OCPI

WBEM Server Benchmarks

Nortel Networks

Issue Date:	**December 3, 2003**
Revision:	**0.5**
Document Status:	**Work in Progress**
Document Number:	**ATI-2003.723**
Issue A	
Security Status:	**Open**
Authors:	**Ying Zeng, Chris Hobbs, John Bell, Brian Quirt**

Revision History

Revision Number	Date	Comments
0.1	August 26, 2003	First draft
0.2	September 09, 2003	Introduction addition, changes from the first review.
0.3	October 1, 2003	First set of result on class tests in section 6.3.1, and test script re-organization in section 5.2.2, and re-addition of property operations in section 4.2.2.
0.4	October 23, 2003	Addition of section 5.2.3, and Appendix B - D. Update of results for instance operations in section 6.3.2 and 6.4.1.
0.5	October 24, 2003	General test condition update

Contributors

ATI, BCM OAM team

Table of Contents

1 Introduction

This document describes a methodology and a set of tests that have been constructed to quantify the performance characteristics of systems that implement the CIM management model. The purpose of these benchmarks is to provide the results necessary to compare different implementations of a CIM system, as well as providing the data necessary to determine if the resource budgets for target managed elements can be met.

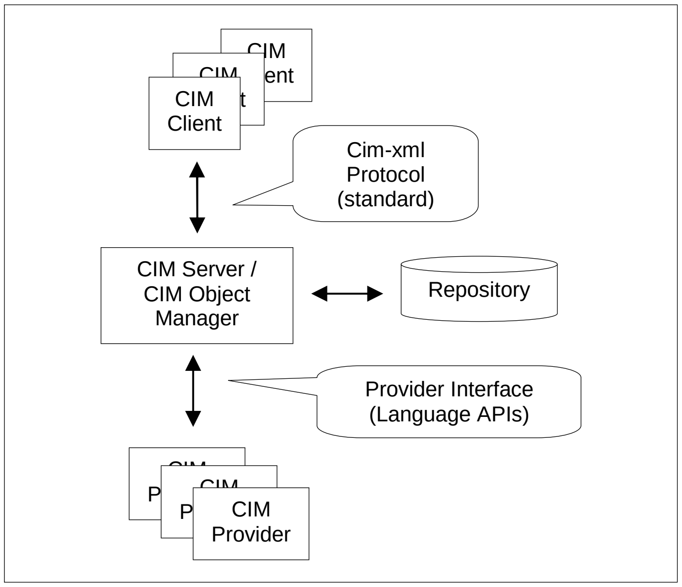

CIM Client
Cim-xml Protocol (standard)
CIM Server / CIM Object Manager
Repository
Provider Interface (Language APIs)
CIM Provider

Figure 1: Major CIM System Components

The major system components under consideration are the client/server protocol, the CIM Server (or CIMOM), the repository and the CIM providers. The instrumentation interface will reside at the CIM client.

The benchmarks will perform measurements along a number of different dimensions: the latency for a set of CIM operations (where an operation describes a management - i.e., monitor or control - action on a CIM modeled resource); the in-memory requirements for the CIM server; and the persistent-storage requirements for the repository. The performance and storage requirements will be gauged by discretely varying the size of the models and the number of instances in the system. The dimensions to be measured are seen as external attributes of the system. Treating the system as a black box has the advantage of being able to infer the properties of the system using external instrumentation interfaces. This in turn means that the observations will have negligible impact on the overall system performance while also providing the means to perform an apples-to-apples comparison of different CIM implementations. This methodology allows the various elements of a CIM system to be characterized and compared. For example, changing the size of the model and the number of instances under consideration will provide a clear indication of how efficient the object encoding is (the in-memory requirements), how efficient the object serialization is (the persistent-memory requirements) and what effect varying these dimensions has on the observed performance of CIM operations (the latency).

Functional Group	**CIM Operations**
Basic Read	GetClass, EnumerateClasses, EnumerateClassNames, GetInstance, EnumerateInstances, EnumerateInstanceNames, GetProperty
Basic Write	SetProperty
Schema Manipulation	CreateClass, ModifyClass, DeleteClass
Instance Manipulation	CreateInstance, ModifyInstance, DeleteInstance
Association Traversal	Associators, AssociatorNames, References, ReferenceNames
Query	ExecQuery
Qualifier Declaration	GetQualifier, SetQualifier, DeleteQualifier, EnumerateQualifier

Table 1: Benchmark test sets for extrinsic method operations

CIM operations can be broken down into functional groups, each supporting a number of set of method calls. The following table lists the functional groups a the operations associated with each group. The benchmarks are designed to measure the performance characteristics for all of the operations within each group. A further discrimination is made between on those operations that are performed by both intrinsic methods and extrinsic methods. (Intrinsic methods operate on the CIM model and are internal to the CIM Server while extrinsic methods are executed by a provider.) Distinguishing between intrinsic and extrinsic operations characterizes the performance overhead of invoking a provider. A final set of tests characterize the performance of indication operations – of particular interest here is what effect indication filters have on event delivery.

