An Online Monitoring Approach for Web Service Requirements

Qianxiang Wang, *Member*, *IEEE*, Jin Shao, Fang Deng, Yonggang Liu, Min Li, Jun Han, *Member*, *IEEE*, and Hong Mei, *Senior Member*, *IEEE*

Abstract—Web service technology aims to enable the interoperation of heterogeneous systems and the reuse of distributed functions in an unprecedented scale and has achieved significant success. There are still, however, challenges to realize its full potential. One of these challenges is to ensure the behavior of Web services consistent with their requirements. Monitoring events that are relevant to Web service requirements is, thus, an important technique. This paper introduces an online monitoring approach for Web service requirements. It includes a pattern-based specification of service constraints that correspond to service requirements, and a monitoring model that covers five kinds of system events relevant to client request, service response, application, resource, and management, and a monitoring framework in which different probes and agents collect events and data that are sensitive to requirements. The framework analyzes the collected information against the prespecified constraints, so as to evaluate the behavior and use of Web services. The prototype implementation and experiments with a case study shows that our approach is effective and flexible, and the monitoring cost is affordable.

Index Terms—Web services, requirements, monitoring, constraints.

1 Introduction

Web services are Internet-based software applications published using standard interface description languages and universally available via XML-based communication protocols. While this technology is widely expected to enable the interoperation of heterogeneous systems and the reuse of distributed functions, the industry uptake of this technology has been slow [1]. Some research has revealed that the lack of quality assurance and guarantee, thus, leading to deviation of service behavior from requirements, is one of the important factors [2].

Software requirements are about the expected behavior of the target software in some environment. For service-oriented software, the behavior of service is not only determined by the program itself, but also affected by many other factors, e.g., hardware, network, and even the client requests. For example, limited hardware resources or low network bandwidth tend to cause a long response time, too many client requests may bring about low performance, and malicious requests may even lead to the denial of service, or wrong result. All these unexpected results are deviations of Web service behavior from the requirements. Thus, to be able to monitor (i.e., extract and analyze) certain runtime information about these behavior-related factors is an important issue in ensuring the behavior of Web services consistent with requirements.

Many traditional technologies, such as fault tolerance, transaction, and security assurance, are approaches to ensure the behavior of software systems consistent with requirements. In recent years, some similar but more general approaches have been proposed, e.g., autonomous computing [3], adaptive software [4], and self-managed system [5]. Monitoring is the first step to all these approaches, where the further actions such as decision making and system adaptation are all based on the information extracted through monitoring.

1.1 Monitoring and Research Issues

Software monitoring involves obtaining the information relating to the state, behavior, and environment of a software system at runtime, so as to deal with potential deviations of system behavior from requirements at the earliest possible time. Monitoring is usually carried out in parallel with the system's normal execution, without interrupting its operation. Starting from early 1960s with the advent of debuggers, software monitoring has been widely used for debugging and testing, correctness checking, security and dependability analysis, performance evaluation and enhancement, and system control [6]. A recent taxonomy shows that runtime software monitoring has been used also for profiling, software optimization, as well as software fault detection, diagnosis, and recovery [7].

As a special form of software, Web services also require monitoring as discussed above. In particular, different from traditional software, Web services and their clients are usually distributed in different physical locations and often controlled by different stakeholders. The behavior of Web services and their clients are more difficult to predict. Early

- *Q. Wang, J. Shao, F. Deng, and H. Mei are with the Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing, China, 100871. E-mail: {wqx, meih}@pku.edu.cn, {shaojin07, dengfang07}@sei.pku.edu.cn.*
- *Y. Liu is with the Netease Corporation, Beijing, China. E-mail: liuyg05@sei.pku.edu.cn.*
- *M. Li is with the China Life Insurance Company, Beijing, China. E-mail: limin05@sei.pku.edu.cn.*
- *J. Han is with the Information and Communication Technologies, Swinburne University of Technology, John Street, Hawthorn/Melbourne, Vic. 3122, Australia. E-mail: jhan@swin.edu.au.*

Manuscript received 15 Jan. 2009; revised 4 May 2009; accepted 19 May 2009; published online 28 July 2009.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2009-01-0007. Digital Object Identifier no. 10.1109/TSC.2009.22.

research work on Web service monitoring was reported in [8], following the work of requirements monitoring in dynamic environments [9]. Since then, other researchers have reported their work, with different motivations and frameworks, including [10], [11], [12], [13].

Although there have been many efforts focusing on Web services monitoring, there are still some critical issues that have not been well explored:

- Requirements specification as constraints. The concept of "constraint" has different meanings in different areas. This paper refers to "constraint" as the predetermined condition or rule about the service behavior and other factors impacting on the service behavior. Many requirements for Web services can directly be specified as constraints. The violation of certain constraint will lead to abnormal behavior. Constraints, thus, serve as the basis to check the information extracted from the system. Especially, if the constraints can be specified formally, monitoring mechanisms can be developed to detect abnormal system behaviors automatically. However, there has not been widely accepted constraint specification approach to date.
- Information extraction. Which kinds of information are sensitive to the behavior of Web services, and thus, need to be extracted from the services? How can information be extracted from the services? Although all proposed approaches extract certain specific data for particular purposes, few efforts systematically explore the types of information and extracting mechanisms for Web services, so as to provide a foundation for monitoring implementation.
- Monitoring cost. Many researchers have noted the issue concerning the intrusiveness of monitoring. Some use terms from physical sciences and called it Uncertain Principle in software [6]: "Most monitoring systems, particularly those that rely on software added to sensors, are intrusive to some degree. Completely nonintrusive monitoring systems use dedicated hardware for monitoring." For software-based monitoring mechanisms, to what degree do monitoring mechanisms cost? Are they affordable? Few efforts explore this issue in detail.

1.2 Main Contributions

In this paper, we introduce a new approach for Web service requirements monitoring. Compared with the existing online monitoring efforts, our approach has the following specific features:

- A pattern-based constraint specification approach. While most current work focuses on temporal constraints on service interaction messages [10], [11], [12], [13], this approach can express not only temporal constraints on messages, but also constraints on parameter values of messages. In addition, it is extensible, providing the capability for expressing new types of constraints.
- A monitoring model. System events related to Web service behavior are usually difficult to monitor because they are usually scattered in the whole system. This paper deals with this problem by introducing a monitoring model, systematically covering the different kinds of events.
- A flexible monitoring mechanism. Based on the monitoring model, the mechanism is able to generate automatically monitoring code from the constraint specification. The monitoring code can be deployed into the target system by administrator.
- A monitoring realization process. It provides guidance to system developer to build the monitoring system for given Web services.
- Assessment of monitoring cost. The performance cost of monitoring is examined through a number of experiments. The results show that the cost is acceptable.

The paper is organized as follows: Section 2 motivates this work by presenting and analyzing an auction system, which is built on the Web and published as a service. Section 3 introduces the pattern-based constraint specification approach. Section 4 discusses the monitoring model. Section 5 describes the monitoring framework and its prototype implementation in detail, while Section 6 demonstrates the activities of using the proposed framework by applying the approach to motivating example. Section 7 presents some experiments assessing the performance cost of the monitoring system. Section 8 provides an overview of related work. Finally, Section 9 concludes the paper and lists some future work.

2 Motivation

To ascertain that the behavior of Web services meets their requirements, one may make assumptions about their operating environment and test them before it is deployed, so as to rectify defects of the system. But it is well known that testing can only tell us there was some defect, never reach a conclusion of "no defect." Furthermore, many environmental factors cannot be emulated realistically during test. Online monitoring and consequent remedial actions to deal with system behavior deviation from requirements should thus be considered as the last barrier of defense against software defects. Considering the distributed nature of client programs and unreliable nature of networks, the importance of monitoring is more pronounced for Web services, compared with traditional software systems.

Let us consider an online Auction Service, which provides a virtual place for selling and bidding items. It has two kinds of clients: the sellers and the bidders. The sellers can register with the service (by sending message "opRegisterRequest"), login to the system (by sending message "opLoginRequest"), and publish the information about the item on sale (by sending message "opPublishRequest"). The bidder can register, login to the system (by sending the same aforementioned messages), bid for an article (by sending message "opBidRequest"), or retract a bid (by sending message "opRetractRequest").

