Performance Evaluation of WBEM Implementations

Sun-Mi Yoo, James Won-Ki Hong
Dept of Computer Science and Engineering
POSTECH
Pohang, Korea
{sunny81, jwkhong}@postech.ac.kr

Jong-Geun Park , Chang-Won Ahn, Seong-Woon Kim
Internet Server Group, Digital Home Division
ETRI
Daejeon, Korea
{queue, ahn, ksw}@etri.re.kr

***Abstract*—We have been developing an Internet server and have decided to use WBEM as its management framework. Rather than developing our own, we have decided to use one of the available WBEM implementations. In order to aid our decision, we have carried out performance evaluations of WBEM Services, OpenPegasus, and OpenWBEM. In order to perform fair and uniform evaluation, we have developed an integrated testing suite. In this paper, we present the design and implementation of our integrated testing suite as well as the performance evaluation results.**

Keywords-WBEM, Performance Evaluation, Testing Suite, WBEM Services, OpenPegasus, OpenWBEM.

I. INTRODUCTION

In the past decade or so, there has been a tremendous growth in the deployment and use of networked computers and applications in enterprises, businesses and our daily lives. The size and complexity of managing these computing and networking resources have also increased considerably. Effective operations and management of these resources has become an essential part of IT departments everywhere. The Distributed Management Task Force (DMTF) [1] is the technology industry organization leading the development of management standards for distributed desktop, network and enterprise environments. DMTF has been standardizing the Web-Based Enterprise Management (WBEM) architecture [2], which is using the Common Information Model (CIM) [3] for information modeling, Extensible Markup Language (XML) [4] for management information encoding and HTTP [5] for the transport protocol.

WBEM is a set of management and Internet standard technologies developed to unify the management of distributed computing environments, facilitating the exchange of data across disparate technologies and platforms. WBEM defines management information in CIM, encodes the message in CIM-XML and transports it using the HTTP protocol. WBEM also includes additional technologies such as CIM Query Language [6], WBEM Discovery using SLP [7] and WBEM URI mapping [8]. Many vendors such as Microsoft, Sun, and Cisco have been equipping their products with WBEM agents to enable their manageability.

Moreover, WBEM is provided in the form that can be easily and interoperable managed by various vendors' network and systems management system solutions. In the case of IBM, WBEM is provided by its representative network and systems management solution suite called Tivoli. It includes "Tivoli NetView", "Tivoli Enterprise", "Tivoli Manager for IBM Nways" and "Tivoli Cross-Sight" [9]. Cisco utilizes CIM in the enterprise network management tool called CiscoWorks2000 [10]. Sun provides Solaris WBEM Service and Sun WBEM SDK [11] for its Solaris Operating Environment. HP provides the WBEM solutions that include WBEM Services, WBEM Providers, HP WBEM Client and HP WBEM SDK [12]. There also exist several WBEM implementations such as OpenPegasus [13], WBEM Services [14], OpenWBEM [15], WMI [16] and SNIA CIMOM [17], some of whose source code is publicly available.

The Electronics and Telecommunications Research Institute (ETRI) in Korea has been developing an Internet server and has decided to use WBEM as its management framework. Rather than developing our own WBEM implementation from scratch, we have decided to use and enhance one of the available implementations. One of the key selection criteria was the performance of the implementation. The WBEM implementations we have decided to test are Open Pegasus [13], WBEM Services [14], and OpenWBEM [15]. In order to perform fair and uniform evaluation, we have developed and integrated testing suite. In this paper, we present the design and implementation of our integrated testing suite as well as the performance evaluation results of these WBEM implementations.

The remainder of this paper is organized as follows. Section II presents an overview of WBEM. Section III presents the design and implementation of our integrated testing suite. Section IV presents the performance evaluation results of open source WBEM implementations. Finally, we summarize our work and discuss future work in Section V.

II. WBEM-BASED ENTERPRISE MANAGEMENT

In this section, we briefly explain the technologies that are the basis of WBEM. Then, we introduce open source implementations of WBEM.

A. Overview of WBEM

WBEM is an initiative of DMTF and it includes a set of technologies that enables the interoperable management of an enterprise network. DMTF has developed a core set of standards that make up WBEM, which includes a data model, the CIM standard; an encoding specification, CIM-XML encoding specification; and a transport mechanism, CIM operations over HTTP.

The CIM specification is the language and methodology for describing management data. CIM is an object-oriented schema for modeling the objects. The CIM schema can be divided into three areas; the core model, the common model and the extension model. First, the core model captures notions that are applicable to all areas of management. Second, the common model is an information model that captures notions that are common to a particular technology. For example, it includes the model for systems, applications, networks and devices. Last, the extension model represents technology-specific extensions of common models.