2 Choices of WBEM Implementations

There are several choices of WBEM server implementations for benchmarks:

- OpenPegasus

This is an open source implementation driven by IBM, HP, EMC, etc. Currently this implementation has a lot of activities and development work being done, and it attracts much industrial attention. The advantages of this implementation are that the repository is implemented in XML which is easy to debug, and that it also comes with a debug client. However, the disadvantages are that the repository is way too large, and a few CIM operations are NOT yet supported. The version used in the benchmark tests is release 2.2.

- OpenWBEM

This is an open source implementation driven by Center7. The openWBEM implementation has a better architecture than openPegasus, and most of the CIM operations are supported. It is also smaller in repository size and has a slightly better memory footprint. However, the openWBEM implementation appears to have less industrial interests at this point. The version used in the benchmark tests is release 2.0.6.

3 General Benchmark Conditions

Unless otherwise stated, all benchmark results described in this document were obtained under the following conditions:

- Platform - Debian Linux running kernel 2.4.18; if available, all benchmark tests will be performed on a PPC G4 machine.
- 100Mb/s LAN connection
- CPU speed - Pentium III 550MHz, 512KB CPU Cache, RAM size - 128MB
- WBEM client and WBEM server are running on the same computer
- WBEM client and WBEM server are running on different computers given that the computers are connected with a 100bps Ethernet connection. The Ethernet connection is shared with other traffic streams in order to simulate the internet cloud.
- OpenPegasus WBEM server version - release 2.2
- OpenWBEM WBEM server version - release 2.0.6

Characteristic	Value
Vendor_id	GenuineIntel
CPU family	6
Model	7
Model name	Pentium III (Katmai)
Stepping	3
CPU MHz	550MHz (548.630MHz)
Cache size	512KB
Fdiv_bug	No
Hlt_bug	No
F00f_bug	No
Coma_bug	No
Fpu	yes
Fpu_exception	yes
Cpuid level	2
Wp	yes
Flags	Fpu vme do pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat pse36 mmx fxsr sse
Bogomips	1094.45

Table 2: Processor Characteristics

4 Benchmark Feature Specification

The main goal of the benchmark tests is to evaluate the WBEM server performance with respect to the normal CIM operations, including latency, size of executables and shared libraries, size of the repository, CPU utilization and memory footprint.

The benchmark tests can be categorized into three groups:

1. CIM class, instance, property, association and qualifier operations on the repository,
2. CIM instance, association, property and method invocation operations on a provider,
3. process indication operations (the class indication and instance indication are not yet supported in most of the WBEM/CIM implementations), and
4. ExecQuery

Note that for each type of CIM operation, the benchmark tests are divided by groups depending on the varying factors of the tests. For example, the repository class operations can be benchmarked against three factors: number of classes in the repository, class inheritance and HTTP/HTTPS protocols. The benchmark tests are grouped according to the conditions so that only one factor is varying at one time while all others are fixed.

4.1 Repository Operations

The operations on the repository include class operations, instance operations, association operations and qualifier operations.

The repository implementations differ from each WBEM server implementations; therefore, the benchmark tests consider the overall performance of the server and the repository operations.

The following performance aspects are measured: latency of a request, size of the repository, CPU utilization, and memory footprint.

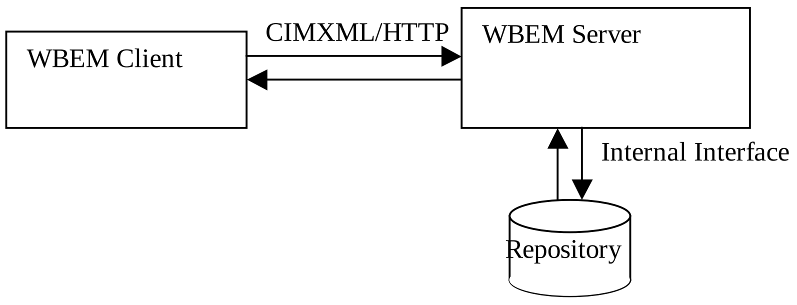


Figure 2: CIM Repository Operation Flow

4.1.1 CIM Class Operations

The class operations include five types of operations:

- GetClass

The GetClass operation retrieves the inheritance, properties, qualifiers and methods of the specified class. An exception is thrown if the class does not exist. This is possibly the only useful class operation to BCM.

- EnumerateClassNames

The EnumerateClassNames operation returns the class names excluding any qualifiers or properties within the namespace specified in the request. This operation maybe used in some cases in BCM.

