An Online Monitoring Approach for Web services

Qianxiang Wang, Yonggang Liu, Min Li, Hong Mei
School of Electronics Engineering and Computer Science, Peking University
Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
Beijing, China, 100871
{wqx, liuyg05, limin05, meih}@sei.pku.edu.cn

Abstract
High quality is one of the critical elements contributing to Web service’s success. Monitoring events that are sensitive to quality of Web services is thus an important issue for Web services. This paper proposes an online monitoring approach for web service. This approach is driven by a quality model which covers five kinds of events: the response to client, application execution, changing states of resources, client request, and management operations. We introduce a monitoring framework that collects quality sensitive events by multiple kinds of probes and agents. The framework can do some analysis according to the pre-specified constraints, so as to evaluate the quality of web service. The initial implementation and experiment with a web-based auction example shows that our approach is feasible, and the monitoring cost is affordable.

1. Introduction
Web services are internet-based software applications published using standard interface description languages and universally available via XML based communication protocols. This technology is widely expected to enable the interoperability of heterogeneous systems and reuse of distributed functions. Meanwhile, the shift of industry towards this technology has been extremely slow [1]. Some research work reveals that the lack of quality assurance, such as security, transaction is the primary reason [2] hindering the shift.

Most of the traditional approaches towards software quality assurance and improvement focus on developing phase. For example, process management, formal verification and software testing are all efficient approaches of improving software quality. For Web services, these approaches are still useful. Meanwhile, web services are provided as integration of many different kinds of components: software, database, hardware, and network, etc. So service quality is related with not only software, but also all other related components. For example, limited hardware resources tend to cause a long response time, too many client requests may bring about low performance, and intrusive request may even lead to the denial of service. These challenges cannot be anticipated completely at the developing phase. Therefore these threats make online quality assurance of Web services a significant and tough issue.

1.1. Problem
Many traditional technologies such as fault tolerance, transaction, security attraction, can be considered as online approaches to enhance quality. In recent years, more general online techniques and concepts have been proposed, e.g. autonomous computing [3], self adaptive software [4]. As for all these approaches, traditional or unconventional, monitoring is always the first step to their solutions. Other actions such as making decisions, taking actions, are all based on events being monitored.

Monitoring technology is widely used in many kinds of software. Starting from early 1960s with the advent of debuggers, it was used for debugging and testing, correctness checking, security, dependability, performance enhancement, performance evaluation, and controlling [5]. Recent taxonomy work shows runtime software monitoring has been used for profiling, performance analysis, software optimization as well as software fault-detection, diagnosis and recovery [6].

Web services monitoring is taken in parallel with the normal executing process without interrupting it. The early research work on web service monitoring was
reported in [7], following the work of requirements monitoring in dynamic environments [8]. After that, some other research groups reported their work, with different motivations and frameworks [9][10][11][12].

Many researchers have noticed the invasion problem of monitoring. Someone called it Heisenberg Effect [5], or Uncertain Principle, which was borrowed from physical science. For the software based monitoring, we emphasize two meanings of uncertain phenomena: (1) the monitor must consume some resource, such as CPU time, memory; (2) the potential defect of monitoring system brings new risk to the target system.

Besides the invasion issue, there are many other issues still need to be further explored. Here we list some of them which this paper is strongly interested in: (1) which kind of events are sensitive to web service quality, and thus need to be monitored? (2) What are the useful underlying mechanisms for event monitoring? (3) How to collect, analyze, and handle these events? Some existed works have touched part of these problems. This paper tries to explore these issues further.

1.2. Approach Overview

In this paper, we propose a new approach for Web services online monitoring. Quality related events are usually difficult to monitor, because they are usually scattered in the whole service system, and thus difficult to collect. This paper copes with this problem by introducing a quality model for monitoring, which covers multiple kinds of events.

Compared with the existing web service online monitoring work, our approach has the following features: (1) it is driven by a quality model which provides a solid foundation for the framework; (2) it emphasizes the low-level event extracting mechanism; (3) it provides a guideline of how to build the monitoring system.

The rest of the paper is organized as follows: Section 2 introduces the quality model for monitoring. Section 3 describes the proposed monitoring framework in detail. Section 4 lists some activities of how to use the proposed framework. Section 5 introduces the initial implementation of monitoring framework, and some experiments which show the cost of monitoring system. Section 6 provides some related works. In the end, section 7 concludes the paper and lists some of the future work.

