COPYRIGHT © 2010 JCIT, ISSN 2078-5828 (PRINT), ISSN 2218-5224 (ONLINE), VOLUME 01, ISSUE 01, MANUSCRIPT CODE: 100714

A Case Based Tool for Monitoring of Web Services Behaviors

Sazedul Alam

Abstract—Monitoring systems is of great practical importance. The use of service oriented distributed system is increasing. Within service orientation (SO) web services (WS) are the de facto standard for implementing service-oriented systems. Users want uninterrupted service from service providers. So the management of these systems is becoming crucial. Monitoring is an important part of any management system. This paper presents a case based tool for monitoring of Web Services. This case based tool uses distributed model driven event processing. Use of models allows developing of generalized solutions for the problem that occur in the service oriented system. This paper provides an overview of the problem, review of existing work and presents a novel approach towards the management of web services.

Index Terms—Distributed Systems, Service Oriented Architecture, Web Services, Monitoring and Management, Event Processing.

1 INTRODUCTION

THE use of distributed applications is increasing daily. As a result, we become increasingly dependent on them - making it necessary to monitor, track and manage them. The functionality and components of distributed systems are spread over a variety of resources, including web servers, application servers, messaging backbones, legacy resources, applications, and so on.

Most organizations have high level monitoring tools to enable coarse grained management of individual resources. These tools don't have any integrated solution to automatically monitor, analyze, and resolve problems at the service, transaction, application, and resource level [1]. Distributed applications tend to be more difficult to monitor and maintain than centralized applications. When users experience application problems, e.g., service brownouts and slowdowns, it is difficult and time consuming to trace and identify the cause of the failure to an individual server, network link, or software element and fix the problem. The problem arises because there is no generalized method for tracing the flow of transactions within a distributed system [1].

Recently service-orientation (SO) has emerged as new system design/integration paradigm that promises to overcome the management problem of distributed systems. Within SO web services (WS) are seen as the de facto standard for implementing service oriented systems. WS enable different applications running on different machines to exchange data and integrate with one another without requiring additional software. The applications that are built on WS technology can exchange data regardless of the language, platform, or internal protocols they use. As a WS network grows, its existence and performance becomes crucial to the business's core activities. So the management of WS is important for providing seamless access to the user of the service.

The important functions of service management include Service Level Agreement (SLA) management, auditing, monitoring, troubleshooting, service redeployment, dynamic routing and graceful degradation, service life cycle and state management, dynamic service provisioning, security management, service maintenance, and service management for WS infrastructure and applications. In this paper main focus will be on monitoring and event processing. Monitoring is needed for the management of distributed systems. The gathered information helps to make management decision and helps to perform the required control actions on the system.

To improve monitoring of web services this paper introduces a case based monitoring tool that uses distributed model-driven event processing. In this approach the events are recorded in an event database. Then event generator reads the events from event database and stores them in XML formatted files. From the files the event generator sends the XML formatted events to the event routers over the network connection. The event router then converts the XML coded event into event instance. The event instances are sent to the event processors. Event router forwards them to the event processors. The events are analyzed by the event processing application using case based scenario models.

The remainder of the paper is organized as follows: Section 2 presents related literature on service oriented architecture (SOA), web services (WS), issues regarding monitoring and management of distributed systems, management of WS, Section 3 presents the proposed architecture of case based monitoring tool, Section 4 discusses the prototypical

• *Sazedul Alam is with the Department of Computer Science and Engineering, University of Asia Pacific (www.uap-bd.edu), Dhanmondi, Dhaka-1209, Bangladesh. E-mail: sazedul.alam@gmail.com*

implementation and preliminary results, and Section 5 concludes the paper with a summary and a outlook of future work.

2 Related Literature

2.1 Service Oriented Architecture

SOA is an approach for designing a software system that offers services to either end-user applications or to other services. SOA as a system design/integration philosophy is independent of specific technologies, e.g., web services, or J2EE.