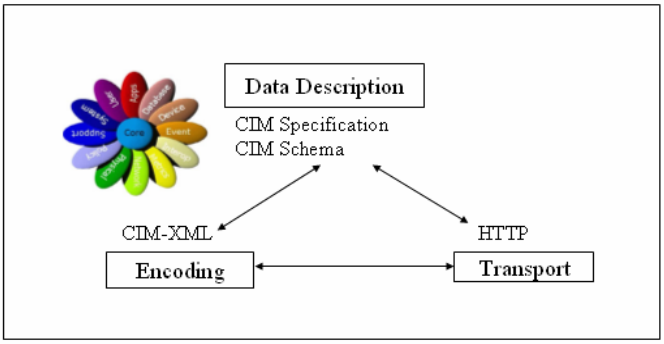

Figure 1. Relationships among WBEM Standard Technologies

The CIM-XML encoding specification defines XML elements, written in Document Type Definition (DTD) that can be used to represent CIM classes and instances. The CIM operations over HTTP specification defines a mapping of CIM operations onto HTTP that allows implementations of CIM to interoperate in an open, standardized manner and completes the technologies that support WBEM. Therefore, in the WBEM architecture, the management information is described by the CIM schema, converted to XML, and finally embedded in an HTTP payload to transport to the target node.

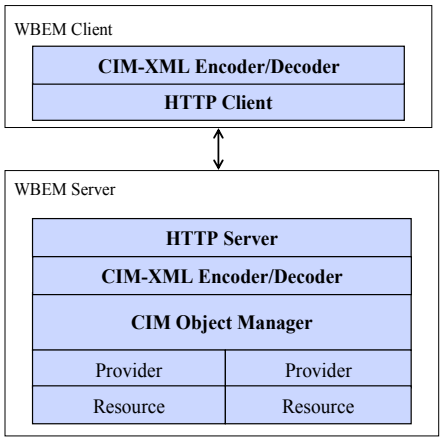

Figure 2. WBEM Architecture

Figure 2 illustrates the WBEM architecture, which includes a WBEM Client, and WBEM Server. WBEM Server has CIM Object Manager (CIMOM) which is a central component that routes objects and events between components. The CIMOM responds to the operations defined in the "CIM operations" specification such as create, modify, and delete. It also checks the syntax and semantic of the messages, and provides security. Providers are also called instrumentation agents. A WBEM Client is commonly represented as the management application, and it can get the information by sending a request message to the CIMOM instead of directly accessing the providers. Providers obtain the information from the resources and forward it to the CIMOM, which then forwards it to the requested client.

B. WBEM Implementations

This subsection presents a brief overview of several WBEM implementations.

1) OpenPegasus

OpenPegasus [13] is an open source implementation with reference to the technology of the WBEM architecture such as the CIM and CIM-XML standard. The Open Group has been developing OpenPegasus, and it is actively participating in the WBEM standardization. OpenPegasus set its goal to providing many functions and being used by many servers. Many vendors are especially interested in OpenPegasus over other WBEM implementations because it possesses three desirable characteristics. Its first characteristic is that it has good portability because it is implemented to be suitable for multiple platforms and multiple programming languages. Second, it is a lightweight and efficient implementation because of its great regard for execution performance. Third, OpenPegasus' core functions are very well modularized.

OpenPegasus' CIMOM supports the functions to create, modify and delete a class, object, property, qualifier, etc. It also provides the functions that register each provider and deliver the client's request to the provider along with a check for syntactic and semantic error. Each provider in OpenPegasus is defined in a class, which has its own method that is equivalent to the CIM operation. A provider manager converts the message delivered to the provider to a form that the provider can use. In addition, multiple language providers can be registered to the provider manager such as a C++ provider, a Java provider and so on.

2) WBEM Services

WBEM Services [14] is developed by Sun Microsystems and implemented in Solaris. WBEM Services has Java-based CIMOM and offers the Java API for a client and provider. When a client tries to get or modify information of a management object through the client API, the CIMOM responds to the client's request by interacting with the CIM repository or the related provider. The CIMOM also performs syntactic and semantic verification. The syntactic verification is to discover errors such as the omission of a semicolon or a brace, and the semantic verification is to find any logical errors in the program. The client application can send the objects to the CIMOM through the client API, and can request WBEM operations such as the creation and deletion of classes and instances in the CIM namespace.