2. Quality Model for Monitoring

When the system is providing service, various events are happening (request, response, method invocation, etc.), and various states are under changing. In this paper, we view state changing as one special kind of event, so that the monitoring mechanisms can be managed uniformly at a higher level.

For all events to be monitored, we need to know firstly which can be used to evaluate service quality, and which are useful to enhance quality of service? To understand these events systematically, we divide them into different classes, based on the analysis of their affection to the quality of service. We used the Chinese Stone Mill\(^1\) model (Mill model for short) to represent graphically the different kinds of event that are quality sensitive and thus need to be monitored. See Figure 1.

The proposed Mill model is composed of five parts. We list them according to their effect degree on quality of service: (1) response messages; (2) application; (3) resource; (4) request messages; and (5) management operations. Although these five parts are shown separately, they are actually closely related. For example, lacking of resource usually leads to a bad response time; malicious requests may lead to the failure of application; code defect of memory leak may result in quickly consuming of memory.

![Figure 1. Mill model for web service monitoring](image)

2.1. Response Message

Response message related events are usually used to evaluate different attributes of service quality. There exists some research work on attributes of service quality [13][14], from the view of service selection, service composition and service management, etc. Based on their work, this paper separates the attributes of service quality into two classes: basic service quality attributes and high-level quality attributes.

\(^1\) Chinese stone mill is a kind of mill that has two stone pies with the same vertical axis, and one handle that drive the upper stone. We use it to emphasize the meaning of decoupling the general quality evaluating issue into the detailed events monitoring issue.
Basic service quality attributes are those that can be monitored directly by sniffing response messages. They are also attributes from the view of the client. In other words, single client can “feel” these attributes directly. The typical basic service quality attributes are: availability, correctness and efficiency.

Availability: Can client get service result, no matter it is correct or wrong? This attribute is application independent, and usually easy to monitor.

Correctness: Can client get the correct service result? This attribute relies on return values in response message. This attribute is application specific, including parameter value error, system-level exception or application-level exception.

Efficiency: Can client get the right service result under some constraint, such as in limited interval, expected precision? The typical constraint should be the response time: the interval between request message and response message. Many web service QoS research works focus on this issue [13][14].

From these basic attributes, we can derive many kinds of high-level quality attributes. High level quality attributes are used to evaluate system in general. e.g.: mean time of denial of service, mean time of service with wrong result, mean response time, response count in certain interval, right response count in certain interval.

2.2. Application Execution

Application itself has the most important effect to service quality. For example, wrong business logic, memory leak, deadlock, data race, and inconsistent data, are all typical reasons that may lead the service into an error state, or even stop of servicing.

Application related events occur inside the application, such as method executions, changing of variable values. Some application itself has some instrumented code which is designed to throw out some important events. Some applications may provide reflective interface which can be used to expose some important internal states, such as instance number. If application itself doesn’t provide such capability, then monitoring these events relies totally on system software. Some middleware based applications use container mechanism to control the execution of application modules and monitor inter-component message event.

2.3. Resource State Changing

Running of application depends on different kinds of low level resources (CPU, memory, file system, etc). Thus the state changing of those resources affects the service quality, especially the performance of service.

The typical resource state changing includes: CPU usage, memory usage, opened file number, etc. Monitoring of such state changes depends strongly on APIs that operating system provides.

There are also some other resource state changes that beyond operating system. For example, bandwidth is a special resource for service providing, and aliveness of computing node is a more general changing state that affects service availability.

2.4. Request Message

Web services are request driven systems. Different with traditional software that often provides service for only one client, Web services usually provide services for multiple clients, even up to thousands of clients at the same time. Thus the request messages from one client may have effect on response messages to other clients. Client connection number and client request frequency, all have effects on response time to the client. Some application dependent events such as imprecise request and intrusive request may also lead to low efficiency or stop of the web service.

Request message and response message are two most important events to be monitored in the proposed framework. Monitoring of request message focuses on method to invoke, parameter value, and role of the client. Service maintainer can specify constraint on these factors, so as to verify whether the request message is acceptable.

2.5. Management Operation

Management operations belong to application provided interfaces which are used to improve service quality: re-configuration of services, update application, add resource, block request message, etc. These operations are taken by service administrator, and need to be monitored also, so as to evaluate whether those operations are correct and effective.