Fig. 1 shows part of the WSDL description of the Auction Service, listing only the description about operation "opBid," which involves messages "opBidRequest" and

```
<xsd:complexType name="BidRequest">
  <xsd:sequence>
    <xsd:element name="userName" type="tns:UserName"/>
    <xsd:element name="itemNum" type="xsd:string"/>
    <xsd:element name="price" type="tns:Price"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="BidRequest" type="tns:BidRequest"/>
……
<wsdl:message name="opBidRequest">
  <wsdl:part element="tns:BidRequest" name="bidRequest"/>
</wsdl:message>
<wsdl:message name="opBidResponse">
    <wsdl:part name="bidResponse" type="xsd:boolean"/>
</wsdl:message>
……
<wsdl:operation name="opBid">
    <wsdl:input message="tns:opBidRequest"/>
    <wsdl:output message="tns:opBidResponse"/>
</wsdl:operation>
……
```

Fig. 1. Part of WSDL description for operation "opBid."

"opBidResponse." The complete description of the auction system is available at www.sei.pku.edu.cn/~wqx/mass/auction-example.

WSDL specifies only the operation signature of a service (abstract definition) and location information (concrete part). The research community has widely recognized the need for richer service description to provide more information about a service, e.g., its interaction protocols [13]. In the Auction Service, besides the service signature, there are some important constraints that request messages must follow:

Constraint 1. The parameter "password" of "opRegisterRequest" is an instance of type "String," and consists only of letters "a-z" and numbers "0-9," and the length is not longer than 16 and not shorter than 8.

Constraint 2. All parameters that are instances of the type "Price" must be greater than 0 and are instances of the type "float" with two fraction digits.

Constraint 3. The interval between two "opBidRequest" messages from the same bidder should be longer than one second.

Constraint 4. For each bid session, all the "opBidRequest" messages can only be sent if the "opLoginRequest" message has been sent.

These constraints reflect some of the original requirements for the Auction Service example.

To monitor the Auction Service, we should formally describe these constraints first. Then, the monitoring mechanism can check the extracted system behavior information against them automatically. We will introduce our approach to formal constraint specification in the next section.

A constraint may be concerned with system behavior involving different points of interest in the service. For the Auction Service, the aforementioned four constraints are quite different: Constraint 1 is about one message parameter which has type "String"; Constraint 2 is about all message parameters of a given type; Constraint 3 is about the time interval between request messages; Constraint 4 concerns the order of different request messages. Of course, they are only part of all potential constraints. How can we identify the monitoring points systematically? We need a model that identifies and classifies the factors for monitoring, just like that quality models were widely used to help us to analyze the software quality [14]. Such a monitoring model is the focus of Section 4.

Monitoring mechanisms rely heavily on the implementation of the target system. Even for one specific target software and for some specific constraint, multiple monitoring mechanisms may be available. For example, to verify the value "password" in constraint 1, we can insert verifying code directly into the implementation procedure of Web services. We can also use existing mechanisms of the system software (Operation system or Middleware). With Apache Axis, for example, the Handler that processes SOAP messages can be used to obtain the value of "password." (Handlers are also used to support security, transaction, etc., in Web services.) The extracted information can be checked against the constraint inline, or sent to one central analyzer for analysis. Section 5 deals with these issues.

3 Constraint Specification

Specification of software is a well-explored issue. It has been pointed out that: specifications "are intrinsically incomplete because system correctness depends not only on computational functionality but also on other properties.... It is impractical to expect full specifications of all these properties because of the prohibitive effort required to specify a wide variety of properties.... Although completeness is impractical, it is still appropriate to expect specifications for a common core of properties" [15]. That may explain why although WSDL can describe "operation signature" very well, much attention has been paid to extending the specification of Web services, e.g., security [16], reliability [17], Qos [18], and service interaction protocols [13]. Some of them have been successfully adopted as formal specification, e.g., WS-Security, WS-Reliability, and most Web service platforms provide relevant internal services for these specifications.

From the monitoring viewpoint, there are still many other constraints that deserve much attention. Considering that there are many kinds of constraints, we propose a pattern-based formal constraint description approach. As general repeatable solutions to commonly occurring problems in software design, design patterns [19] have achieved great success, and motivated a series of work on patterns, e.g., analysis pattern [20] and specification pattern [21]. We believe that patterns can be used to capture not only the description of recurring solutions to software design and analysis problems, but also the description of properties or requirements of services and service-based systems [22].

In this paper, we consider mainly two kinds of constraint patterns: value constraint and event constraint. Both of them are divided further into subcategories. The framework of the pattern hierarchy is shown in Fig. 2.

We designed a new description language, called Web Service Constraint Description Language (WSCDL), to

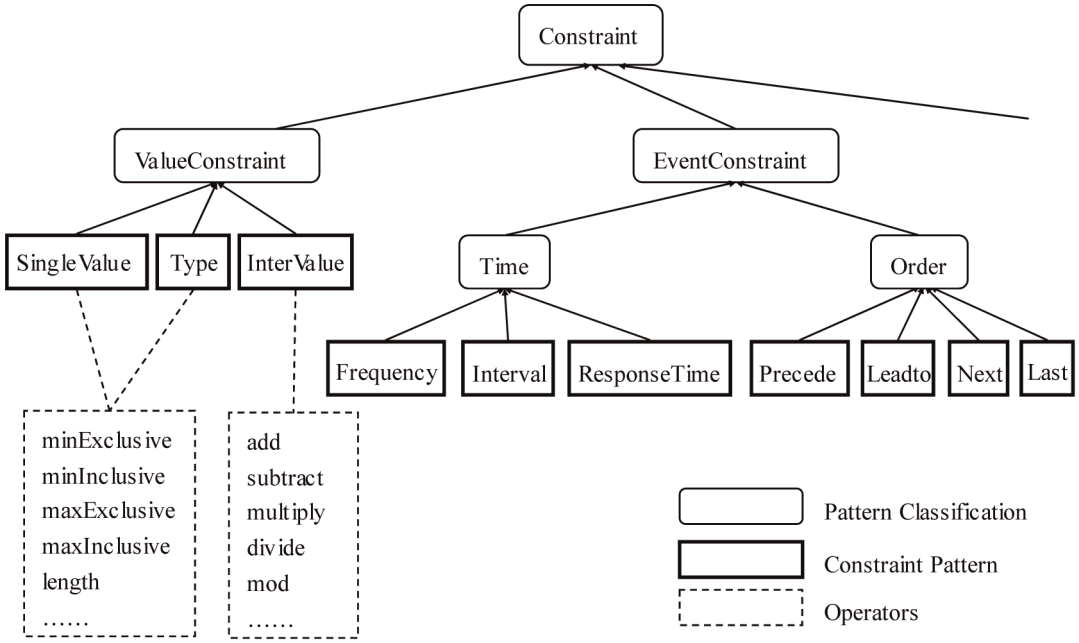


Fig. 2. Pattern hierarchy of Web services constraints. It can be extended to include other constraints, e.g., those about security and reliability.

describe formally these constraints. The WSCDL has the following features:

- It separates the basic description and constraint description for Web services by defining them in WSDL and WSCDL files, respectively. The former information is functional and relatively stable, while the latter is optional and configurable. In such a way, we can change or remove constraints easily: when we change some constraint of service, we can just modify the WSCDL file, without modifying the basic description information in WSDL.
- The syntax of WSCDL is quite simple. Our specification approach is based on Resource Description Framework (RDF) [23] and focuses on the triple concepts of "<subject, predicate, object>."
- It inherits the extensibility mechanism of RDF naturally. You can add your own specification of the newly discovered constraints by extending the specification of a constraint schema.
- Reusable specification is supported in our approach. If there are two constraints that are the same in different services, we can give one WSCDL specification for one service, and let the other refer to it instead of specifying once again.