The WBEM Services' providers exist as MOF files. These providers represent the systems, processes and resources such as CPU cycles, memory and so on. After the MOF files of each provider are compiled to a class by the MOF complier which is supported by the WBEM Services, each provider can then be

available to the client. WBEM Services also provide a security service such as authentication and authorization. For users to be authenticated, the client users need to provide the user ID and password for the WBEM server. Also the WBEM server has an Access Control List (ACL) which has a list of users that allow to access to the WBEM server.

3) OpenWBEM and SNIA CIMOM

OpenWBEM [15] is an open source implementation driven by Center7 and is C++-based. OpenWBEM supports the most number of CIM operations. Also, OpenWBEM has the advantage that it has a smaller repository than OpenPegasus. However, the OpenWBEM implementation appears to have less industrial interest at this point.

SNIA CIMOM [17] is developed by SNIA (Storage Networking Industry Association). SNIA CIMOM is a JAVA daemon and has no repository. SNIA CIMOM has had almost no research activity after 2002 and it's not clear weather they have abandoned further development. Thus, we decided not to use SNIA CIMOM and did not include it in our performance testing. .

III. INTEGRATED TESTING SUITE

In this section, we describe the design and implementation of an integrated testing suite, which has been used to obtain the performance of WBEM implementations. We first analyze the requirements of an integrated testing suite and then present the design and implementation details.

A. Requirements Analysis

The following are the functional and nonfunctional requirements for an integrated testing suite.

- The integrated testing suite should provide a black box testing without knowledge of the internal workings of the WBEM being tested.
- The integrated testing suite should be able to communicate with any WBEM implementation. Thus, it should be able to send a request message to a WBEM server and then receive the response message.
- The integrated testing suite should provide a profiling function to carry out performance.
- The integrated testing suite should provide an API that can be used by various applications for general as well as specialized testing if necessary.

B. Design Architecture

Here, we describe the architecture of our integrated testing suite. Figure 3 illustrates the internal and communicate architecture of our integrated testing suite. The integrated testing suite consists of several components, which are a main application, a common Client API, a message generator, a CIMOM HTTP Server, and a profile generator.

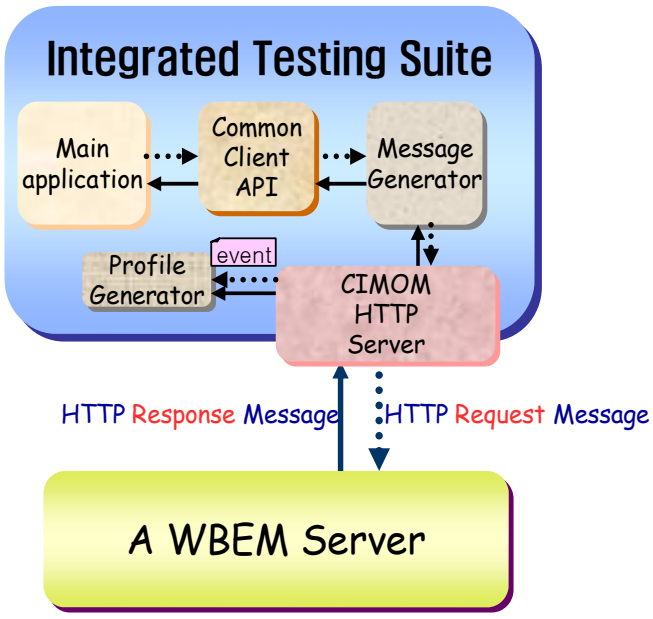

Figure 3. Architecture of integrated testing suite

The main application is used by an administrator to input necessary parameters which are required to perform CIM operations. Examples of such parameters are CIM Object Path, User ID, and password. The inputted parameters are forwarded to the message generator through the common Client API. The Common Client API provides object-oriented features and makes other applications to use without knowledge of the internals. An application can get or modify information of a management object through the Common Client API. An application can also connect to any WBEM Server with only information of the WBEM Server location through the Common Client API. The Common Client API provides all CIM operations which are standardized by DMTF. The flowchart shown in Figure 4 describes the steps involved in executing a CreateInstance CIM operation through the Common Client API as an example.