- CreateClass

The CreateClass operation creates a new class in the namespace in the repository. If a class with the same class name and the version number already exists, then a CIM exception is thrown and the operation fails.

- DeleteClass

The DeleteClass operation removes a class from the given namespace in the repository. If the class does not exist in the namespace, a CIM exception is thrown.

- ModifyClass

The ModifyClass operation modifies the properties and/or qualifiers of a class. Note that some CIM implementations do NOT support ModifyClass feature yet.

Since the BCM box uses mostly GetClass and occasionally EnumerateClassNames, we will focus the benchmarks on these two operations.

The performance of the above operations is assumed to depend mainly on the number of classes within the specified namespace. Therefore, the benchmarks are tested against the number of classes in the namespace. The reasonable lower boundary for the benchmarks would be around 120 classes. This is because the new class is extended either from the CIM core models or from other standard CIM models which are extended from the core models, and thus the classes in the core models are always presented in a namespace. A reasonable upper boundary for number of classes in the repository is around 5000 classes. This number may vary depending on the complexity of the models.

The class operations are also tested using CIMXML over HTTP and CIMXML over HTTPS protocols, and the level of inheritance of a class.

The following tables capture the benchmark test groups for a class operation. The same sets of the tests are repeated for all the CIM class operations.

Benchmark tests against scales						
	Lower boundary					**Upper boundary**
Number of Classes in the repository	120	500	1000	2000	3000	5000
Benchmark tests against qualifiers						
	Lower boundary					**Upper boundary**
Number of qualifiers of a class	10	100	150	200	250	300
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of classes				HTTP		HTTPS

Table 3: Benchmark test sets for CIM class operations

4.1.2 CIM Instance Operations

There are six types of instance operations. The instance operation performance would likely be affected by the number of classes in the repository, number of instances of a class, number of keys of an instance and key types.

- EnumerateInstanceNames

The EnumerateInstanceNames operation returns the CIM object path in a string format including the namespace, class name and key property values of all instances of a requested class.

- EnumerateInstances

The EnumerateInstances operation returns all the instances of a specified class. If there is a filter condition specified in the request, only the specified properties of the instances will be returned. The combination usage of `LocalOnly` and `DeepInheritance` allows instances of the superclass(es) and/or the subclass(es) to be returned. But the benchmark tests described in this document returns only the instances of the **requested** class.

- CreateInstance

The CreateInstance operation creates an instance of a specified class in the repository. A CIM exception is thrown if there is already an instance with the same key property value or values.

- GetInstance

The GetInstance operation returns an instance with the specified key property values. All the properties of the instance are returned. This operation is tested with `IncludeQualifiers` field is set to `TRUE`.

- DeleteInstance

The DeleteInstance operation deletes an instance from the repository. A CIM exception is thrown if the instance does not exist.

- ModifyInstance

The ModifyInstance operation changes the property or properties of an instance. Note that the key properties of an instance cannot be changed and a CIM exception is thrown if one attempts to change a key property.

The benchmark tests include only GetInstance, EnumerateInstanceNames and EnumerateInstances operation as they are the only ones being used in BCM.

The instance operations are tested using the following benchmark tests:

Benchmark tests against scales						
	Lower boundary					**Upper boundary**
Number of instances of the requested class given a fixed number of classes in the	0	50	100	200	500	1000

repository						
Number of keys on an instance given a fixed number of instances of a class	1	2	4	6	8	10
Benchmark tests against key types						
Key Types	String			Integer		
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of classes and a fixed number of instances of a class	HTTP			HTTPS		

Table 4: Benchmark test sets for CIM instance operations

4.1.3 CIM Property Operations

The property operations will not be used in BCM, and the related benchmark tests are therefore omitted.

The property operations include two types of operations:

- GetProperty

The GetProperty operation returns the requested property value of a particular instance.

- SetProperty

The SetProperty operation modifies the requested property value of a particular instance.

Benchmark tests against scales						
	Lower boundary					**Upper boundary**
Number of instances of the requested class given a fixed number of classes in the repository	0	10	50	200	500	1000
Number of properties on an instance given a fixed number of instances of a class	1	5	10	20	30	50
Benchmark tests against property types						
Property Types	String			Integer		
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of classes and a fixed number of instances of a class	HTTP			HTTPS		

Table 5: Benchmark test sets for CIM property operations

4.1.4 CIM Association Operations

The association operations will not be used in BCM, and the related benchmark tests are therefore omitted.

The CIM association operations include four types of operations.

- Associators

The Associators operation returns a list of instances associated with a particular instance or classes associated with a particular class.

- AssociatorNames

The AssociatorNames operation returns a list of names of instances associated with a particular instance or class names associated with a particular class.

- References

The References operation returns a list of associations for a particular class or an instance.

- ReferenceNames

The ReferenceNames operation returns a list of names of associations for a particular class or an instance.