WSCDL is complementary to WSDL. It lies in the same layer as WSDL in the protocol stack of Web services (i.e., service description layer). Each WSCDL file is attached to a certain WSDL file, and their corresponding relationship can be indicated by their identical main file name but with different postfixes (foo.wsdl versus foo.wscdl). It may also refer to other WSCDL with the "import" keyword declaration to reuse constraint specifications. Two WSCDL files are shown in Figs. 3 and 4, which will be explained below. The remainder of this section provides an overview of our approach to constraint specification. Further details can be found in [24].

3.1 Value Constraint

All interactions between Web services and their clients are carried out through the request and response messages, as defined in the WSDL file. Value constraints are about parameter values of these messages. Many preconditions and postconditions on service operations are constraints on parameters. These constraints are important in determining whether a request is valid, or the Web service behaves correctly. We specify this kind of constraints based on the

```
<Constraint>
  <name>AuctionopRegisterPassword0 </name>
  <category> SingleValueConstraint </category >
  <description>
   The "password" parameter of "opRegister" operation
   should be a String composed of letters and numbers,
   and the length should be no longer than 16 and no
   shorter than 8 </description>
  <context>
    <subject name="password">
      <serviceName>Auction</serviceName>
      <operationName> opRegister</operationName>
      <parameterName>password</parameterName>
      <index>1</index>
    </subject>
  </context>
  <content>
     <minLength> 8 </minLength>
     <maxLength> 16 </maxLength>
     <pattern> [a-zA-Z0-9]+</pattern>
  </content>
  <condition/>
</Constraint>
```

Fig. 3. XML-based formal description of Constraint 1.

```
<Constraint>
  <name> BidLoginConstraint</name>
  <category> PrecedeConstraint </category>
  <description> To the same person, when he send the "op-
                BidRequest", he must have done the "opLogin"
                operation.</description>
  <context>
     <subject name="opLoginResponse">
          <servieName>AuctionService</serviceName>
          <operationName>opLogin</operationName>
     </subject>
     <subject name="opBidRequest">
          <serviceName>AuctionService</serviceName>
          <operationName>opBid</operationName>
     </subject>
  </context>
  <content>
        <event1="opLoginRequest"/>
        <event2="opBidRequest"/>
  </content>
 <condition>opLogin.userName=opBid.userName /condition>
</Constraint>
```

Fig. 4. XML-based formal description of Constraint 4.

XML data types and the constraint facets to these types [25]. To date, we have identified three subpatterns for this constraint pattern: Single Value Constraint, Type Constraint, and Inter Value Constraint.

3.1.1 Single Value Constraint

This subpattern is used to specify the value range of a specific parameter in a message. In WSDL, only the type information is declared. With this pattern, we can identify a constraint, which is represented by one of the 12 constraint facets in XML Data Type (minExclusive, minInclusive, maxExclusive, maxInclusive, length, minLength, maxLength, totalDigits, fractionDigits, pattern, enumeration, and whiteSpace), and apply it to the corresponding parameter. We don't consider these constraint facets as subconstraint patterns under the single value constraint pattern, but operators for expression of value constraints. See Fig. 2.

Constraint 1 on the Auction Service example is a typical single value constraint. Fig. 3 presents the XML-based constraint description of Constraint 1.

3.1.2 Type Constraint

Some WSDL file defines "User-derived datatypes" using "Built-in datatypes" in the XML Data Type when there are multiple parameters that have similar restrictions. We can attach the type constraint to this kind of type definition. With this pattern, we only need to define one type constraint for multiple parameters with similar constraints, not each parameter.

Constraint 2 on Auction Service example is a typical Type Constraint: there are multiple "float" parameters representing the price of different articles and all of them have two fraction digits. So we define a type constraint "the length of 'fraction digits' is two" on the type "Price" defined in WSDL, which is just a "float" type.

This subpattern employs the same constraint facet operators with the "SingleValueConstraint" pattern. The only difference between them is: The former is applicable for a group of parameters that is instance of the same type, while the latter is applicable for a single parameter.

3.1.3 Inter Value Constraint

This pattern is used to specify value constraints that are related to multiple parameters, which belong to different types. They may belong to the same message, belong to different messages of the same Web service, or even belong to different Web services. The constraints may use different computational operators: "Add," "Subtract," "Multiply," "Divide," and "Mod." Because the computation result is still a value, all the constraint facets in Single Value Constraint Pattern are applicable to this pattern.

For the considered Auction Service example, the value of returned price in "opBidResponse" message should be equal to the value of published price in "opBidRequest" message.

3.2 Event Constraint

Event constraints are usually temporal rules on occurrence of messages. Many library APIs of a programming language have implicit event constraints. For example, the Java standard class library v1.3.1 contains 914 classes, out of which at least 81 classes have method temporal constraints [26]. So far, we have identified two subpatterns or this constraint pattern: Time Constraint and Order Constraint.

3.2.1 Time Constraint

This pattern is used to specify the time aspects of one message. Three further subpatterns have been derived so far: 1) frequency constraint, which indicates the number of times that a message can occur in certain duration; 2) interval constraint, which represents the duration of time between two messages' occurrence; and 3) response time constraint, which refers to the duration of time between the request message to a service and the corresponding reply message. These three subpatterns are quite common constraints in the Web service context.

Time constraints can be applied to messages of some specific operation or messages of all operations. Such a constraint could also be applied to the messages of all operations, i.e., the interval between all operations should be longer than one second. Constraint 3 on the Auction Service example is about the request message of operation "opBidRequest."

3.2.2 Order Constraint

This pattern is used to describe the occurrence sequence of some messages, such as the precondition, postcondition, and other limitations. Constraint 4 on the Auction Service example is a typical Order Constraint. Fig. 4 lists XML-based formal specification for Constraint 4.

We identify four further subpatterns to this pattern: "precede," "lead to," "last," and "next." Suppose that we have two events A and B, then:

1. "A precedes B" means when B happens, then A must have happened at least once before.
2. "A leads to B" means once A occurred, then before the end of the whole event sequence, B must occur.

3. "Last event of A is B" means that in the whole event sequence, B is the immediate preceding event of A.
4. "Next event of A is B" means that in the whole event sequence, B is the direct next event of A.

3.3 Pattern Extension

In this paper, the pattern-based approach is used to tackle the incomplete description problem of Web service. One benefit of the pattern-based approach is that it is easy to be extended when a new constraint type is identified.

There are two ways to extend the pattern hierarchy: horizontal and vertical. By horizontal extension, we mean that when some new constraint types are discovered and are not within the scope of the current pattern hierarchy, we can add new patterns to the hierarchy. For example, we can consider WS-Security, WS-Reliability as extensions at the same level as Value Constraint and Event Constraint.

By vertical extension, we mean descriptions that belong to some known patterns, but need some more specification. The typical vertical extension examples are time scope and parameter condition. According to [22], 10 percent of constraints have scopes, which are the extent of the program execution that the pattern must hold. There are four main kinds of scopes: "before," "after," "between...and," and "after...until." One example of scope is that "before event C happens, event A precedes event B." Parameter conditions are used to filter some specific messages and set constraints on these messages. Details about parameter conditions can be found in [13].

4 Monitoring Model

When a system is in operation, various activities and events happen to the system and its operating environment (e.g., request, response, method invocation, etc.), while the system states also undergo frequent changes. For all events and states relevant to a service to be monitored, we need to know first which of them will affect the service behavior. To understand these events and states systematically, we divide them into different categories, using the Chinese Stone Mill[1] model ("Mill model" for short). The model represents graphically the different kinds of events that are constraint sensitive, and thus, need to be monitored (see Fig. 5).

The Mill model has five parts as follows:

1. request message,
2. response message,
3. application (state and event),
4. resource (state), and
5. management operation.

Although these five parts are shown separately, they are actually closely related to each other. For example, lacking resource usually leads to bad response time; malicious requests may lead to the failure of application; code defect of memory leak may quickly exhaust available memory.

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. When we classify the factors to be monitored, we have a model that has a similar shape to the stone mill.