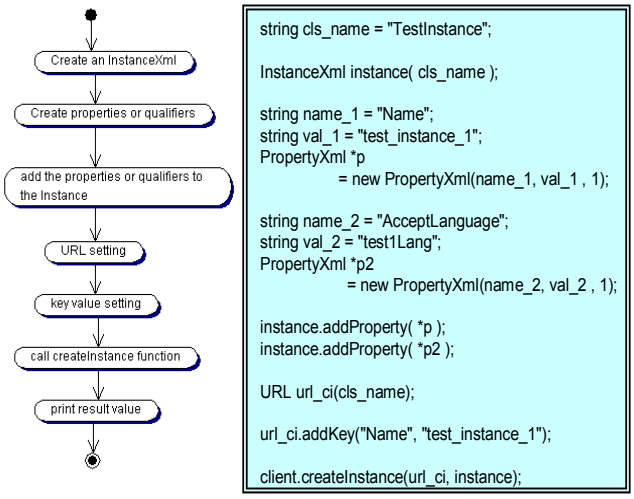

Figure 4. Flowchart of CreateInstance operation

The inputted parameters are used to create a CIM-XML request message by the message generator. Then the CIM-XML request message is added in payload of CIM Operations over HTTP request message by the CIMOM HTTP server. This CIM operation over HTTP was explained in Section II. The CIMOM HTTP Server uses several Internet protocol API (e.g., Curl, WinHTTP, URL, HTTP) to communicate with a WBEM Server. The CIMOM HTTP Server performs initial setup with information such as user id and password before sending CIM operations over HTTP request message. That is, the CIMOM HTTP Server provides a connection-oriented mechanism. Thus, a channel which is created between the CIMOM HTTP Server and a WBEM Server is maintained before sending a close request message from the CIMOM HTTP Server.

The generated request message is transferred to the WBEM Server by the CIMOM HTTP server. At this time, the CIMOM HTTP server sends an event message to the Profile Generator for checking run time. The Profile Generator records the request time and response time when it receives an event message from the CIMOM HTTP Server. The event message is sent regardless of whether the response message is due to error or not because the execution time of the operation does not have any connection with the result's accuracy. The Profile Generator provides the round-trip time (RTT) of the CIM Operation message.

When a WBEM Server receives a request message, it performs the requested operation. The WBEM Server then generates the result into a CIM Operations over HTTP response message and transfers the response message to the CIMOM HTTP server of the integrated testing suite. When the CIMOM HTTP server receives the response message, it sends an event message to the Profile Generator for checking the time called response time again.

Table I shows messages between the integrated testing suite and a WBEM Server. TABLE I(a) describes a HTTP request message which is created by operating a flowchart shown in Figure 4. TABLE I(b) describes a HTTP response message which is resulted by processing the request message by the WBEM Server.

TABLE I. HTTP REQUEST/RESPONSE MESSAGE

(a) HTTP Request Message Example

```
M-POST /cimom HTTP/1.1
HOST: localhost
Content-Type: application/xml; charset="utf-8"
Content-Length: 220
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
<MESSAGE ID="4711" PROTOCOLVERSION="1.0">
<SIMPLEREQ>
<IMETHODCALL NAME="CreateInstance">
<LOCALNAMESPACEPATH>
<NAMESPACE NAME="root"></NAMESPACE>
<NAMESPACE NAME="cimv2"></NAMESPACE>
</LOCALNAMESPACEPATH>
<IPARAMVALUE NAME="NewInstance">
<INSTANCE CLASSNAME="TestInstance">
<QUALIFIER NAME="Description" TYPE="string"
TRANSLATABLE="true" >
<VALUE>Used to test instance provider.</VALUE></QUALIFIER>
<PROPERTY NAME="Name" TYPE="string">
<VALUE>test_instance_1</VALUE></PROPERTY>
<PROPERTY NAME="AcceptLanguage" TYPE="string">
<VALUE>test1Lang</VALUE></PROPERTY></INSTANCE></IPARA
MVALUE>
</IMETHODCALL></SIMPLEREQ>
</MESSAGE></CIM>
```