Benchmark tests against scales						
	Lower boundary					**Upper boundary**
Number of association instances with fixed number of classes in the repository	0	10	50	200	500	1000
???						
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of association instances and a fixed number of classes	HTTP			HTTPS		

Table 6: Benchmark test sets for CIM association operations

4.1.5 CIM Qualifier Operations

The qualifier operations will not be used in BCM, and the related benchmark tests are therefore omitted.

The qualifier operations retrieve and set the qualifier of a class in the repository. This set of operations only applies to the repository.

- EnumerateQualifiers

The EnumerateQualifiers operation returns a list of qualifiers in the namespace.

- GetQualifier

The GetQualifier operation returns a qualifier from a given namespace.

- SetQualifier

The SetQualifier operation creates or modifies a particular qualifier in a given namespace.

- DeleteQualifier

The DeleteQualifier operation removes a qualifier from a namespace.

Benchmark tests against scales						
	Lower boundary					**Upper boundary**
Number of Classes in the repository	120	500	1000	2000	3000	5000
Number of qualifiers of the requested class with fixed number of classes in the repository	2	10	30	50	100	200
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of qualifiers on a class and a fixed number of classes	HTTP			HTTPS		

Table 7: Benchmark test sets for CIM qualifier operations

4.1.6 ExecQuery Operation

The ExecQuery operation will not be used in BCM, and the related benchmark tests are therefore omitted.

The ExecQuery operation executes a database-style query on the classes and instances.

Benchmark tests against scales						
	Lower boundary					**Upper boundary**

Number of Classes in the repository	120	500	1000	2000	3000	5000
Number of instances of the requested class given a fixed number of classes in the repository	0	10	50	200	500	1000
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of qualifiers on a class and a fixed number of classes			HTTP		HTTPS	

Table 8: Benchmark test sets for ExecQuery operation

4.2 Provider Operations

The operations on the provider are similar to those on the repository except the exclusion of class operations and qualifier operations. The normal provider operations include instance operations, association operations, method invocations and property operations. Note that the indication operation is a special type of provider operation, and it is covered in the indication benchmark test section.

The following performance aspects are measured: latency of a request, size of the repository, size of the shared libraries, CPU utilization, and memory footprint.

The provider benchmarks do not consider the specialty of the provider implementations, and thus the providers used in the tests are designed to perform NULL operation and immediately return with `OK` status.

Unless otherwise stated, all provider benchmarks are measured under the condition that the provider shared libraries are loaded prior to the tests and thus the library initial loading time is not considered.

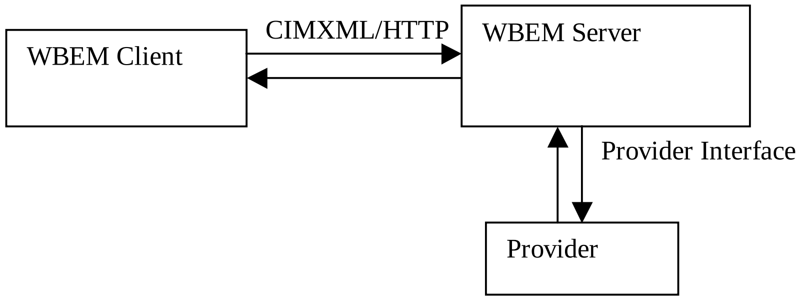


Figure 3: CIM Provider Operation Flow

4.2.1 *CIM Instance Operations*

There are six types of instance operations. The instance operation performance may be affected by the number of classes in the repository, number of instances of a class, number of keys of an instance and key types.

- EnumerateInstanceNames
- EnumerateInstances - least used
- CreateInstance
- GetInstance
- DeleteInstance
- ModifyInstance

The instance operations on a provider are similar to those of the instance operations on the repository. The instance operations are tested using the following benchmark tests:

Benchmark tests against scales						
	Lower boundary					**Upper boundary**
Number of instances of the requested class given a fixed number of classes in the repository	0	50	100	200	500	1000
Number of keys on an instance given a fixed number of instances of a class	1	2	4	6	8	10
Benchmark tests against key types						
Key Types	String			Integer		
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of classes and a fixed number of instances of a class	HTTP			HTTPS		

Table 9: Benchmark test sets for CIM instance operations

4.2.2 *CIM Property Operations*

The property operations on a provider are similar to those on the repository and include two types of operations:

- GetProperty
- SetProperty

Benchmark tests against scales						
	Lower boundary					**Upper boundary**

Number of instances of the requested class given a fixed number of classes	0	10	50	200	500	1000
Number of properties on an instance given a fixed number of instances of a class	1	5	10	20	30	50
Benchmark tests against property types						
Property Types	String			Integer		
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS given a fixed number of classes and a fixed number of instances of a class	HTTP			HTTPS		

Table 10: Benchmark test sets for CIM property operations

4.2.3 CIM Association Operations

The association operations on a provider are similar to those on the repository and include four types of operations:

- Associators
- AssociatorNames
- References
- ReferenceNames

Benchmark tests against scales						
	Lower boundary					**Upper boundary**
Number of association instances with fixed number of classes in the repository	0	500	1000	3000	4000	5000
Number of references defined in the association class	2	3	4			
Benchmark tests against protocols						
CIMXML over HTTP/HTTPS with fixed number of association instances and fixed number of classes	HTTP			HTTPS		

Table 11: Benchmark test sets for CIM association operations

According to the spec., an association class can have more than two references defined. However, the spec. does not provide descriptions on how enumerating association and reference works. In the case of a four-way association, it is unclear how the references are returned from a request of enumerating references. TODO

4.2.4 *CIM Extrinsic Method Operations*

- InvokeMethod

The extrinsic method invocation operations are tested using the following benchmark tests:

Benchmark tests against scales					
	Lower boundary				**Upper boundary**
Number input parameters in the method invocation	0	1	2	5	10
Number of output parameters in the method invocation	0	1	2	5	10
Benchmark tests against parameter types					
Types of the input/output parameters	String	Integer			
Benchmark tests against protocols					
CIMXML over HTTP/HTTPS	HTTP	HTTPS			

Table 12: Benchmark test sets for extrinsic method operations

4.3 Indications

TODO

The indication tests measures the latency and throughput of indications. The latency does NOT consider the delay of the handler to the client because the handlers are highly customized by programmers.

4.3.1 *Indication Test Setup in openPegasus*

The indication tests are performed using the RT_Indication example provided by openPegasus. The setup is shown as in the figure below:

FIGURE: TODO

4.4 Client Library XML Encoding/Decoding

This benchmark test is to evaluate the WBEM client XML encoding and decoding performance using the client library provided by each WBEM implementation.

Benchmark tests against request types		
Request Type	Simple operation	Multiple operations

Table 13: Benchmark test sets for extrinsic method operations

5 Benchmark Test Instrumentation and Implementation

5.1 Instrumentation

There are two options of the instrumentation locations. Please see the diagram below

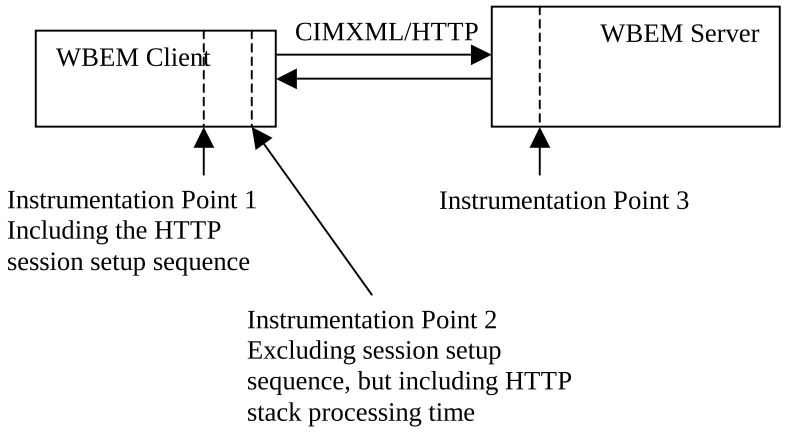


Figure 4: Instrumentation Points

5.1.1 Option 1: Instrumentation at Client Session End

The instrumentation at the client session end implies that the benchmark is the duration of the period between the HTTP or HTTPS session is setup and the same session finishes. This instrumentation provides benchmarks of overall performance of the CIM operations over HTTP/HTTPS.

5.1.2 Option 2: Instrumentation at Client Protocol End

The instrumentation at the client protocol end implies that the benchmark is the duration of the period between the CIMXML request is put onto a HTTP or HTTPS protocol stack and the CIMXML response is received. This instrumentation provides benchmarks of CIMOM operation performance without consideration of the HTTP/HTTPS session setup procedures.

5.1.3 Option 3: Instrumentation at CIMOM Server Protocol End

The instrumentation at the CIMOM server protocol end implies that the benchmark is the duration of the period between a CIMXML request is received by WBEM server and a

CIMXML response is sent by the server. This operation considers only the server operation benchmarks without any consideration of the CIMXML messaging and HTTP/HTTPS session setup procedures. This instrumentation concerns mainly the performance of WBEM server repository interface and provider interface. To work with this option, one must insert the timestamps in the corresponding CIMOM server process.

5.1.4 Option 4: Instrumentation using Binary Interface

The instrumentation can also be done using the binary interface provided by some CIMOM implementations. The binary instrumentation point is not shown in the figure, but the measurement point would be on the backend of the client side.