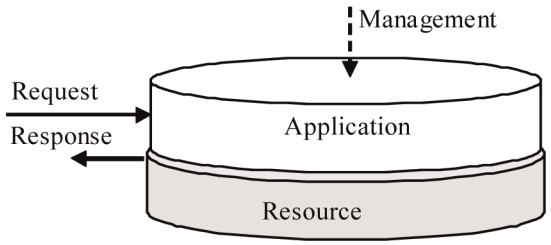


Fig. 5. The Mill model for Web service monitoring, which captures the main factors that should be monitored.

4.1 Request Message

Web services are request-driven software systems. Different from 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. The number of concurrent client connections and the frequency of client requests also have effects on the response time to the clients. Some application-dependent events such as invalid requests and malicious request may also lead to low efficiency or unavailability of service.

Monitoring of request message focuses on the invoked method, parameter values, and client role. Service administrator can specify constraints on these factors, so as to verify whether the request message is acceptable.

For example, in the Auction Service, Constraint 4 is about request messages: For each bid session, all "opBidRequest" messages can only happen after the "opLoginRequest" message.

4.2 Response Message

Request message and response message are the two most important events to be monitored in the proposed framework. Events related to response messages are usually used to evaluate different attributes of service quality. There has been much research into attributes of service quality, from the viewpoint of service selection, composition, and management. This paper separates the attributes of service quality into two classes: basic quality attributes and high-level quality attributes.

Basic service quality attributes are those that can be monitored directly by sniffing response messages. They are also attributes from the client's viewpoint. In other words, a client can "feel" these attributes directly. Availability, correctness, and efficiency are all typical basic service quality attributes.

Availability. Can a client receive the service result, regardless of its correctness? This attribute is application independent and is usually easy to monitor.

Correctness. Can a client receive the correct service result? This attribute relies on the return values in the response message. This attribute is application specific and may include parameter value error, system-level exception, or application-level exception.

Efficiency. Can a client receive the service result under some constraints, such as within limited time period and expected precision? A typical constraint is the response time

concerning the time interval between the request and response messages. Many research efforts into Web service QoS focus on this issue, e.g., [27], [28]. In the Auction Service example, Constraint 2 involves response messages: the value of bid price in "opBidResponse" must be greater than zero.

4.3 Application

Application here means internal events and states of the service. In fact, application itself has the most important effect on service behavior. For example, wrong business logic, deadlock, data race, and inconsistent data are all typical reasons that may bring the service into an error state, or even breakdown of the service. Application-related events occur inside the service such as method executions and changing of variable values. Some applications have in themselves some instrumented code that is designed to raise some important events. Some applications may provide reflective interfaces that can be used to expose some important internal states such as the number of EJB instances. If the application itself doesn't provide such capabilities, monitoring these events relies totally on the system software. Monitoring on this part of Web services has been well explored by traditional monitoring mechanisms [6], [7]. So this paper doesn't explain how to monitor these internal events and states.

4.4 Resource

The execution of application depends highly on low-level resources, e.g., CPU, memory, and file system. The states of those resources, thus, affect the service behavior, especially the performance of the service.

Some typical factors relating to resource state include CPU usage, memory usage, and the number of opened files. There are also some other resource states beyond the operating system. For example, the network bandwidth is a special resource for service provision and the liveliness of a computing node is a more general changing state that affects service availability.

Constraints on these resources are usually specified in the implementation phase and seldom stated directly in service requirements. End users care little about these constraints as well. But they are very important for service providers, so as to ensure the quality of service while consuming the least resources.

Monitoring of such resource-related states depends strongly on APIs that the operating system provides. Many traditional technologies, e.g., fault tolerance and load balancing, have explored this issue very well. As such, we will not further discuss it in this paper either.

4.5 Management Operation

Many Web services provide a management interface for administrator, in addition to the interface for end users. The management interface is used to control the services: reconfiguration, component updating, resource adjusting, message blocking, and so on. These operations are issued by the service administrator and need to be monitored as well, so as to assess whether those operations are correct and effective. For example, some critical adjusting operations can only be issued if there is no active session, in order to keep the service in a safe state. In this area, OASIS has published the Web Services Distributed Management (WSDM) to enhance the management of Web services [29].

Although the interfaces for administrators have quite different functions from the interfaces for end users, the monitoring mechanisms for them are quite similar: Both of them are message-based and rely on the implementation of the Web service platform. We can consequently use similar mechanism to monitor events and states.

5 Monitoring Framework

While the monitoring model aims to answer the question of "what should be monitored," the monitoring framework tries to answer the question of "how to monitor?" We can divide this question further into a series of subquestions: "What are useful low-level mechanisms for online Web service monitoring?" "How to analyze the monitored events?" Despite the fact that the runtime environments that the monitoring framework depends on are different, we aim to provide a generic framework to realize the common capabilities.

Our monitoring framework is composed of distributed probes, an agent, a central analyzer, and a management center. Fig. 6 shows the relationship between main components of the proposed framework and the service under monitoring.

5.1 Probe

As the key component of the monitoring framework, probe extracts events from the target system. Different mechanisms are available to implement probes. Generally, these mechanisms can be classified along two dimensions: instrumentation-based versus interceptor-based and inline-based versus outline-based [6], [33].

Instrumentation is the most widely used monitoring mechanism. It is also widely used in program testing. In this approach, the monitoring code is embedded inside the target code. Traditionally, the instrumentation code is inserted manually by the programmers. The most important feature of the instrumentation approach is that: the code can be inserted freely into any location of the monitored code. Another reason for using instrumentation is that it does not need support from the platform. In recent years, some new instrumentation mechanisms (such as Javaassit and AspectJ) are developed, in order to instrument code automatically according to configuration information.

Interceptors are widely used in Web services implementation. Interceptor can obtain details of the messages to and from the monitored service, and thus, can be used naturally as probes. Interceptors in CORBA, Handlers in AXIS, and JVMTIs in JVM are all well-known interceptors that can do some processing to messages. The most important feature of interceptors is that it is independent of the target system both at coding time and runtime.

Besides the difference in ways of event extraction, probes are also different in analysis capability. Many probes just capture the event and send it to the central analyzer. However, some probes can do some basic analysis, especially for those constraints that are easy to be checked, e.g., parameter value, response time. For those probes that have analysis ability, it needn't forward the extracted

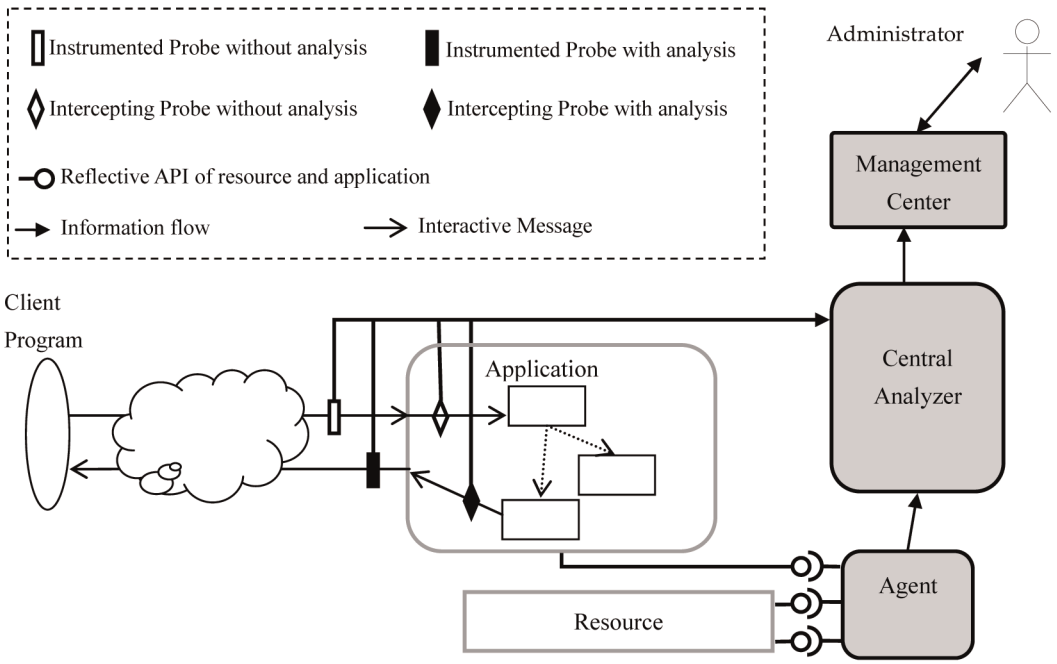


Fig. 6. Relationships among the main components of the monitoring framework and the Web service under monitoring. The elements with darker borderline are main components of the monitoring framework.

information to the central analyzer. The proposed framework can support the following four kinds of probes (see Fig. 6):

1. instrumented probe with analysis,
2. instrumented probe without analysis,
3. intercepting probe with analysis, and
4. intercepting probe without analysis.