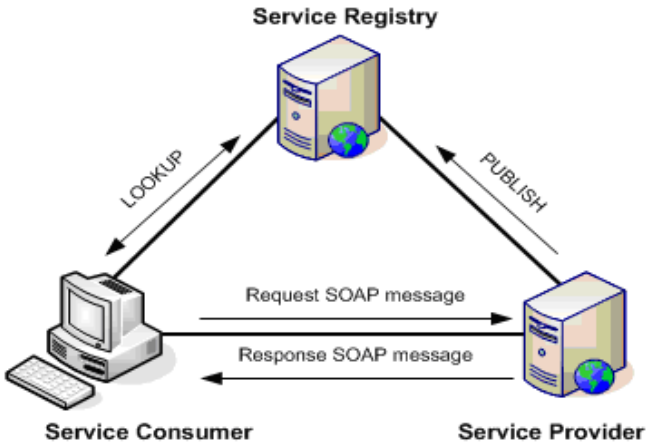

Fig. 1. Service Oriented Architecture.

As shown in fig 1. the main building blocks of SOA are – service provider, the service registry, and the service consumer (client). Providers are software agents that provide the service. Providers are responsible for publishing a description of the services they provide via a service registry. Clients are software agents that request the execution of a service. Agents can be simultaneously both service clients and providers. Clients must be able to find the description of the services they require and must be able to bind to them.

From a business perspective the WS provider is the organization that owns the WS and implements the business logic that underlies the service. From an architectural perspective this is the platform that hosts and controls access to the service. WS are published in a service registry hosted by a service discovery agency. Service publication involves describing the business, service, technical information of the WS and registering this information with the registry in a format specified by the discovery agency.

2.2 Web Service

A WS is a self-describing, self-contained software module available via a network, such as the internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application [1]. WS constitute a distributed computer infrastructure made up of many different interacting application modules trying to communicate over a private or public network to virtually form a single logical system. WS are loosely coupled software modules. The service interface is defined in a neutral manner that is independent of the underlying platform, the operating system, and the programming language the service is implemented in. This allows services, built on a variety of such systems, to interact with each other in a uniform and universal manner.

A WS is a self contained software module that performs a single task. The module describes its own interface characteristics, i.e., the operations available, the parameters, data typing, and the access protocols, in such a way that other software modules can invoke its functionality, and knows what results to expect.

WS can be accessed programmatically. A web service provides programmable access – this allows embedding WS into remotely located applications.

WS can be dynamically found and included in applications. WS are described in terms of a standard description language. The Web Services Description Language, WSDL, describes both functional as well as non-functional service characteristics.

Functional characteristics include operational characteristics that define the overall behavior of the service while non-functional characteristics mainly describe characteristics of the hosting environment.

WS are distributed over the internet. WS make use of existing, ubiquitous transport internet protocols like HTTP. This helps to comply with current corporate firewall policies.

2.3 Management of Distributed Systems

Since distributed applications are more difficult to monitor and manage than the applications that are running in a single system, organizations require monitoring and management of these distributed applications.

"A managed or manageable resource in a distributed environment could be any type of a hardware or software component that can be managed and that can maintain embedded management related metadata. A managed resource could be a server, storage unit, database, application server, service, application, or any other entity that needs to be managed." A distributed application management system controls and monitors an application throughout its lifecycle, from installing and configuring to collecting metrics and tuning the application to ensure responsive execution [1]. Manageability can be distinguished by three functional parts [2]:

1. Monitoring: The ability to capture run-time and historical events from a particular component, for reporting and notification.
2. Tracking: the ability to observe aspects of a single unit of work or thread of execution across multiple resources.
3. Control: The ability to alter the run-time behavior of a managed resource.

Manageability requires a mechanism to ensure that an application is both active and functioning properly, as well

as the ability of checking an application's performance over time. Any management solution must address four necessary management questions [3].

1 What resources need to be managed?
2 What are their properties?
3 How the management information is exchanged (operations, notifications, and kinds of protocols needed)?
4 What are the relationships among the managed resources?