5.2 Implementation

5.2.1 wbemexec Utility

OpenPegasus provides a command line utility called "wbemexec". This tool reads the formatted CIMXML request from standard input, generates and sends a CIMXML request to the WBEM server, receives the CIMXML response and print the response to standard output. The wbemexec is a CIM client. Once a XML file containing a CIMXML request is created, the wbemexec utility performs the following steps. The steps in red refer to the timestamp insertion points for the benchmarks.

1. open the XML file
2. read the CIMXML request into a memory buffer
3. check for the XML syntax error
4. make a copy of CIMXML request
5. encapsulate XML into HTTP message
6. debug print the request message
- start timer at instrumentation point 1
7. connect to the WBEM server
- start timer at instrumentation point 2
8. send the CIMXML request and receive a CIMXML response
- stop timer for both instrumentation points
9. put the response into a buffer
10. format the response and write to standard output

A Perl script is written to repeatedly invoke the wbemexec utility for TODO times and the benchmark result is the average time in a single invocation.

5.2.2 Test Suite

- Introduction

The test suite is designed to allow the performance of an assortment of CIM servers to be benchmarked. Any number of tests and/or test scripts may be written to examine whatever performance aspects are desired. The scripts expect to have access to the wbemexec command-line program (a part of OpenPegasus), in particular, a version which has been modified to display how long the current request took to complete in the form

```
Operation took X.x/Y.y seconds
```

where X.x is the total number of seconds required (not including the time required to connect to the server) and Y.y is the
total number of seconds required including the time required to connect to the server.
The test suite is started by typing

```
./TestSuite.pl <Testfile>
```

at the command line, where <Testfile> is a file containing the tests to be run (its format will be discussed later). The file "Script.txt" is included as a default file for the test suite.

- Directory Structure in CVS

```
bcm/
  cimBenchmarks/
      TestSuite.pl
      <TypeOfTest>Tests.txt
      openwbem/
          <TypeOfTest>/
              <TypeOfTest>Provider.cpp
              <TypeOfTest>Provider.h
              Makefile
              ...
      Pegasus/
          <TypeOfTest>/
              <TypeOfTest>Provider.cpp
              <TypeOfTest>Provider.h
              <TypeOfTest>ProviderMain.cpp
              Makefile
              ...
      repository/
              <TypeOfTest><Action>.pl
              ...
      provider/
              <TypeOfTest><Action>.pl
              ...
      xmlRequests/
      cimResponses/
      utils/
          XmlGen.pm
```

<TypeOfTest> can be "class", "instance", "property", "association" and "method" depending on what type of CIM operation is tested. <Action> refers to "setup", "tests" or "cleanup" of the specified type of CIM operation. For instance, the test scripts for the instance operation with provider would be in directory bcm/cimBenchmark/provider,

named as ClassSetup.pl, ClassTests.pl, and ClassCleanup.pl, and the provider code would be in directory bcm/cimBenchmark/openwbem/instance. The sequence of setup, tests and cleanup would allow one to produce a number of dummy classes, instances, or qualifiers in order to achieve a certain set of discrete test points, to perform the corresponding tests and finally to do the proper cleanup.

- Series format

The TestSuite.pl file invokes a series of tests by reading a script file. The script file contains information about which tests are to be performed and is formatted as below:

```
    #lines starting with a '#' are comments
    #
    #Test Comment: <script> <command line options>
    #
    # Class setup options: <namespace> <dummyClassName>
    # <total#OfClassesInRep> <serverHostname>
    Example 1: ./repository/ClassSetup.pl root/widget DummyClass 5000
pkany216
    # Class test options: <repetitions> <namespace> <serverHostname>
    Example 2: ./repository/ClassTests.pl 100 root/widget pkany216
    # Class setup options: <namespace> <dummyClassName>
    # <total#OfClassesToClean> <serverHostname>
    Example 3: ./repository/ClassCleanup.pl root/widget DummyClass 4020
pkany216
```

Note that the test scripts are invoked just as if they were being invoked from a shell. Any arguments after the test name are passed to the appropriate test script as command line options.

- Test Script Format

Each individual test script may be in any format, as long as it adheres to the following rules:

  - It must be possible to invoke the script from the command line with any needed options
  - The test script must return an integer value which, when shifted 8 bits to the left, indicates the number of errors which occurred when running the scripts (EG if a script was run for 1000 repetitions and 4 of those repetitions resulted in errors, the script would return (4 << 8). In the case of a perl script, it would simply return 4).

The current test scripts are written in perl, and may serve as a template for any future test scripts. They make use of the XmlGen perl module (which functions as an object) to create an xml file which will be passed to wbemexec, and they then receive and interpret the results.

- Error Checking

The TestSuite.pl will, before anything is run, go through the test scripts and verify that each file exists. Note that, as a result, a full path to any script file must be provided, since the test script will not use $PATH when checking for existence.
During running, TestSuite.pl will report on how many errors were experienced by each test.