3. Architecture of Monitoring Framework

While monitoring model aims to answer the question of “what should be monitored”, monitoring framework tries to answer the question of “how to monitor”? We can divide this question further into a series of sub-questions: “What are useful low-level mechanisms for online web service monitoring”? “How
to analyze the monitored events"? “How to help the administrator to make adapting decision”? Monitoring framework depends strongly on the runtime environment. One feature of the proposed approach is that it is middleware based. That means, many of the mechanism implementations are dependent on middleware. Actually, the framework itself can be considered as one kind of special middleware.

Our framework is composed of distributed probes, agent, central analyzer and high-level representation. Figure 2 lists the key components of the framework. In this figure, the monitored objects are grayed, so as to make the framework clearer. Another grayed component is service constraint description. It is used to analyze the event by some probes and central analyzer.

3.1. Supporting Probes

As the key component of the monitoring framework, probe is used to extract interesting events. There are different mechanisms to extract events. These mechanisms can be divided generally from two dimensions: instrument based versus interceptor based, in-line based versus out-line based [6] [18].

Instrument mechanism is widely used in program testing. In this approach, the monitoring code is embedded inside the target code module. This is the most widely used monitoring mechanism so far. Traditional instrumented code is inserted manually by the programmers. The most important feature of the instrument based approach is the freeness: the code can be inserted freely into any location of the monitored code. In recent years, some new instrument mechanisms (such as Javaassist, AspectJ) are developed, in order to instrument code automatically according to the configuration information.

Interceptor mechanism is widely used in middleware. Interceptor can get details of all messages to and from the monitored code, thus can be used naturally as probes. Interceptor in CORBA, Handler in AXIS, and JVMTI in JVM are all well known interceptors that can do some processing to messages. The most important feature of interceptor is the flexibility: it keeps independent both at coding time and at runtime. Interceptor executes in the same process with the monitored code.

Besides the difference in ways of event extracting, probes are also different in analyzing capability. Many probes just capture the event and send it to special analyzing component (central analyzer in this paper). However, some probes can do some analyzing, especially for those constraints that are easy to be checked, e.g., parameter value and response time. For those probes that have analyzing ability, it needn’t forward all the messages to the central analyzer. However, it still needs to send message to central analyzer when some deviation or error occurs.

The proposed framework can thus support the following four kinds of probes. See Figure 2: 1) instrumented probe with analyzing; 2) instrumented probe without analyzing; 3) Intercepting probe with analyzing; and 4) Intercepting probe without analyzing.

3.2. Agent

Agent is another mechanism to extract events of Web services. While probes run in the same process with
the application, agent keeps independent from specific application and runs in its own process. It collects events by calling APIs of resource or application actively and periodically.

It is called agent because it is independent of applications, and can invoke APIs automatically. It always sends processed information rather than raw event to the central analyzer. It is responsible for some calculation, e.g. average CPU usage for the past 5 minutes, memory usage for the past 10 minutes.

3.3 Central Analyzer

Central analyzer is used to process different kinds of monitored event. The analyzing result can present the service quality at a higher level, and can be used directly in other actions such as adjusting or re-configuration.

The basic analyzing approach is to validate the monitored events against the pre-specified constraint. The central analyzer can be responsible for the following tasks:

(1) Constraint violation detection. Wrong parameter value region, wrong message order, over weighted load, and response time that exceeds some limit are all typical constraint violations, and thus can be detected by comparing online.

(2) Risk prediction. This function is used to predict potential risks, such as load overweight, memory leak, and deadlock, so as to adjust the system in time. Load overweight can be done by comparing the measurement we got with the threshold set in advance. For programming language without GC, some memory leak checking approaches should be taken to deal with this problem. Deadlock detection is a hard problem can be avoided through some algorithm based on the status of threads.

(3) Problem determination. This function is used to reckon the reason of service behavior deviation. It is very important for deciding what to do further, so as to guarantee the service quality. The typical reckoned reasons include deficient memory and malicious client.

3.4 High-level Representation

To give Web services administrator a high-level view of system quality, the proposed framework provides visual representation of Web services state. The representation is based on the analyzed result: list of violated constraints, list of current risks to web service, why the violation or risk occurred, and even the candidate solutions.