Probes are generated from the information contained in the service constraint description. This information identifies the events to be monitored, the verification logic for the constraint, and the ways of deviation reporting. Different probing mechanisms are chosen according to the constraint patterns. For example, inline instrumented probe is generated for constraints of value pattern by default and outline interceptor probe is generated for event pattern constraints by default.

5.2 Agent

While probes observe events in application passively, an agent acts as an active events collector. As discussed in Section 5.1, probes run in the same process as the application. As another mechanism to extract events from Web services, an agent is deployed independent of the specific application and runs in its own process. It collects events by calling APIs of resource or application periodically and proactively. Examples of such APIs include Windows Management Instrumentation (WMI), Java Management Extensions (JMXs), and so on.

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

5.3 Central Analyzer

The central analyzer is used to process different kinds of monitored events. The analysis result can present the service quality at a higher level, and can be used directly in further actions such as adaptation or reconfiguration.

The basic analysis approach is to validate the monitored events against a prespecified constraint. The central analyzer can be responsible for the following tasks:

1. Constraint violation detection. Parameter values that are out of range, wrong message order, overloading, and too long response time are all constraint violations and can be detected by online comparison.
2. Problem determination. This function is used to identify the reason of service behavior deviation. It is very important for deciding what to do next, so as to guarantee the service quality. Typical reasons include deficient memory and malicious clients. According to the analysis result, the central analyzer will offer some suggestions to the administrator.

5.4 Management Center

From the management center, administrators of Web service can be aware of its states through human-friendly management interface and perform potential actions to keep the behaviors of service consistent with requirement. The framework provides a visual presentation of Web service states, based on the analysis result, including a list of violated constraints, why and where the violation occurred, and even some potential solutions.

5.5 Framework Implementation

As a preliminary prototype, the implementation of our framework is based on Java and used to monitor Web services developed in Java-based platform. The prototype

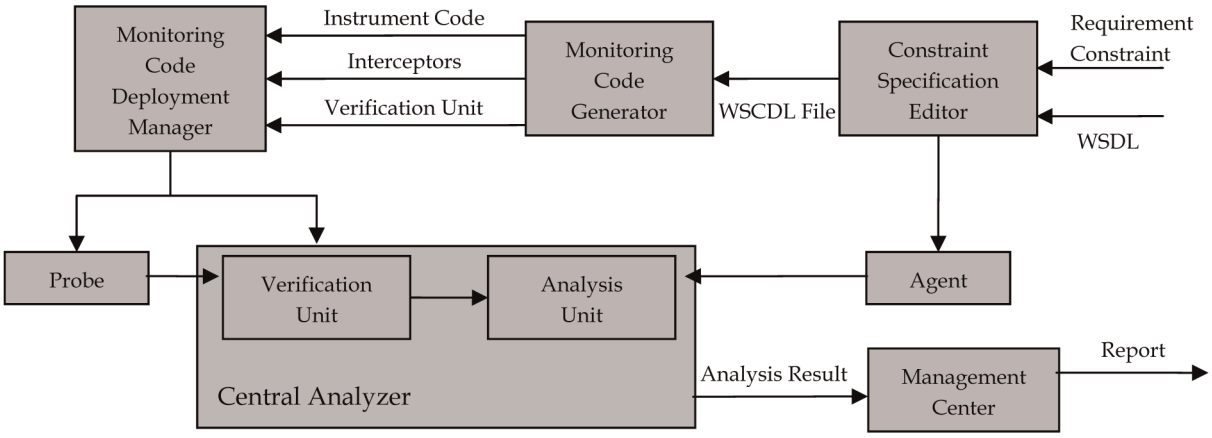


Fig. 7. Architecture of the monitoring framework, including the components used at the monitor development phase.

includes the following parts: constraint specification editor, monitoring code generator, monitoring code deployment manager, agent, central analyzer, and management center, as shown in Fig. 7.

The constraint specification editor helps the user to specify constraints by providing a graphical user interface, which can reduce possible errors caused by manually editing an XML file. This editor supports the tabular and graphical specification modes for value and temporal constraints, respectively, and produces the XML-based constraint specification file. In Section 2, Figs. 3 and 4 list such XML-based constraint specifications for Constraints 1 and 4. The structural information (such as port type and operation signature) about the service to be monitored is first extracted from its WSDL file. Based on this information, the user can then construct the constraint description via the graphical interface. The constraint specification is implemented using Eclipse Modeling Framework (EMF). The snapshot of editor tool is illustrated in Fig. 8.

The monitoring code generator receives as input the XML-based constraint specification file from the constraint specification editor. As shown in Fig. 3, the constraint description file contains information indicating the location of probe (the "context" node), constraint type (the "category" node), and verification logic (the "content" node). The generator then generates the monitoring code, including the probes and the verification unit. The deployment manager takes charge of their deployment.

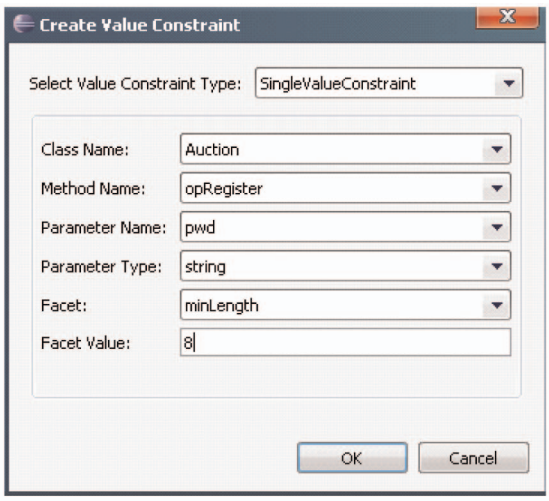


Fig. 8. A snapshot of the constraint specification editor.

Appropriate probe type is chosen according to the constraint type by default. In general, inline instrumentation probe is generated for value constraint and outline interceptor probe is generated for temporal constraint. But this correspondence between constraint types and probe types can also be changed in the constraint specification file. In the prototype, we employ AspectJ [34] as our instrumentation mechanism. An autogenerated probe is added to the original Web service via an aspect in which the join point is defined as the operation under monitoring. Thus, a probe can be instrumented into the system by either compile-time weaving or load-time weaving.

Intercepting probe needs the support from the execution platform of the Web service. In our prototype, we use Apache Axis2 as the supporting Web service platform. Handlers in Axis support the implementation of interceptors. A handler can intercept the SOAP request and response messages.

Agent is implemented as an independent module in the framework. It obtains state information of the application and resources by calling the relevant reflective APIs, such as the Windows Management Instrumentation, Java Management Extension, or Java Virtual Machine Tool Interface. The concerned data are collected and transferred to the central analyzer according to the policy set in the agent.

The central analyzer is implemented in Java as an independent application, which can be run on the same or a separate host as the Web service. The central analyzer includes the verification unit and the analysis unit. The central analyzer loads the generated verification unit dynamically. The verification unit maintains one finite state automaton for each temporal constraint, which accepts event of interest from probes and then changes the state. When reaching an error state, a constraint violation occurs and is reported. The analysis unit receives information from the agent, verifies it, and sends the analysis result to the administrator.

We use Java RMI to implement the communications between the probes and the central analyzer, agent and the central analyzer.

All the verification and analysis results are then sent to the management center for presentation to the administrator.