- Some Sample Scripts

These scripts are not currently used in the benchmark tests. They are template scripts for the new test scripts and located in the directory bcm/cimBenchmark/sampleScripts.

`EnumerateClassNames.pl`
Usage: `EnumerateClassNames.pl <Repetitions> <Namespace>`
Purpose: This script tests how long it takes to return a list of all classes in the current namespace

`CreateClass.pl`
Usage: `CreateClass.pl <Repetitions> <Namespace> <ClassName>`
Purpose: This script creates classes `ClassName0` to `ClassNameN` in the supplied namespace where `(N+1)` is the number of repetitions.

`CreateSubClass.pl`
Usage: `CreateSubClass.pl <Repetitions> <Namespace> <ClassName> <SuperClass>`
Purpose: This script creates classes `ClassName0` to `ClassNameN` (`N+1` is the number of repetitions) in the supplied namespace, where all of the classes inherit from the supplied SuperClass. The SuperClass must, of course, exist.

`DeleteClass.pl`
Usage: `DeleteClass.pl <Repetitions> <Namespace> <ClassName> <Numeric>`
Purpose: This script deletes the specified class in the specified namespace. If Numeric is 1, it will delete the classes `ClassName0` to `ClassNameN` (where `N+`1 is the number of repetitions). If Numeric is `0`, it will delete `ClassName`.

`GetClass.pl`
Usage: `GetClass.pl <Repetitions> <Namespace> <ClassName> <LocalOnly> <IncludeOrigin> <IncludeQualifiers> <Numeric>`
Purpose: This script retrieves information about the specified class. If Numeric is 1, it will retrieve `ClassName0` to `ClassNameN` (as per the other numeric scripts). If Numeric is `0`, it will retrieve `ClassName`. `LocalOnly`, `IncludeOrigin`, and `IncludeQualifiers` are all IPARAM's to be passed to the cimserver. They take either `TRUE` or `FALSE`, and their effect is as the effect of the equivalent IPARAM.

5.2.3 Test Providers

The tested providers include the instance provider, property provider, association provider, method provider and indication provider.

Usually, the actions inside the provider are highly customized depending on the features of the managed element. Thus, the provider processing time varies greatly from application to application. In order to obtain fair measurements of the provider interface of different WBEM implementations, we designed the providers under the following conditions.

Test condition A: The tests are to measure the CIMOM-provider interface latency, without considering the provider internal processing time. Therefore, the provided providers contain the minimum actions that are required to return a successful response or to throw an exception immediately upon an incoming request. The providers do NOT retain any instance/property/association data corresponding to the incoming request. The test results listed in this section are obtained under this test condition. The sample provider code can be found in the appendix D.

Test condition B: The providers are designed to perform minimum set of real actions. For example, the instance provider retains a local array of instances and return a valid instance upon a GetInstance request. The provider processing time is recorded and an average of the provider processing time for each method is calculated at the end of the test. The client-server roundtrip latency is measured using the total roundtrip latency minus the provider processing time. The provider may also take measurement of the memory usage and CPU usage. A `BmDataClass` is used to store the provider processing time as its properties. Note that the tests under this condition are listed as future work if effort allows.