Distributed management solution should have the components for user experience monitoring, infrastructure monitoring, transaction monitoring, resource provisioning, SLA monitoring. Enterprise management systems are network management systems capable of managing devices, independent of vendors and protocols, in IP based enterprise networks [4]. Most Enterprise management systems use functions like fault management, configuration management, accounting management, performance management, and security management [4].

The SNMP (Simple Network Management Protocol) is an application layer protocol for network management [5]. The SNMP framework lets network administrators manage network performance, find and solve network problems, and plan for network growth.

The Common Information Model (CIM)/ Web based Enterprise Management also provide solution for management of elements across the enterprises, including systems, networks, and applications [6].

Java Management Extensions (JMX) technology provides the tools for building distributed, web-based, modular, and dynamic solutions for managing and monitoring devices, applications, and service driven networks [7].

2.4 Distributed Management of Web Services

With SOAs, a standard WS management framework can provide support for discovering, introspecting, securing, and invoking managed resources, management functions, and management infrastructure services and toolsets [1].

WS need to be managed in at least two dimensions – one is operational management dimension and the other is the tactical or business management dimension. Tactical WS management provides end-to-end visibility and control over all parts of a long lived, multi-step information request or transaction/process that spans multiple applications and human actors in one or more enterprises. Operational WS management is the functionality required for discovering the existence, availability, performance, health, patterns of usage, extensibility, as well as the control and configuration, lifecycle support, and maintenance of a web service or business process within the context of SOAs.

A WS management framework is a set of components and objects that enable SLA management, auditing, monitoring, troubleshooting, dynamic service provisioning, and service management for WS infrastructure and applications.

Management Using Web Services (MUWS) enables management of WS using WS [8]. MUWS provides management capability for monitoring the quality of a service, enforcing a service level agreement, controlling a task, and managing a resource lifecycle.

MUWS is able to work with multiple, existing, domain-specific models. CIM from DTMF is the most prominent model for this work [6]. SNMP information model is also considered. Access to the manageability for a resource is provided by a web service endpoint. The web service endpoint providing access to some managed resource is called a manageability endpoint. A manageability endpoint realizes a number of management interfaces. Each management interface represents all or part of a manageability capability. Similarly, a single manageability capability may be represented in one or more interfaces.

Management of Web Services (MOWS) is based on the concepts and definitions expressed in the MUWS specification [9]. WSDM MOWS defines the MUWS capabilities that are applicable to WS endpoints are - Identity, Identification, Metrics (Number of requests, number of failed requests, number of successful requests, service time, max response time and last response time), Operational State, Operational status, request processing state [10].

3 Proposed System Architecture

This paper introduces a case based tool for monitoring of WS. This tool uses case based distributed event processing.

It has following major components:

1 Enterprise Service Bus (ESB)
2 Event Database (ED)
3 Event Generator (EG)
4 Event Router (ER)
5 Event Processor (EP)

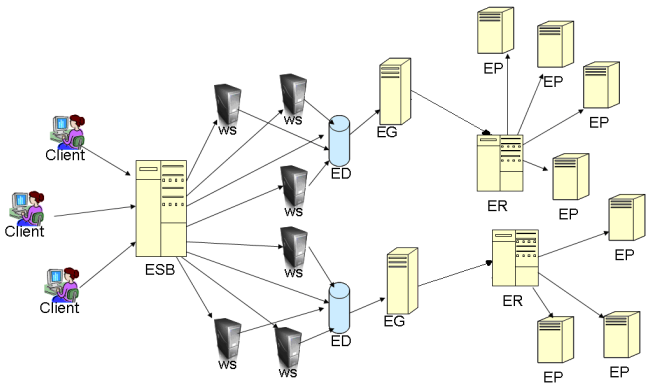

Fig. 2. Proposed System Architecture.