6 How to Use the Framework

In order to use the proposed approach, there are four main steps to take:

1. acquire service constraints,
2. select suitable probe mechanisms,
3. instantiate the monitoring system, and
4. monitor the service.

6.1 Acquire Service Constraints

The key to instantiating the monitoring model is to acquire the service constraints. Service constraints are additional description to the service's functional description. The most effective analysis results, no matter from probes or from the central analyzer, are obtained by comparing the monitored events against the prespecified constraints. The constraints can be described formally using the constraint description language. Such descriptions are used to generate automatically the corresponding probe code or rules in the central analyzer automatically. The constraint specification editor is used to assist the developer to specify the constraints.

6.2 Select Suitable Probe Mechanisms

For application-independent resource monitoring, the best way to extract information is by agent. Agent is also suitable for extracting events and states from applications if the Web service platform provides the relevant APIs. Application-related events can also be acquired through instrumentation. If the application is developed with components like EJBs, interceptor can also be used to extract messaging between different components of the application.

All other three kinds of events (request, response, and management) are message based. Probes can be implemented using the instrumentation or interceptor mechanism. The selection of one mechanism over the other depends on many factors. The following factors are of particular importance:

1. Which kind of platform does the service run on? Different platforms have different ability to support a particulate probe mechanism, and only some certain platforms support the interceptor mechanism.
2. Is the service source code accessible? Instrumentation requires the availability of the application source code, while interceptor does not.
3. To what degree can the service endure the intrusion? Different monitoring mechanisms bring about different degrees of intrusiveness to the target. Instrumentation probes suffer more from this perspective. When there are multiple monitoring probes inside a module, the matter becomes more complex. The scattered monitoring code makes the module difficult to understand.
4. Is the loss of performance acceptable? All software monitoring mechanisms lead to negative impact on system performance, while instrumentation has the least performance cost (see Section 7 for the detailed cost evaluation).

```
@Aspect
public class opRegister_Aspect{
    @Before("execution(*cn.edu.pku.mass.AuctionService.op
    Register(..))")
    public void validate(JoinPoint pjp) throws
      Throwable {
        ValueConstraintValidator validator =
            new opRegisterValueConstratValidator();
        boolean result = validator.validate
            (((java.lang.String)pjp.getArgs()[0]));
        EventManagerClient client =
            new EventManagerClient();
        EventManagerPortType service =
            client.getEventManagerHttpPort();
        service.sendMessageOfValueConstraint
            ("pws_opRegister",
            "SingleValueConstraint about opRegister",
             result);
    }
}
```

```
public class opRegisterValueConstraintValidator imple-
ments ValueConstraintValidator{
    public boolean validate(Object subject){
        java.lang.String value = (String) subject;
        Pattern p = Pattern.compile("[a-zA-Z0-9]+");
          Matcher m = p.matcher(value);
        if(value.length()>=6&&value.length()<=8
          &&m.matches())
                return true;
        else
                return false;
    }
}
```

Fig. 9. Automatically generated monitoring code from Constraint 1.

All the monitoring codes, including probes and verification units in the central analyzer, can be generated automatically from the XML-based constraint specification file written in WSCDL. For Constraint 1 on the Auction Service example, an inline instrumentation probe can be generated in the form of an AspectJ aspect and the verification code is encapsulated in a separate validating class. The generated code segments are shown in Fig. 9.

6.3 Instantiate the Monitoring System

The monitoring framework constitutes the main part of the monitoring system. However, it alone does not form the entire monitoring system. To complete the monitoring system, we need to deploy the probe code into the service and load the verification unit into the central analyzer.

The deployment process for the probe code is strongly dependent on the deployment time. The typical life cycle phases for deployment includes: coding, compiling, loading,

and runtime. Both the instrumentation and interceptor mechanisms can be deployed at either of these phases.

At the coding time, the probe source code can be added to the service easily. The disadvantage of coding time deployment is that the probe code may make service difficult to understand, thus, difficult to maintain.

In our approach, a probe can be added to the monitored service by compile-time weaving and load-time weaving. At compile time, the probe source code can be weaved to the service automatically. At load time, the probe code and the service are integrated when the system is being loaded into main memory. Both of them involve code manipulation in binary or intermediate form.

6.4 Monitor the Service

Once all the monitoring codes are deployed successfully, the monitoring framework begins to monitor the service against the specified constraints. The central analyzer begins to process and analyze the monitored events, and the results are presented to the administrator via the management center.

For value constraints, the verifications are performed in the probes every time the probes receive the value, and any violation is reported to the central analyzer. For temporal constraints, the verification is performed outline in the central analyzer, where the loaded verification unit maintains a finite state automaton for each temporal constraint. The automation accepts events of interest from the probes and advances its state. When an error state is reached, a constraint violation is detected and reported.

7 Evaluation and Discussion

To assess the effectiveness and impact of the proposed approach and framework, we carried out some experiments in association with the motivating Auction Service example system. The goal of these experiments is to evaluate how the monitoring system impacts on the target service, especially its end users. In particular, we have strong interest in the following two issues: Whether the impact is affordable? And what is the better probe mechanism for a specific event or constraint in a specific situation? Since performance monitoring has been studied widely and is not our focus, we do not show experiments on CPU and memory utilization that are monitored by the Agent.

7.1 Evaluation

We implemented the prototype framework and the Auction Service example in the following environment: the hardware configuration is an Intel Pentium4 3.0 GB with 512 MB main memory; the software configuration is Windows XP+SP2, with JDK1.5 and Eclipse + WTP (Web Tools Platform, a plug-in used for quick Web Service development, integrated with Axis2). It should be noted that in the experiments, all the clients, Web service, and the analyzer run on the same host. These experiments are carried out in a simulated environment in laboratory.

The experiments were conducted on the same example given in Section 2. Each auction activity is initialized by a seller's requesting an auction. Only the successfully registered bidders can log in to the system and make bids on any

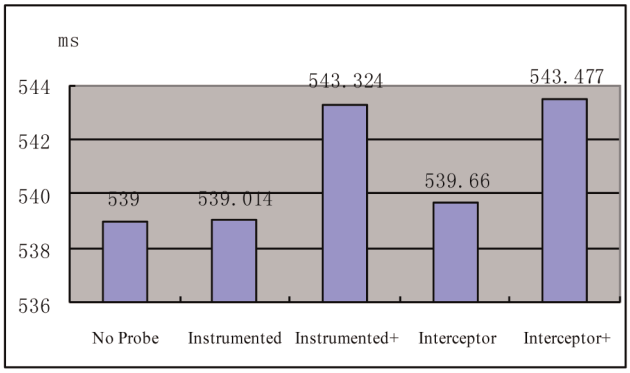


Fig. 10. The average time cost of "opRegisterRequest" messages.

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 activities end, 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 reserve price.

For this example, we have focused on message-related events discussed in Section 4.1. Those events can be used to validate two types of constraints: value constraint and order constraint. Probes are deployed to collect those events and they are checked against the constraints.

For Constraint 1 on Auction Service, we monitor the "opRegisterRequest" message to the Auction Service. Each probe is responsible for getting the values of four parameters of this method (socialNumber, creditCardNum, userName, and password). These values are compared with the prespecified value constraint. We calculate the time cost (from the time that request messages were sent, to the time that response messages were obtained) for the following five cases:

1. Original function without probe ("No Probe" in Fig. 10).
2. A probe with analysis ability is instrumented into the function body ("Instrumented").
3. A probe without analysis ability is instrumented into the function body and the central analyzer is used to check the value ("Instrumented+").
4. A probe with analysis ability is encapsulated in an interceptor ("Interceptor").
5. A probe without analysis ability is encapsulated in an interceptor and the central analyzer is used for analysis ("Interceptor+").

For each case above, we simulate the message for 100 times and calculate the average time cost to get more reliable results.