4. How to Use the Framework

In order to use the proposed approach, there are two main steps to take. The first step is to embody the monitoring model according to the specific service system under monitoring (SUM for short). The second step is to complete the monitoring system, based on the embodied monitoring model and the proposed monitoring framework.

4.1 Acquire Service Constraint

The key content of embodying the monitoring model is to acquire service constraint. Service constraint is additional description to service function description. It provides the scale for service quality measurement. Most efficient analyzing results, no matter from probe or from central analyzer, are got by comparing the monitored event with the pre-specified constraint. The constraint can be described formally or informally. Formal constraint description can be used to generate corresponding probe code or rules in central analyzer automatically [10] [11] [12].

4.2 Select Suitable Probe Mechanism

For application independent resource monitoring, the only way to extract events is by agent. Agent is suitable for applications also, if middleware provides API to extract events or states inside the application. However, more application related events are acquired by instrumented code. If application is built with components such as EJB, then interceptor can be used also to extract messages between different components of application.

All other three kinds of events(request, response, and management) are message based. The probe can be implemented by instrument mechanism or interceptor mechanism. This situation proposed a new question to administrator: how to select the best probing mechanism for some specific event?

The selection of probe relies on many different factors. Among them, the following factors are particular important: 1) which kind of platform does the service run on? Different platforms have different ability to support probe mechanism, and only some flexible platforms support the interceptor mechanism; 2) to which degree can the service endure the intrusion? Different monitoring mechanisms bring intrusive effect to the target in different degree. Instrument probe suffers more from this problem. When there are multiple monitoring probes inside a module, things become more complex:
the scattered monitoring code makes the module difficult to understand; 3) is the loss of performance acceptable? All software monitoring mechanisms lead to performance decrease, while instrument mechanism has the least cost. Experiment in section 5 gives more details to these issues.

4.3 Implement Monitoring System

Monitoring framework is the major coverage of the monitoring system. But it is not the whole implementation of monitoring system. To complete the monitoring system, we need to deploy the probe code into the SUM. The deploying process is strongly related with the deploying time. The typical deploying time includes: coding time, compiling time, loading time, and servicing time. Both instrument mechanism and interceptor mechanism can be deployed at these four times.

At coding time, the probe source code can be added to the SUM easily and freely. The defect of coding time deploying is that the probe code may make SUM difficult to understand, thus difficult to maintain.

At compile time, the probe source code can be instrumented to the SUM automatically, according to the service configuration.

At loading time, the probe code and the SUM are integrated when the system is being loaded into main memory. Both of them are manipulated in binary code or intermediate code forms.

The probe code and the SUM are not only manipulated in binary forms when deployed, but also when the service keeps running. It is also called hot deploying.

4.4 Manage Monitoring System

Monitoring system is usually static. That means, once the monitoring system is deployed, it will not change later, and has the same life cycle with the target system. In the proposed framework, the deployed probe can be managed (add, remove, update) online, with the help of online evolution approach [17].

5. Framework Implementation and Experiment

To get an in-depth investigation of the proposed approaches, we have implemented the proposed monitoring framework and have done some experiments, accompanied with a web based example. The goal of these experiments is to evaluate several key aspects that the monitoring system affects the SUM. Especially, we have strong interest with the following two issues: What’s the general performance cost of the monitoring system? And what’s the best probe mechanism for a specific event or constraint?

The web based example is an auction system composed of three web services: “Auction”, “CreditAuthentication”, and “Payment”. The system has two kinds of clients: the bidder and the seller. Each client who wants to join the auction system must register firstly with the personal information. The “CreditAuthentication” service must be called to authenticate the candidate client. The “Auction” service is responsible for the main process. Each auction activity is initialized by a seller’s requesting of an auction. Only the successfully registered bidders can login an auction. Then the bidders are free to bid any item published in the auction system. Each auction activity lasts for a certain period of time. During that time, bidders can query the current highest price. When the activity ended, the auction service should notify all the bidders and the seller about the final bidding result of the item. If the auction ends successfully, there must be one bidder that likes to buy the item at a price higher than the owner’s claiming price. The paying process is done automatically by the “Payment” service. Please visit http://www.sei.pku.edu.cn/~wqx/mass/auction-example for more detailed information about this example.

For this example, we focused on message related events mentioned in section 4.2. Those events can be validated by two kinds of constraints: value constraint and sequence constraint. Probes are deployed to collect those events and compare them with these two kinds of constraints.