3.1 Enterprise Service Bus (ESB)

An Enterprise Service Bus is the main interface between client and WS. Client sends request for the WS to the ESB. ESB send the request to the proper WS. If the WS generates any response message of the request to the client that response is sent to the ESB at first and then ESB sends the response to the client. So there is no direct communication

between client and web service provider.

The use of ESB has some advantages. If any WS is down then the client need not know about it. In this case ESB will re-direct the request to another running WS which provides same service. This will improve availability of the service from the client's perspective. The use of ESB also shields the WS from malicious clients.

3.2 Event Database (ED)

Different events occur in the running systems. All the events are reported to and stored in an Event Database (ED). The events are reported in a standard format. The format includes information about event type, address of event source, importance of the event, event generation time and some other relevant information.

The event Id defines a globally unique id for the event. The event may be a service request event, service failure event, security event, status event, performance event etc.

The event source describes the source of the event. This can be the address of the event source or any other unique identifier of the event source.

The reporter id is the id of event detector that detects the event and reports the event to the event database. Event detectors can be the web service, client of a service, or the event processor itself.

The event priority field describes the priority of the event. The higher number signifies the more important event whereas the low numbers mean less important event.

The report time is the time when the event occurred.

The Event description is a general description of the event.

Event Id	Event Source	Reporter Id	Event Priority	Report Time	Event Description

Fig. 3. General format of an Event

3.3 Event Generator (EG)

An Event Generator (EG) reads events from event database and converts these events in XML format and sends these events to Event Router (ER). The XML format of an event is shown figure 4.

```
<event>
        <eventId>11</eventId>
        <eventSource>127.0.0.1</eventSource>
        <reporterId>127.0.0.1</reporterId>
        <eventPriority>5</eventPriority>
        <reportTime>234233445534</reportTime>
        <eventDescription>service request</eventDescription>
</event>
```

Fig. 4. Event in XML format

3.4 Event Router (ER)

Event Router has connection with a number of Event Processors (EP). When an EP connects to the ER, the router sends case scenarios to the EP. The case scenarios define different cases of patterns of events. One case scenario file can contain many scenarios. Case Scenario files are in XML format. ER reads the XML file and converts the XML file into instances of scenarios. The scenario has two main elements. It has a sequence of events and a list of actions. When any pattern is matched in the event stream by the event processors the actions will be executed.

The following figure shows a sample scenario file in XML format.

```
<casescenarios>
<scenario id="1">
    <event>
        <eventId>11</eventId>
        <reportTime>12345678</reportTime>
        <reporterId>192.168.0.3</reporterId>
    </event>
    <event>
        <eventId>12</eventId>
        <reportTime greaterThan="yes">12345989</reportTime>
        <reporterId>192.168.0.4</reporterId>
    </event>
    <event>
        <eventId>13</eventId>
        <reportTime greaterThan="yes">12346789</reportTime>
        <reporterId>192.168.0.3</reporterId>
    </event>
    <event>
        <eventId>14</eventId>
        <reportTime greaterThan="yes">12347000</reportTime>
        <reporterId>192.168.0.3</reporterId>
    </event>
    <action>
        <showMessage>event 11 12 13 14 matched</showMessage>
    </action>
</scenario>
</casescenarios>
```

Fig. 5. Case Scenario in XML format

When the events stream come to the ER, ER distributes the events to the EPs based on the case scenarios the EP can process.

3.5 Event Processor (EP)

The Event Processor (EP) searches the event stream to find any match in the patterns. At first ER sends the case scenarios that EP can work on. When any scenario is matched then the EP executes the actions that are described in the action part of the case scenario. This action can be to send a report to the management application or to generate a new event and report to ED.

3.6 Algorithm

According to the proposed architecture events will be stored in the event database. The event generator reads the events from the event database and writes them in files in XML format. Then event generator sends them to the event router. The event router will send the events to the event processors that subscribe to the events. When event processors subscribe to event router for events they send the names of the required event sources.