The experiment results are shown in Fig. 10. It is easy to see that in general, the cost of the monitoring system is affordable: all monitoring costs are in the range of milliseconds. Compared with the normal function execution, which costs several hundred milliseconds, this cost is very low. Even for the probe of "Interceptor+" class, the

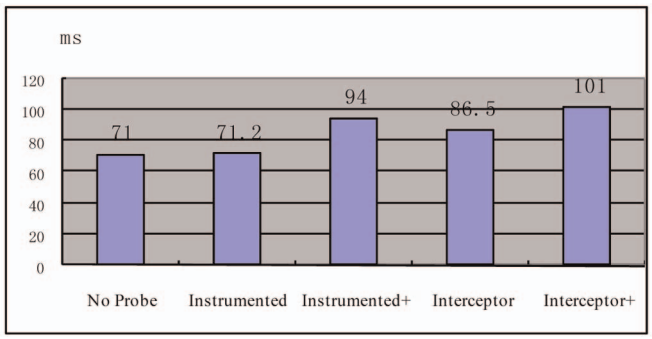


Fig. 11. The average time cost of "opLoginRequest" and "opBidRequest" messages.

cost is only 4.477 ms, which is about 0.83 percent of the "No Probe" execution time.

For Constraint 4 on Auction Services, we monitor the opLoginRequest message and opBidRequest message. The correct request sequence is opLoginRequest and opBidRequest. Other sequences will lead to a constraint violation. The five cases are the same as in the value constraint experiment. For each case, we simulate the message sequence for 100 times and calculate the average results. The experiment results are shown in Fig. 11. The result is similar with Fig. 10, except that all the times are about 1/7 to 1/6 of those in Fig. 11. That is because that "opRegister" operation, which is inside the application, needs to access more data in DBMS, may consume much time.

From these two groups of experiment results, we also observe that the time cost of instrumented probes is the lowest: it is less than one millisecond in our experiments. The time cost for the interceptor-based probes is a little higher than instrumented probes. But note that this kind of probe is independent of service implementation, and thus, is much flexible than instrumented probes. For a monitoring system that has better maintainability, it is still a good candidate.

Another obvious observation we can make is that outline monitoring mechanisms have a higher cost than inline ones. This reflects the additional message passing cost between the probes and the central analyzer.

7.2 Discussions

Expressiveness and extensibility. Our Web service constraint specification language, WSCDL, supports not only the temporal patterns in the Specification Pattern System [26], but also value constraints on the parameters of service interaction messages. By adding the time patterns, the language can also express certain real-time constraints such as Constraint 3 in Section 2.

We note that our pattern-based constraint specification language is extensible. As they are discovered and formulated, patterns concerning other aspects of services can be added to the language, such as security and reliability.

Completeness. By analyzing systematically the issues concerning service monitoring, our monitoring model identifies and classifies the factors (events, states, and messages) that need to be monitored, thereby providing a reference model for designing (custom) monitoring systems.

By including instrumentation, interception, and agent-based probe design techniques, our monitoring framework gains more flexibility and general applicability. Instrumentation, interception, and/or agent-based probing mechanisms can be adopted depending on the implementation language and platform of Web services.

Generality. Our prototype implementation of the monitoring framework is based on Java, and is used to monitor Web services developed in Java and deployed in the Apache Axis2 Web service platform. However, this framework is not restricted to only this implementation. It provides a generic way of implementing Web service monitoring, which is independent of the platform and operating environment. Practitioners can implement their own monitoring tools in their particular environments by referring to this framework.

8 Related Work

Software monitoring has been explored for a long time in various fields, such as performance evaluation and enhancement, fault tolerance, and autonomic computing [6], [9], [30], [31]. Based on these efforts, Delgado et al. [7] proposed a taxonomy and catalog for runtime monitoring based on a comprehensive survey of the area, which analyzed and summarized research concerned with monitoring of traditional software. The specific characteristics of Web service that differs a lot from traditional software (such as dynamism and loose coupling) have posed additional issues for investigation when implementing Web service monitoring. While some work [36] focuses on monitoring of nonfunctional requirements, our work together with [8], [10], [11], [12], [13], and [35] focuses on monitoring of functional requirements. As followed, our work is compared with these works in three aspects.

8.1 Constraint Specification

There have been different ways of specifying constraints. An early effort in the domain of monitoring Web service requirements is the research conducted by Robinson [8] in which requirements are expressed using KAOS. But KAOS only presents a high-level goal-oriented software requirement, which cannot provide enough information needed for monitoring. Thus, the transformation from specification to code needed human involvement.

Mahbub and Spanoudakis [10] use event calculus to specify behavioral properties and assumptions that are to be monitored. The event logs recorded during the execution of a service-based system were then checked against the specifications. However, event calculus is not very intuitive for software developers to use, which impeded the adoption of this approach.

Based on Dwyer's Property Specification Pattern System, Li et al. [13] and [35] proposed their approaches for specifying constraints, respectively. While Li et al. [13] provide a way to specify interaction constraints in association with the service interface definition using the OWL-S framework, [35] identifies a subset of UML 2.0 Sequence Diagram as a property specification language. Comparing to their work, we are not only able to specify service interaction constraints, but also able to specify value constraints about message parameters, which has made the constraint specification more complete.

8.2 Monitoring Target

There have been a number of frameworks focusing the monitoring of service compositions in BPEL [10], [11], [12]. Mahbub and Spanoudakis [10] have introduced five different types of system behavior deviations from requirements. The event stream is obtained from the BPEL engine, instead of services. Another BPEL-oriented framework [11] emphasizes the ability of self-healing. It instruments the original BPEL specification to allow execution of the required rules. The instrumentation feeds the monitoring manager, a proxy service that is responsible for understanding the Web service. Barbon et al. [12] proposed an architecture that separates the business logic of a Web service from its monitoring functionality, and supports both "instance monitors" and "class monitors."

Comparing with these works, our work focuses on the monitoring of individual Web services, including the client and operating environment. However, it has the ability to extend to service composition monitoring by treating service compositions as an application.

The authors of [13] of proposed a Web service monitoring framework. This framework focuses on the monitoring of service interaction protocols. The proposed framework in this paper covers more events inside the implementation of Web services, such as resource state changes, and messages among the components of application.

8.3 Interleaving Manner

Another aspect to be discussed is how the monitoring code interleaves with the original service. Many approaches aim to achieve nonintrusiveness in their monitoring frameworks [10], [11]. But they usually need the support from the underlying platform. Our framework accommodates both intercepting probes (when there is platform support) and instrumentation probes (when there is no platform support for interceptors).

9 Conclusion and Future Work

This paper has introduced an online monitoring approach for Web service requirements. It is capable of extracting relevant information about a Web service and its operating environment, and providing support for checking the service behavior against its requirements (in the form of constraints). The proposed approach begins with the pattern-based constraint specification in a new language, WSCDL. The language supports the specification of value and event constraints. The proposed monitoring model covers a wide range of system events and states, and systematically identifies the locations for deploying monitors. The monitoring framework is able to use the formally specified constraints to generate automatically monitoring code. A process guide for how to use the proposed framework is also introduced. Based on a prototype implementation of the framework and a case study, we have conducted a series of experiments to assess effectiveness and impact of our approach. The experiment results have shown that our approach is effective and flexible, and the efficiency and monitoring cost are affordable.

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

As mentioned in Section 1, monitoring is only the first step for quality assurance. Our ongoing work includes: 1) enhancing the analysis ability of the framework; 2) mining request patterns from the execution of Web service, new constraints from administration operations; and 3) extending our work in online updating [32] to include other online quality assurance capabilities such as online tuning and online migration.

Acknowledgments

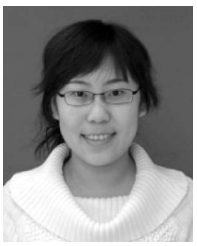
This work was supported in part by grants from the National High-Tech Research and Development Plan of China (No. 2006AA01Z175), the National Grand Fundamental Research Program of China (No. 2005CB321805), the National Natural Science Foundation of China (No. 60773160), the Key National Science Foundation of China (No. 90412011), and the Australian Research Council (No. LP0775188). Y. Liu and M. Li were with the Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing, China, 100871. H. Mei is the corresponding author.

