, vol. 39, no. 4, pp. 91–92, 2006.

[4] "EX-13: Annual or Quarterly Report to Security Holders," tech. rep., Intel Corp, 1996.

[5] NIST, "Software Errors Cost U.S. Economy $59.5 Billion Annually: NIST Assesses Technical Needs of Industry to Improve Software-Testing," 2002.

[6] H.wang CHEN, J.WANG, W. Dong. High Confidence Software Engineering Technologies. ACTA ELECTRONICA SINICA, 31(12A), December 2003.

[7] M. Kim, S. Kannan, I. Lee, O. Sokolsky. Java-MaC: A run-time assurance tool for Java. In First International Workshop on Run-time Verification. Paris, France. 2001.

[8] I. Lee, H. Ben-Abdallah. A Monitoring and Checking Framework for Run-Time Correctness Assurance. Proc. 1998 Korea-U.S. Technical Conf. Strategic Technologies, 1998.

[9] N. Delgado, A. Q. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," IEEE Transactions on Software Engineering, vol. 30, Decemer 2004.

[10] D. Peters, "Automated Testing of Real-Time Systems," Technical Report, Memorial University of Newfoundland, Nov. 1999.

[11] C. Jeffery, W. Zhou, K. Templer, and M. Brazell, "A Lightweight Architecture for Program Execution Monitoring," ACM SIGPLAN Notices, vol. 33, no. 7, pp. 67–74, 1998.

© 2011 ACADEMY PUBLISHER

[12] H. Barringer, A. Goldberg, K. Havelund, and K. Sen, "Program Monitoring with LTL in EAGLE," in Proceedings of 18th International Conference on Parallel and Distributed Processing Symposium, 2004.

[13] D. S. Rosenblum, "A Practical Approach to Programming with Assertions," IEEE Transactions on Software Engineering, 1995.

[14] L. Fei, K. Lee, F. Li, and S. P. Midkiff, "Argus: Online Statistical Bug Detection," in Proceedings of Fundamental Approaches to Software Engineering 2006 (FASE'06), pp. 308–323, Springer-Verlag, 2006.

[15] L. Fei and S. P. Midkiff, "Artemis: Practical Runtime Monitoring of Applications for Execution Anomalies," in The ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation (PLDI'06), (Ottawa, Ontario, Canada), June 2006.

[16] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes ofWeb Service Compositions," in IEEE International Conference on Web Services (ICWS'06), 2006.

[17] B. Bruegge, T. Gottschalk, and B. Luo, "A Framework for Dynamic Program Analyzers," ACM SIGPLAN Notices, vol. 28, no. 10, pp. 65–82, 1993.

[18] B. Liblit, M. Naik, A. Zheng, A. Aiken, and M. Jordan, "Scalable Statistical Bug Isolation," in Proceedings of the 2005 ACM SIGPLAN conference on Programming Language Design and Implementation, pp. 15–26, ACM Press New York, NY, USA, 2005.

[19] M. Zulkernine and R. E. Seviora, "A Compositional Approach to Monitoring Distributed Systems," in Proceedings of the International Conference on Dependable Systems and Networks (DSN'02), 2002.

[20] M. Diaz, G. Juanole, and J. Courtiat, "Observer– A Concept for Formal On-Line Validation of Distributed Systems," IEEE Transactions on Software Engineering, vol. 20, no. 12, pp. 900–913, 1994.

[21] M. James and L. Dubon, "An Autonomous Diagnostic and Prognostic Monitoring System for NASA's Deep Space Network," 2000.

[22] A. Q. Gates and P. J. Teller, "DynaMICs: An Automated and Independent Software-Fault Detection Approach," in Proceedings of the Fourth International High-Assurance Systems Engineering Symposium, 1999.

[23] G. T. Leavens, A. L. Baker, and C. Ruby, "Preliminary Design of JML: A Behavioral Interface Specification Language for Java," in ACM SIGSOFT Software Engineering Notes, vol. 31, March 2006.

[24] H. Barringer and D. Rydeheard, "Modelling Evolvable Systems: A Temporal Logic View," in We will Show them: Essays in Honour of Dov Gabbay, vol. 2, pp. 195–228, College Publications, 2005.

[25] W. Gu, G. Eisenhauer, K. Schwan, and J. Vetter, "Falcon: On-Line Monitoring for Steering Parallel Programs," Concurrency: Pract. Exper, vol. 10, no. 9, pp. 699–736, 1998.

[26] J. Bowring, A. Orso, and M. Harrold, "Monitoring Deployed Software Using Software Tomography," in Proceedings of the 2002 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering (PASTE'02), pp. 2–9, 2002.