The event processor receives case scenarios from event router and stores the scenarios in a vector. When an event arrives for checking event processor checks the scenario vector to match the event with an event in the case scenario. So the size of the scenario vector plays a role to determine

the runtime. As multiple faults can occur in the system, event processor has to check all the scenarios of the scenario vector. As a result, when the number of case scenarios increases the event processing time increases linearly. Processing of one event does not affect the processing time of the next event. For a constant number of case scenarios processing time of events has linear relation with the number of events. More events will take more time for processing. If the number of events is n and the number of case scenarios is m then the complexity of the algorithm is O(mn).

```
EventRouterThread :

        getSourceNamesFromEventProcessor();
        storeSourceNamesInHashtable();
        sendCaseScenariosToEventProcessor();

        while(true)
        {
                getEventFromEventObjectGenerator();
                sendEventToEventProcessor();
        }
```

Fig. 6. Pseudo code of Event router

```
EventProcessor :

        sendSourceNamesFromEventProcessor();
        getCaseScenariosFromEventRouter();

        while(true)
        {
                getEventFromEventRouter();
                matchEventWithScenarios();
        }
```

Fig. 7. Pseudo code of Event Processor

4 PRELIMINARY EXPERIMENTS AND RESULTS

To implement the event processing system some simple WS are developed. The clients invoke the simple WS. During the runtime of the system different events are generated by the clients and the web services. These events are recorded in the event database.

The simple WS are created using JAX-WS 2.0 with the Java SE 6 platform. JAX-WS is the Java API for XML Web Services. JAXWS version 2.0 is the center of the API stack for WS. It includes Java Architecture for XML Binding (JAXB) 2.0 and SOAP with Attachments API for Java (SAAJ) 1.3.

The event database is implemented in MySQL Server 5.0. JDBC technology is used to connect to the database. JDBC technology is an API (included both in J2SE and J2EE) that provides connectivity with the databases. I used MySQL Connector/J 3.1 as the JDBC driver to implement the connection between the database and J2SE platform. Event generator, event router and event processors are implemented in Java programming language.

I ran the setup in different machines. The Event Router ran in Hewlett-Packard HP xw6400 Workstation, Intel Xeon CPU 5140 @ 2.33GHz, 2.0 GB of RAM @ 1.98 GHz. The operating system was Microsoft Windows XP, Professional Version 2002 with service pack 2. The event database and Event Generator ran in Microsoft Virtual PC 2007 with Microsoft Windows XP, Professional Version 2002 with service pack 2, and Intel Xeon CPU 5140 @ 2.33GHz, 512 MB of RAM @ 2.29 GHz. The clients and Event Processor ran from different machines with different configurations.

4.1 Average processing time of events

In this experimental data the processing time of event is shown. The calculated time was the time between generation time of event by event generator and processing time of event by event processor. During the experiment the number of fault scenarios and the number of event processors were kept unchanged.

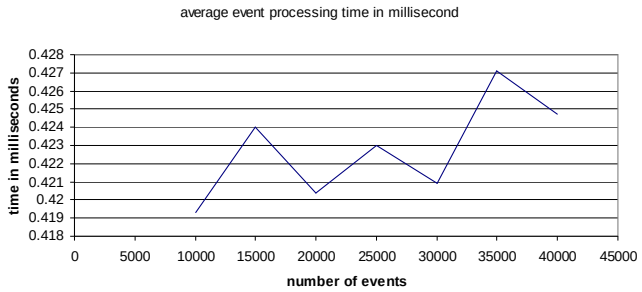

Fig. 8. Processing time of events for different total number of events. Y axis indicates time in milliseconds per event and X axis indicates number of events

From the presented data it can be inferred that when the number of events incease the average processing time remains nearly same. There is not much deviation in average processing time.

4.2 Processing time for different number of fault scenarios

In this experiment the number of fault scenarios was changed but the number of event processors was constant. In the fault scenario file at first there was 10 case scenarios. Then in successive experiments the number of case scenarios was increased. Each time 2 new scenarios were added.