References

[1] C.S. Langdon, "The State of Web Services," *Computer,* vol. 36, no. 7, pp. 93-94, July 2003.
[2] S. Degwekar, S.Y.W. Su, and H. Lam, "Constraint Specification and Processing in Web Services Publication and Discovery," *Proc. Second IEEE Int'l Conf. Web Services (ICWS '04),* pp. 210-217, 2004.
[3] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer,* vol. 36, no. 1, pp. 41-50, Jan. 2003.
[4] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, and A.L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems,* vol. 14, no. 3, pp. 54-62, May/June 1999.
[5] J. Kramer and J. Magee, "Self-Managed Systems: An Architectural Challenge," *Proc. Conf. Future of Software Eng. (FOSE '07),* pp. 259-268, 2007.
[6] B.A. Schroeder, "On-Line Monitoring: A Tutorial," *Computer,* vol. 28, no. 6, pp. 72-78, June 1995.
[7] N. Delgado, A.Q. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," *IEEE Trans. Software Eng.,* vol. 30, no. 12, pp. 859-872, Dec. 2004.
[8] W.N. Robinson, "Monitoring Web Service Requirements," *Proc. 11th IEEE Int'l Conf. Requirements Eng. (RE '03),* pp. 65-65, 2003.
[9] S. Fickas and M.S. Feather, "Requirements Monitoring in Dynamic Environments," *Proc. Second IEEE Int'l Symp. Requirements Eng.,* pp. 140-147, 1995.
[10] K. Mahbub and G. Spanoudakis, "Run-Time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience," *Proc. Third IEEE Int'l Conf. Web Services (ICWS '05),* pp. 257-265, 2005.
[11] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of Ws-Bpel Processes," *Proc. Third Int'l Conf. Service-Oriented Computing (ICSOC '05),* 2005.
[12] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes of Web Service Compositions," *Proc. Fourth IEEE Int'l Conf. Web Services (ICWS '06),* pp. 63-71, 2006.
[13] Z. Li, Y. Jin, and J. Han, "A Runtime Monitoring and Validation Framework for Web Service Interactions," *Proc. 17th Australian Software Eng. Conf. (ASWEC '06),* pp. 70-79, 2006.
[14] B.W. Boehm, *Characteristics of Software Quality.* North-Holland, 1978.

[15] M. Shaw, "Truth vs. Knowledge: The Difference between What a Component Does and What We Know It Does," *Proc. Eighth Int'l Workshop Software Specification and Design,* pp. 181-185, 1996.
[16] D. Geer, "Taking Steps to Secure Web Services," *Computer,* vol. 36, no. 10, pp. 14-16, Oct. 2003.
[17] S. Ran, "A Model for Web Services Discovery with Qos," *ACM SIGecom Exchanges,* vol. 4, no. 1, pp. 1-10, 2003.
[18] V. Tosic, B. Pagurek, and K. Patel, "WSOL—a Language for the Formal Specification of Classes of Service for Web Services," *Proc. First Int'l Conf. Web Services (ICWS '03),* pp. 23-26, 2003.
[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.
[20] M. Fowler, *Analysis Patterns: Reusable Object Models.* Addison-Wesley Professional, 1997.
[21] S. Konrad and B.H.C. Cheng, "Real-Time Specification Patterns," *Proc. 27th Int'l Conf. Software Eng.,* pp. 372-381, 2005.
[22] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett, "Patterns in Property Specifications for Finite-State Verification," *Proc. 21st Int'l Conf. Software Eng. (ICSE '99),* pp. 411-420, 1999.
[23] W3C, *Resource Description Framework (RDF),* http://www.w3.org/RDF/, 2004.
[24] Q. Wang, M. Li, N. Meng, Y. Liu, and H. Mei, "A Pattern-Based Constraint Description Approach for Web Services," *Proc. Seventh Int'l Conf. Quality Software (QSIC '07),* pp. 60-69, 2007.
[25] D. Peterson, S. Gao, A. Malhotra, C.M. Sperberg-McQueen, and H.S. Thompson, *W3c XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes,* World Wide Web Consortium (W3C) recommendation, http://www.w3.org/TR/xmlschema11-2/, June 2008.
[26] J. Whaley, M.C. Martin, and M.S. Lam, "Automatic Extraction of Object-Oriented Component Interfaces," *ACM SIGSOFT Software Eng. Notes,* vol. 27, no. 4, pp. 218-228, 2002.
[27] L. Taher, R. Basha, and H. El Khatib, "QoS Information & Computation (QoS-IC) Framework for QoS-Based Discovery of Web Services," *Standardization for ICT Security,* vol. 6, no. 3, 2005.
[28] A. Mani and A. Nagarajan, "Understanding Quality of Service for Web Services," http://www.ibm.com/developerworks/library/ws-quality.html, 2002.
[29] *Oasis Web Services Distributed Management (WSDM) TC,* OASIS, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm, 2006.
[30] D.K. Peters and D.L. Parnas, "Requirements-Based Monitors for Real-Time Systems," *IEEE Trans. Software Eng.,* vol. 28, no. 2, pp. 146-158, Feb. 2002.
[31] M.S. Feather, S. Fickas, A.V. Lamsweerde, and C. Ponsard, "Reconciling System Requirements and Runtime Behavior," *Proc. Ninth Int'l Workshop Software Specification and Design,* p. 50, 1998.
[32] Q. Wang, J. Shen, X. Wang, and H. Mei, "A Component-Based Approach to Online Software Evolution," *J. Software Maintenance and Evolution: Research and Practice,* vol. 18, no. 3, pp. 181-205, 2006.
[33] 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, pp. 108-127, 2003.
[34] G. Kiczales and E. Hilsdale, *Aspect-Oriented Programming.* Springer, 2003.
[35] J. Simmonds, M. Chechik, S. Nejati, E. Litani, and B. O'Farrell, "Property Patterns for Runtime Monitoring of Web Service Conversations," *Proc. Eighth Workshop Runtime Verification,* 2008.
[36] L. Fei, Y. Fangchun, S. Kai, and S. Sen, "A Policy-Driven Distributed Framework for Monitoring Quality of Web Services," *Proc. Sixth Int'l Conf. Web Services (ICWS '08),* pp. 708-715, 2008.

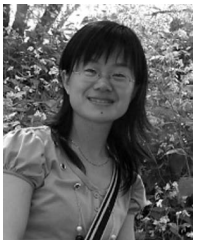
Qianxiang Wang received the PhD degree from the Northwestern Polytechnic University, China, in 1997. He is currently a professor in the School of Electronics Engineering and Computer Science at Peking University. His research interests include software monitoring, program analysis, and middleware. He has published more than 50 papers. He is a member of the IEEE and the IEEE Computer Society.



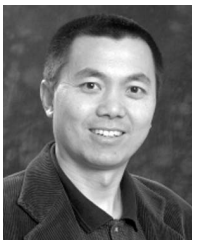
Jin Shao received the BSc degree from Northeastern University, China, in 2007. She is currently working toward the PhD degree in computer science in the School of Electronics Engineering and Computer Science at Peking University. Her research interests include software monitoring and middleware.

Fang Deng received the BSc degree from Peking University, China, in 2007. She is currently working toward the master's degree in computer science in the School of Electronics Engineering and Computer Science at Peking University. Her research interests include software monitoring and middleware.

Yonggang Liu received the BSc degree from the Beijing Information Technology Institute in 2003 and the master's degree from Peking University in 2008. He is now with the Netease Corporation, Beijing, China.



Min Li received the BSc degree from the Civil Aviation University of China in 2004 and the master's degree from Peking University in 2008. She is now with the China Life Insurance Company, Beijing, China.



Jun Han received the PhD degree from the University of Queensland, Australia, in 2002. He is currently a professor of software engineering at the Swinburne University of Technology, Australia. His research interests include services engineering, software architecture, and adaptive software systems. He has published more than 120 peer-reviewed articles. He is a member of the IEEE and the IEEE Computer Society.

Hong Mei received the PhD degree from the Shanghai Jiaotong University, China, in 1992. He is currently a professor in the School of Electronics Engineering and Computer Science at Peking University. His research interests include software engineering, software reuse, and programming languages. He has published more than 180 papers. He is a senior member of the IEEE and the IEEE Computer Society.