(b) HTTP Response Message Example

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset="utf-8"
Content-Length: 71
Ext:
Cache-Control: no-cache
<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
<MESSAGE ID="4711" PROTOCOLVERSION="1.0">
<SIMPLERSP><IMETHODRESPONSE NAME="CreateInstance">
<IRETURNVALUE>
<INSTANCENAME CLASSNAME="TestInstance">
<KEYBINDING NAME="Name">
<KEYVALUE VALUETYPE="string">test_instance_1</KEYVALUE>
</KEYBINDING></INSTANCENAME></IRETURNVALUE></IMETH
ODRESPONSE></SIMPLERSP></MESSAGE></CIM>
```

IV. PERFORMANCE EVALUATION

A. Measurements Metrics and Methods

In this section, we present the performance evaluation results of the tested WBEM implementations, namely the OpenPegasus, WBEM Services and OpenWBEM. The test machine on which we installed three WBEM implementations was a Linux server (kernel v2.6.9) with a Pentium IV 1.6GHz CPU and 256 MB RAM. To get only the WBEM Server execution time, we installed our integrated testing suite and WBEM servers on the same machine and they communicated via HTTP. We used our integrated testing suite as a WBEM Client to perform the tests. If we measure a WBEM Server's performance with its own respective WBEM Client, the WBEM Client's communication mechanism may affect the measured value. Thus, we have solved this issue by using our integrated testing suite for testing all three WBEM implementations. For fair testing, we have used the providers which perform same functions in each WBEM implementation. These providers are default providers which do not provide special functions and get static values of CIM objects from their repository.

Within this study, many operations were performed on the initialized WBEM Server which does not perform any CIM operations. For the statistical significance, we have measured the running time and memory usage of a WBEM Server for each CIM operation 10 times and averaged the results. We measured running time using the `gettimeofday()` system call function. We measured the running time between the integrated testing suite and a WBEM Server and the time is provided by the integrated testing suite. We also measured the processing overhead of the WBEM Server in terms memory usage. We measured the memory usage by investigating 'status' and 'statm' file included in the '/proc' directory and processor ID (pid) number.

We have performed three types of tests. First, we evaluated the running time and memory usage of all CIM operations

standardized by DMTF. Each CIM operation is performed on an initialized WBEM Server. For the second experiment, we increased the number of classes 10 to 1000 by adding ten or a hundred of classes and the number of instances 10 to 250 by adding ten instances in a class. Then we performed the EnumerateClasses and EnumerateInstances operations. Lastly, we measured the running time to compare binary XML structured repository and XML structured repository in OpenPegasus. In this test, we measured the running time and the memory usage of each WBEM Server which has different structure repository as increasing the number of Classes. This test was performed by operating the EnumerateClasses CIM operation.

B. Performance Evaluation

As explained in the previous section, we have measured the running time and memory usage. In this section, we show the performance evaluation results of each test.

1) Testing 1: Performing each CIM Operation