From the experiment data we can see that the processing time is higher when there are more scenarios in the scenario file. According to the graph of figure 9, in case of 10 case scenarios the event processing time is minimum and in case of 18 scenarios the event processing time is maximum. So it is evident that when there are more scenarios to match the system takes more time to process the events.

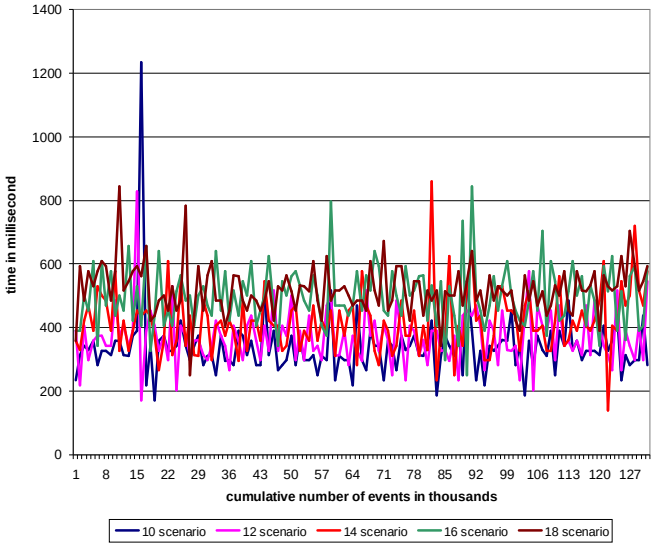

Fig. 9. Processing time of events for different number of scenarios

4.3 Processing time for different number of event processors

This experiment is done to find the impact of different number of Event Processors on the system. In this experiment the number of events and the number of fault scenarios in the fault scenario file were kept as constants. The number of event processors was changed during the experiment.

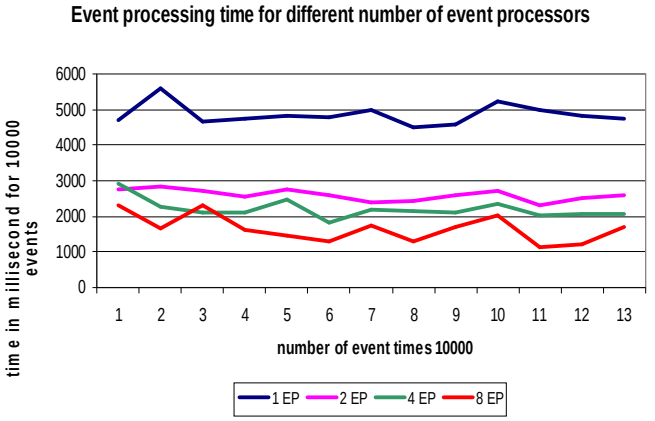

Fig. 10. Processing time of events per 10000 for different number of event processors

From the output we can see that if there is only one event processor in the system then it takes maximum time to process the events. When two event processors are used the case scenarios are distributed evenly between the two event processors. As a result there is a significant improvement in the performance. The processing time is even better for four and eight event processors. So the more the number of event processors the better the performance of the system for processing events.

The use of more event processors improves reliability of the system because if one event processor goes down then other processors will process the events

4.4 Effect of ESB on service time

Introduction of ESB increases the delay in service. Without the ESB the client could send a service request directly to the service provider. But now clients send service request to the ESB. Then ESB sends the request to the service provider.

TABLE 1
RESPONSE TIME OF WS WHEN EVENT DATABASE AND WS ARE IN SAME MACHINE AND ESB IN ANOTHER MACHINE

WS Id	Time taken by WS in ms	Time taken by ESB in ms	Extra time due to ESB
1	113.03	524.53	411.49
2	112.74	526.23	413.51
3	112.01	516.05	404.03
4	109.09	524.56	415.47

TABLE 2
RESPONSE TIME OF WS WHEN EVENT DATABASE IN ONE MACHINE AND ESB AND WS ARE IN DIFFERENT MACHINE

WS Id	Time taken by WS in ms	Time taken by ESB in ms	Extra time due to ESB
5	164.00	447.29	283.29
6	150.65	419.96	269.30
7	151.26	421.11	269.84
8	146.18	410.51	264.32
9	152.17	416.58	264.40
10	147.65	408.43	260.78

From table 1 and 2 we can see that, if ESB and WS are in same machine then response time is significantly better. In both cases the event database and ESB are in different machines. The use of ESB increases response time but shields the real services from malicious clients.