[27] E. S. Al-Shaer, Hierarchical Filtering-Based Monitoring Architecture For Large-Scale Distributed Systems. PhD thesis, Old Dominion University, 1998.

[28] M. Consens, M. Hasan, and A. Mendelzon, "Using Hy+ for Network Management and Distributed Debugging," in Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering, vol. 1, pp. 450–471, IBM Press, 1993.

[29] D. Olge, K. Schwan, and R. Snodgrass, "Application-Dependent Dynamic Monitoring of Distributed Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 4, pp. 593–622, 1989.

[30] E. Bodden, "J-LO A Tool for Runtime-Checking Temporal Assertions," Master's thesis, RWTH Aachen University, 2005.

[31] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan, "Java-MaC: A Run-time Assurance Tool for Java Programs," Electronic Notes in Theoretical Computer Science, vol. 55, no. 2, pp. 218–235, 2001.

[32] D. Peters, "Automated Testing of Real-Time Systems," tech. rep., Memorial University of Newfoundland, November 1999.

[33] B. Meyer, Object Oriented Software Construction. Prentice-Hall, 1998.

[34] F. Chen and G. Rosu, "MOP: An Efficient and Generic Runtime Verification Framework," in the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'07), 2007.

[35] K. Havelund and G. Rosu, "Java PathExplorer – A Runtime Verification Tool," in the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: A New Space Odyssey, pp. 18–21, June 2001.

[36] D. Bartetzko, C. Fischer, M. M¨oller, and H. Wehrheim, "Jass – Java with Assertions," Electronic Notes in Theoretical Computer Science, vol. 55, no. 2, pp. 103–117, 2001.

[37] M. Karaorman and J. Freeman, "jMonitor: Java Runtime Event Specification and Monitoring Library," in Runtime Verification 2004 (RV'04), 2004.

[38] K. Marzullo, R. Cooper, M. D. Wood, and K. P. Birman, "Tools for Distributed Application Management," IEEE Computer, vol. 24, pp. 42–51, August 1991.

[39] Z. Li, Y. Jin, and J. Han, "A Runtime Monitoring and Validation Framework for Web Service Interactions," in Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06), vol. 6, pp. 70–79, 2006.

[40] Q.Wang, Y. Liu, M. Li, and H. Mei, "An Online Monitoring Approach for Web services," in Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC'07), vol. 1, pp. 335–342, IEEE Computer Society Washington, DC, USA, 2007.

[41] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes ofWeb Service Compositions," in IEEE International Conference on Web Services (ICWS'06), 2006.

[42] D. Luckham, S. Sankar, and S. Takahashi, "Two-Dimensional Pinpointing: Debugging with Formal Specifications," IEEE Software, vol. 8, no. 1, pp. 74–84, 1991.

[43] N. Nethercote and J. Seward, "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation," in The ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI'07), (San Diego, California, USA), June 2007.

[44] K. Chan, I. Poernomo, H. Schmidt, and J. Jayaputera, "A Model-Oriented Framework for Runtime Monitoring of Nonfunctional Properties," in the 1st International Conference on the Quality Of Software Architectures (QOSA'05), 2005.

[45] J. Ahluwalia, I. H. Kr¨uger, W. Phillips, and M. Meisinger, "Model-Based Run-Time Monitoring of End-to-End Deadlines," in International Conference on Embedded Software 2005 (EMSOFT'05), (Jersey City, New Jersey, USA), September 2005.

[46] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff, "SOBER: Statistical Model-based Bug Localization," SIGSOFT Software Engineering Notes, vol. 30, no. 5, pp. 286–295, 2005.

© 2011 ACADEMY PUBLISHER

Chang-Guo Guo was born in 1973. He is an associate professor in School of Computer Science, at the National University of Defense Technology, Changsha, China. Prof. Guo received his Ph.D. degree in computer science at National University of Defense Technology in 2002. He has taken part in several research projects of the National High Technology Research and Development 863 Program of China and the National Natural Science Foundation of China. Till now, he has published more than 15 papers. His current research interests include software engineering, dependable software, real-time systems and distributed computing.

Jun Zhu was born in 1981. He received his M.S. degree in computer science in 2006. He is currently a Ph.D. candidate of computer science at the National University of Defense Technology. His research focuses on software engineering, dependable software and distributed computing.

Xiao-Ling Li was born in 1985. He is currently a Ph.D. candidate in school of Computer Science, at the National University of Defense Technology. His current research interests include software engineering, dependable software, distributed computing, distributed constraint optimization problem and artificial intelligence.

© 2011 ACADEMY PUBLISHER