For this experiment, we measured the running time and memory usage while performing each CIM Operation on three WBEM implementations. We limited the number of CIM objects like classes and instances within 10 in this test because the difference in the number CIM objects may be affect the measured values. Therefore, we performed the test using the static number of CIM objects. Each measurement was repeated 10 times and TABLE II shows the average values of running time that was measured on each WBEM Server.

TABLE II. RUNNING TIME (IN MSEC)

	WBEM Services	Pegasus	OpenWBEM
EnumerateQualifier	48.130	23.790	13.406
Enumerate Classes	43.284	17.630	45.453
Enumerate Instances	166.973	21.604	15.147
Get Qualifier	390.557	12.586	11.385
Get Class	11.919	12.845	11.309
Get Instance	15.282	17.453	12.838
Set Qualifier	4936.217	12.698	13.954
Create Class	96.484	12.826	15.729
Create Instance	47.908	12.583	8.141
Modify Class	50.275	9.998	8.907
Modify Instance	37.246	9.523	5.863
Delete Qualifier	4366.599	8.897	7.822
Delete Class	17.887	9.255	5.850
Delete Instance	26.808	9.395	5.168
Call Method	21.006	10.705	12.481
Get property	21.207	20.365	12.037

As shown in TABLE II, we have discovered that WBEM Services running time yielded the longest among the three WBEM Servers. Especially, WBEM Services shows even worse performance when it performs CIM operations that are related to modifying a CIM object such as objection creation, modification and deletion. On the other hand, OpenPegasus and OpenWBEM show similar running time for every CIM operations. OpenWBEM shows a slightly better performance than OpenPegasus for most operations. However, the Enumerate Classes operation running time is some different with these results.

The result that WBEM Services performance is worst is also shown similarly in the memory usage. TABLE III shows memory usage that was measured simultaneously with measurement of running time. The values in TABLE III are also average values of measurement repeated 10 times. The difference of memory usage in each CIM operation is very little.

TABLE III. MEMORY USAGE (KB)

	WBEMServices	Pegasus	OpenWBEM
EnumerateQualifier	23047.2	7087.6	6028
Enumerate Classes	23513.2	7135.2	8303.2
Enumerate Instances	23054	7331.2	6370.8
Get Qualifier	23287.2	7081.6	6011.6
Get Class	23466.8	7081.5	6039.2
Get Instance	23472	7207.6	6122.8
Set Qualifier	24201.2	7103.6	5995.6
Create Class	23276	7099.2	6027.2
Create Instance	23513.2	7332.8	6146.8
Modify Class	23549.2	7143.6	6095.6
Modify Instance	23534.8	7244	6183.6
Delete Qualifier	24359.2	7126.4	6003.6
Delete Class	23395.6	7140	6040
Delete Instance	23505.6	7366.4	6091.6
Call Method	23517.2	7380.4	6336
Get property	23465.6	7218.4	6124

These measurement results are shown as difference of implementation language platform. WBEM Services is implemented in Java and OpenPegasus and OpenWBEM are implemented in C++. WBEM Services provides a convenient and easily portable API using Java, but it has the overhead of Java virtual machine processing. The operations that simply get values of CIM Objects do not show big difference with other WBEM Servers, but the operations which perform any write functions to CIM Objects show very lower performance as compared to other WBEM Servers.

2) Testing 2: Getting CIM Operations Values

For this experiment, we have increased the number of classes from 10 to 100 in intervals of 10 and from 100 to 1000 in intervals of 100, and measured the running time and memory usage. This measurement was also repeated 10 times. As illustrated in Figure 5, the WBEM Sevices' running time increased sharply as the number of classes increased, whereas the OpenPegasus' running time increased less sharply. Interestingly, the OpenWBEM's running time did not show much change as the number of classes increased. According to this result, the OpenPegasus showed the best performance in the 10-100 classes range. We have performed more experiments to get more results with more classes.