5 CONCLUSIONS AND FUTURE WORKS

In this work a new approach for monitoring of web services has been presented. This approach uses distributed cased based event processing. The event processing applications in the implemented system can detect and process the events to help monitoring and management of the services successfully.

The system is scalable since all the components are stand alone running processes. So in the system new event processors, event routers and event generators can be easily included without stopping the running system.

The fault management system is reliable. As the event processors are distributed over the internet, single component failure will not stop running the management system. This improves the reliability of the fault management system.

The processing load can be made balanced among the event processors. All the event processors do not need to process all the events. One event processor will get only the events it has subscribed for. One event processor can look after the faults of some specific web services instead of monitoring and managing all web services.

The event processor needs to know the case scenarios related to the web services it is monitoring. The case scenarios are stored in XML formatted file. The creator of the case scenario file does not need to know the all possible case scenarios of all the web services. He has to know only case scenarios related to the WS he is trying to manage. Anyone who has knowledge about creating XML files can create case scenario file and he does not need to have programming skills.

There are some limitations in the implemented experimental setup. This work assumed that the database and the event router are stable systems. But this is not always true in a real world scenario. Maintaining a stable database is out of scope of this research paper but ensuring stable ER is an important issue in this area.

JMX technology provides tools for building distributed, web based, modular and dynamic solutions for monitoring managing devices, applications, and service driven networks. In this paper JMX technology was not used. But in our future work we want to incorporate the use of JMX technology.

In the future I also want to work on improving the analyzing capability of the event processing application. Then I want to use this approach to predict the future behavior of the services.

REFERENCES

[1] M. P. Papazoglou: "Web services: Principles and technology", ISBN 9780321155559, Prentice Hall.

[2] M. P. Papazoglou, "Extending the service oriented architecture", Business Integration Journal, February 2005

[3] T. Mehta, "Adaptive Web Services Management Solutions", Enterprise Networks and Servers, vol. 17, no. 5, May 2003, available at http://www.enterprisenetworksandservers.com/monthly/toc.php?4

[4] D. Kakadia et al., "Enterprise Management Systems: Architectures and standards", Sun Microsystems, April 2002, available at: http://www.sun.com/blueprints/0402/ems1.pdf

[5] SNMP specification is available at: http://www.sei.cmu.edu/str/indexes/references/SNMPv2_Specs.html

[6] CIM specification is available at: http://www.dmtf.org/standards/cim/cim_spec_v22

[7] JMX technology home page is available at: http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement

[8] MUWS documentation is available at: http://docs.oasisopen.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf

[9] MOWS documentation is available at: http://www.oasisopen.org/committees/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf

[10] I. Sedukhin, "Web Services Distributed Management: Management of Web Services(WSDM-MOWS) 1.0", OASIS-Standard, March 2005, available at http://docs.oasisopen.org/wsdm/2004/12/wsdm-mows-1.0.pdf

[11] J. Murry, "Designing Manageable Applications", Web Developer's journal, October 2002, available at http://www.webdevelopersjournal.com/articles/design_man_app/

Sazedul Alam has been serving as a Lecturer in the Department of Computer Science and Engineering (CSE), University of Asia Pacific (UAP). He joined at UAP in April 2010. He completed his M.Sc. in computer science in 2009 from university of Saskatchewan, SK, Canada. He completed BSc. in computer science and engineering from Bangladesh University of Engineering and Technology (BUET) in 2005.