We implemented the initial framework and the web based auction example in the following environment: the hardware configuration is Intel Pentium4 3.0G, with 512M main memory, and the software configuration is Windows XP+SP2, JDK1.5, Eclipse + WTP (Web Tools Platform, a plug-in used for quick Web Service development, integrated with Axis 1.3.0). To support distributed and portable monitoring, low level and platform independent communication mechanism such as Socket should be employed. Here, for simplicity, we use Java RMI to implement the communication between probes and the central analyzer. It should be noticed that in this experiment, all the clients, Web services and the analyzer run on the same host.

For value constraint, we monitor the requests to the method “opRegister” of “Auction” service. Each probe is responsible for getting the values of four parameters of this method (socialNumber, creditCardNum, userName, and pwd). These values are compared with the pre-specified value constraints. We calculate the average
request time cost for five cases: 1) pure function without probe (“No Probe” in figure 3); 2) a probe with analyzing ability is instrumented into the function body ("Instrumented"); 3) a probe without analyzing ability is instrumented into the function body and central analyzer is used to check the value ("Instrumented+"); 4) a probe with analyzing ability is encapsulated in an interceptor ("Interceptor"); 5) a probe without analyzing ability is encapsulated in an interceptor and central analyzer is used ("Interceptor+"). The experiment result is shown in Figure 3.

From these two groups of experiment results, we also find that the time cost of instrumented probe is the lowest: it is less than one millisecond in our experiments! And the interceptor based probes is a little larger than instrumented probes. But remember that this kind of probe is independent of method implementation, and thus is much flexible than instrumented probes. For monitoring system that needs to be well maintained, it is still a good candidate. The separate central analyzer is time consuming. But as the central analyzing component of the monitoring framework, it is indispensable in most monitoring systems.

6. Related Work

Software monitoring has been explored for a long time [5][8][15][16]. In recent years, when researchers found that quality is a serious problem to the success of Web services, monitoring Web services became a hot topic.

To our knowledge, the pioneer Web services monitoring framework is REQMON which was introduced in [7]. REQMON uses KAOS to express requirements and tries to capture obstacle online. Although the high-level requirements locate at the center of framework, REQMON introduced the basic monitor implementation approach.

After that work, series frameworks that focused on BPEL monitoring were reported [9] [10] [11]. In [9], the authors proposed five different types of deviations from the requirements. Event stream is got from BPEL engine, not from service, so it is claimed as non-intrusive, with the support of BPEL engine. Another BPEL oriented framework [10] emphasizes the ability of self-healing. It instruments the original BPEL specification to allow execution of the proper rules. The instrument feeds the monitoring manager, a proxy service that is responsible for understanding the Web service. In [11], the authors claimed an architecture that clearly separates the business logic of a web service from its monitoring functionality, and supports both “instance monitors” and “class monitors”. It still based on BPEL engine. Our framework is not BPEL specific, thus can be used in a wider area.

In [12], the authors proposed their Web service monitoring framework. This framework is not limited in BPEL. But it focuses on interaction protocols mainly. Our framework covers more events inside the implementation of web service, such as resource state changing, messages among the components of application.
Besides the academic work, there are also some practical works from industry. For example, Tivoli from IBM [19] and SOA Validation System from Amberpoint [20] all provide some monitoring mechanisms.

7. Conclusion and Future Work

Monitoring online event for Web services is an important issue for the quality of Web service. This paper proposed a model driven monitoring approach for Web services. Compared with the existing related frameworks, it has the following feature: 1) it is quality model driven, 2) it emphasizes the low-level event extracting mechanisms; 3) it provides a guideline of how to build the monitoring system.

As we mentioned in section I, monitoring is just the first step for quality assurance. Our ongoing work include: (1) enhance the analyzing ability of the framework; (2) mining request pattern from the execution of Web services, new constraint from administration operations; (3) extending online updating work [17] to online quality assurance approaches such as online tuning, online migration.

This paper is supported by the National High-Tech Research and Development Plan of China No. 2006AA01Z175, the National Grand Fundamental Research Program of China No. 2005CB321805, and Key National Science Foundation of China, No. 90412011.

References

[12] Zheng Li, Yan Jin, Jun Han, A runtime Monitoring and Validation Framework for web service interactions, 2006 Australian Software Engineering Conference (ASWEC)