In Figure 6, the running time of OpenPegasus increased similarly with WBEM Services at 1000 classes. On the other hand, the OpenWBEM running time still shows a very little increase as compared to other WBEM Servers. As mentioned in the previous section, the difference of implementation language platforms is possibly one reason for this result. We believe another reason is the difference of operating method for each repository. OpenWBEM operates its repository as a MOF structure. On the other hand, OpenPegasus uses XML structure for its repository. Thus, OpenPegasus shows good performance

when it operates few classes, however, it shows poor performance than OpenWBEM with many classes.

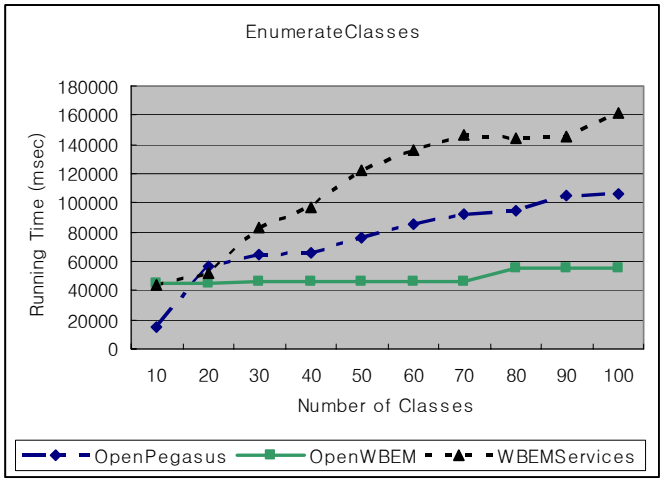

Figure 5. The Running Time (10~100)

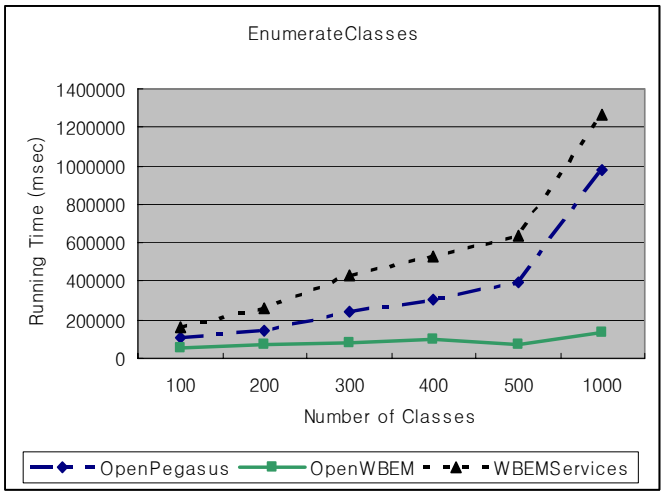

Figure 6. The Running Time (100~1000)

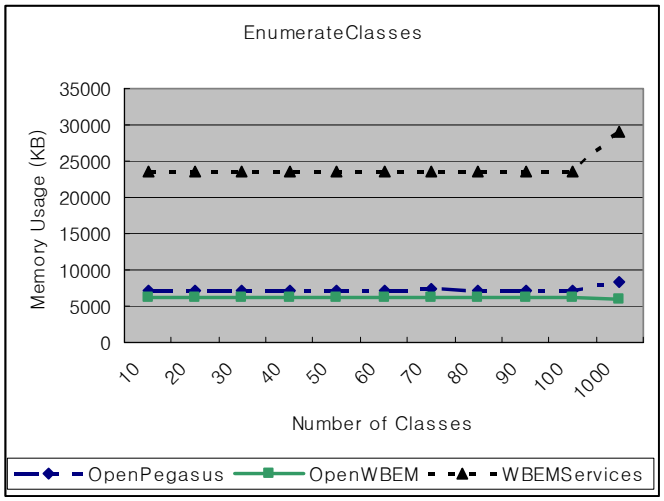

Figure 7. The Memory Usage

We have measured not only the running time but also memory usage during the same operations. Figure 7 illustrates the results. As shown in Figure 7, the memory usage does not show a seismic change as the number of classes increase.

3) Testing 3: Binary vs. XML Repository in OpenPegasus

The OpenPegasus Group recently released a new version of the OpenPegasus which has a repository encoded as binary XML. We experimented to discover some difference between a binary XML encoding repository (binary repository) and a XML structure repository (XML repository) in OpenPegasus. We installed the OpenPegasus as compiling differently in the testbed, one with XML and another with binary. In this experiment, the running time and memory usage are also measured and we repeated 10 times. TABLE IV shows the average value of running time that was measured at operating all CIM Operations.

TABLE IV. RUNNING TIME OF BINARY REPOSITORY VS. RUNNING TIME OF XML REPOSITORY IN THE OPENPEGASUS(IN MSEC)

	Binary Repository	XML Repository
Enumerate Qualifier	20.665	21.495
Enumerate Classes	18.789	18.827
Enumerate Instances	16.247	15.703
Get Qualifier	4.692	4.612
Get Class	11.810	13.502
Get Instance	3.882	9.172
Set Qualifier	9.797	4.862
Create Class	5.498	9.881
Create Instance	8.859	5.807
Modify Class	7.615	11.195
Modify Instance	10.823	10.859
Delete Qualifier	9.859	9.826
Delete Class	7.386	10.881
Delete Instance	8.193	4.931
Call Method	10.695	10.700
Get property	11.516	11.551

As shown in TABLE IV, the binary repository generally shows slightly better performance than the XML repository. Especially, the values of operations related to the class show a sensible difference. That is, the binary repository has better performance at performing operations related to the repository.

We increased the number of classes in this experiment as well. As described in Figure 8, the binary repository and the XML repository show similar results upto 100 classes. However, the XML repository increased a little more than the binary repository at 1000 classes. Before performing this test, we expected that the binary repository would show less running time than the XML repository. However, as shown in Figure 8, two types of repository resulted about the same performance. Moreover, we discovered a strange situation from this experiment. The running time decreased suddenly increasing at 90 classes and increased again. We installed a new OpenPegasus source for each test, but the results were same. Figure 9 describes the results at increasing the number of classes from 100 to 1000.

Figure 10 illustrates the memory usage increase according to the number of classes. It shows that the memory usage does not result in much difference at few numbers of classes.

However, the memory usage of the binary repository is less than the XML repository at large number of classes.

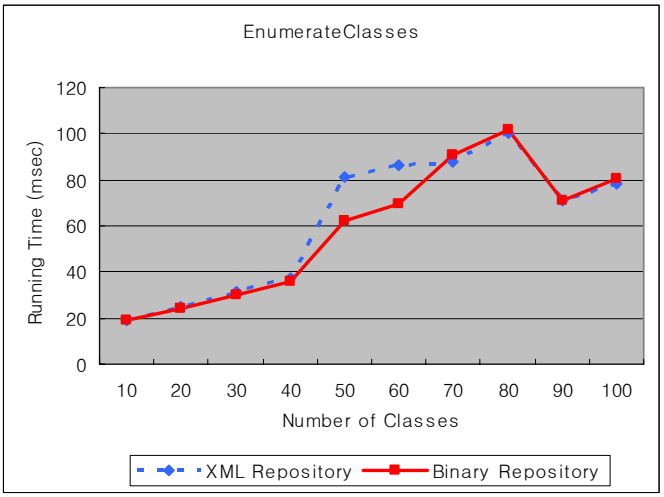

Figure 8. The Running Time (10~100)

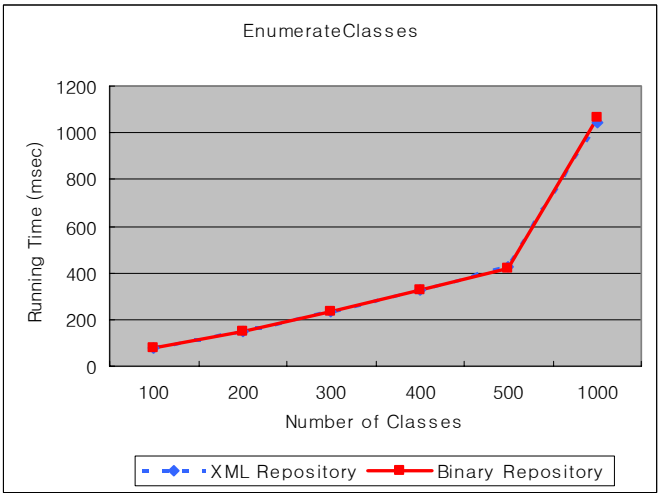

Figure 9. The Running Time (100~1000)

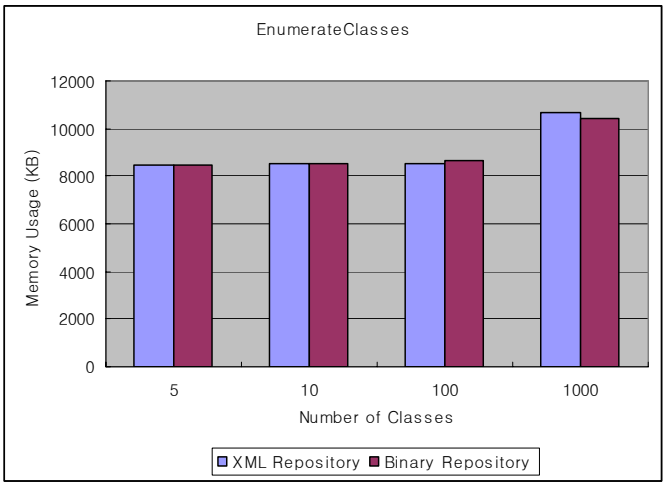

Figure 10. The Memory Usage

V. Concluding Remarks

We have presented the design and implementation of an integrated testing suite that was used to measure the performances of WBEM implementations, namely OpenPegasus, WBEM Services and OpenWBEM. We have also provided comparison analysis of their performance results.

Overall, WBEM Services performed the worst where as OpenPegasus and OpenWBEM performed relatively the same. We suspect that the reason for poor performance by WBEM Services compared to the other two is because it is implemented in Java.

The results of the performance comparison analysis have helped us in selecting the WBEM implementation that we will use for implementing our Internet server management system. We hope that the results presented in this paper can be also useful to those who are in similar situation as us as well as to those implementing WBEM implementations in the future.

The future work includes performance evaluation of these implementations using profiling tools and module testing in order to determine the bottlenecks of each WBEM server. We also plan to implement a confirmation testing suite by adding testing scenarios to our integrated testing suite.

References

[1] DMTF working group, http://www.dmtf.org.

[2] WBEM, “WBEM Initiative,” http://www.dmtf.org/standards/wbem/.

[3] DMTF, “Common Information Model (CIM),” http://www.dmtf.org/standards/standard_cim.php.

[4] Tim Bray, Jean Paoli and C. M. Sperberg-McQueen, “Extensible Markup Language (XML) 1.0,” W3 Recommendation REC-xml-19980210, Feb. 1998.

[5] DMTF, “Specification for CIM operations over HTTP Version 1.1,” DMTF Specification, January 06. 2003.

[6] DMTF, “CIM Query Language Specification Version 1.0.0”, December 9, 2004.

[7] DMTF, “WBEM Discovery using Service Location Protocol 1.0.0”, January 27, 2004.

[8] DMTF, “WBEM URI Mapping Specificaton 1.0.0”, January 27, 2004.

[9] IBM, Tivoli, http://www-306.ibm.com/software/tivoli/.

[10] Cisco, CiscoWorks2000, http://www.cisco.com/warp/public/cc/general/bulletin/wr2k/index.shtml

[11] Sun Microsystems, Solaris WBEM Service, http://www.sun.com/software/solaris/wbem/.

[12] HP, HP WBEM Solutions, http://h71028.www7.hp.com/enterprise/cache/9920-0-0-225-121.aspx.

[13] OpenPegasus, http://www.openpegasus.org.

[14] Sun Microsystems, Java Web Based Enterprise Management (WBEM Services), http://wbemservices.sourceforge.net.

[15] OpenWBEM, http://www.openwbem.org.

[16] Microsoft, Windows Management Instrumentation (WMI), http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_reference.asp

[17] SNIA CIMOM, http://www.opengroup.org/snia-cimom/.

[18] S.J. Lee, M.J. Choi, S.M. Yoo, J.W. Hong, “Design of a WBEM-based Management System for Ubiquitous Computing Servers,” Academic Alliance Paper Competition for 2004, http://www.dmtf.org/education/academicalliance, Enterprise Management World, Philadelphia USA, September 12-15.

[19] IBM, Standard Based Linux Instrumentation Manageability (SBLIM) project, http://www124.ibm.com/sblim/