For tests under condition B, a benchmark test class is defined to hold the latency data for the provider processing. For example, the mof file below defines such a class:

```
[Description ("Benchmark Data Class")]
class BmDataClass : CIM_System
{
    [
    Description ("For benchmark test purpose. If its value is set to 1, "
    "it marks the end of the benchmark tests, and the provider will print "
    "the elapsed time for internal provider data processing") ]
    uint32 TestProperty;

    [Description ("Elapsed time in usec for successful CreateInstance
opreations")]
    uint64 CiElapsedTime;

    [Description ("Elapsed time in usec for exceptions in CreateInstance
opreation")]
    uint64 ECiElapsedTime;

    [Description ("Elapsed time in usec for successful GetInstance opreations")]
    uint64 GiElapsedTime;

    [Description ("Elapsed time in usec for exceptions in GetInstance
opreation")]
    uint64 EGiElapsedTime;

    [Description ("Elapsed time in usec for successful EnumerateInstanceNames
opreations")]
    uint64 EnElapsedTime;
    [Description ("Elapsed time in usec for exceptions in EnumerateInstanceNames
opreation")]
    uint64 EEnElapsedTime;

    [Description ("Elapsed time in usec for successful EnumerateInstances
opreations")]
    uint64 EiElapsedTime;

    [Description ("Elapsed time in usec for exceptions in EnumerateInstances
opreation")]
    uint64 EEiElapsedTime;

    [Description ("Elapsed time in usec for successful DeleteInstance
opreations")]
    uint64 DiElapsedTime;

    [Description ("Elapsed time in usec for exceptions in DeleteInstance
opreation")]
    uint64 EDiElapsedTime;

    [Description ("Elapsed time in usec for successful ModifyInstance
opreations")]
    uint64 MiElapsedTime;

    [Description ("Elapsed time in usec for exceptions in ModifyInstance
opreation")]
    uint64 EMiElapsedTime;

    [Description ("Number of successful CreateInstance operations")]
    uint32 CiCount;

    [Description ("Number of exceptions in CreateInstance operations")]
    uint32 ECiCount;

    [Description ("Number of successful GetInstance operations")]
    uint32 GiCount;

    [Description ("Number of exceptions in GetInstance operations")]
    uint32 EGiCount;

    [Description ("Number of successful EnumerateInstanceNames operations")]
    uint32 EnCount;

    [Description ("Number of exceptions in EnumerateInstanceNames operations")]
    uint32 EEnCount;

    [Description ("Number of successful EnumerateInstances operations")]
    uint32 EiCount;

    [Description ("Number of exceptions in EnumerateInstances operations")]
    uint32 EEiCount;

    [Description ("Number of successful ModifyInstance operations")]
    uint32 MiCount;

    [Description ("Number of exceptions in ModifyInstance operations")]
    uint32 EMiCount;

    [Description ("Number of successful DeleteInstance operations")]
    uint32 DiCount;

    [Description ("Number of exceptions in DeleteInstance operations")]
    uint32 EDiCount;
};

instance of BmDataClass
{
    CreationClassName="BmDataClass";
    Name="Benchmark";
    TestProperty="0";
    CiElapsedTime=0;
    ECiElapsedTime=0;
    GiElapsedTime=0;
    EGiElapsedTime=0;
    EnElapsedTime=0;
    EEnElapsedTime=0;
    EiElapsedTime=0;
    EEiElapsedTime=0;
    MiElapsedTime=0;
    EMiElapsedTime=0;
    DiElapsedTime=0;
    EDiElapsedTime=0;
    CiCount=0;
    ECiCount=0;
    GiCount=0;
    EGiCount=0;
    EnCount=0;
    EEnCount=0;
    EiCount=0;
    EEiCount=0;
    MiCount=0;
    EMiCount=0;
    DiCount=0;
    EDiCount=0;
};
```

A sample provider method looks like the following:

```
CIMInstance GetInstance (…)
{
    mark startTime;
    inst = getInstance(BmDataClass.
                       CreateClassName="BmDataClass",Name="Benchmark");
    // do real work
    mark endTime;
    inst.setProperty("Property", "value");
    modifyInstance(inst);
    return;
}
```

6 Benchmark Results

6.1 HTTP Session Setup Time

From the results of all tests, the HTTP session setup time for openPegasus server with the server and client on the same machine is 2.35 msec ± 0.02msec. The HTTP session setup time on different machine is 5.5msec ± 2.0msec. This value is not accurate because the two test machines are connected to a LAN which is not dedicated to the benchmark tests.

The openWBEM server has HTTP session setup time around 3.0 msec ± 0.1msec with the server and client on the same machine, and 4.0 msec ± 0.1msec with the server and client on different machines.

The values listed in the following sections are the roundtrip time including the session setup time.

6.2 HTTPS Session Setup Time

From the results of all tests, the HTTPS session setup time is TODO. The values listed in the following sections are the roundtrip time including the session setup time.

6.3 Repository Operations

6.3.1 Class Operations

The class operations are tested with the following settings:

```
IncludeQualifiers = TRUE
IncludeClassOrigins = TRUE
LocalOnly = FALSE
DeepInheritance = FALSE
```

All roundtrip latency values use msec as unit and they are measured including the HTTP/HTTPS session setup time.

Test 1: The test class is created with only the "Description" qualifier and with no super class. The protocol is HTTP. The WBEM client and server are on the same machine.

Number of classes in the repository	123	500	1000	2000	3000	4000	5000
GetClass	12.61	13.16	14.09	15.57	16.99	18.46	19.87
EnumerateClassNames	19.05	37.71	73.57	124.84	179.94	230.96	296.48
CreateClass	13.82	13.87	14.08	14.31	14.47	14.77	15.03
DeleteClass	13.23	14.46	16.37	19.73	22.98	26.30	29.65

Test 2: The test class is created with `CIM_System` as super class and the qualifiers of the super class are included in the get operation. Including the qualifiers inherited from `CIM_System`, the test class has total 55 qualifiers. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openPegasus WBEM server.

Number of classes in the repository	123	500	1000	2000	3000	4000	5000
EnumerateClassNames	19.41	37.64	72.97	124.06	179.74	230.43	294.32
CreateClass	87.33	87.49	87.51	87.95	88.34	88.72	88.91
GetClass	49.51	49.33	50.30	51.81	53.17	54.62	57.63
DeleteClass	31.12	32.46	34.37	37.68	41.39	44.82	47.73

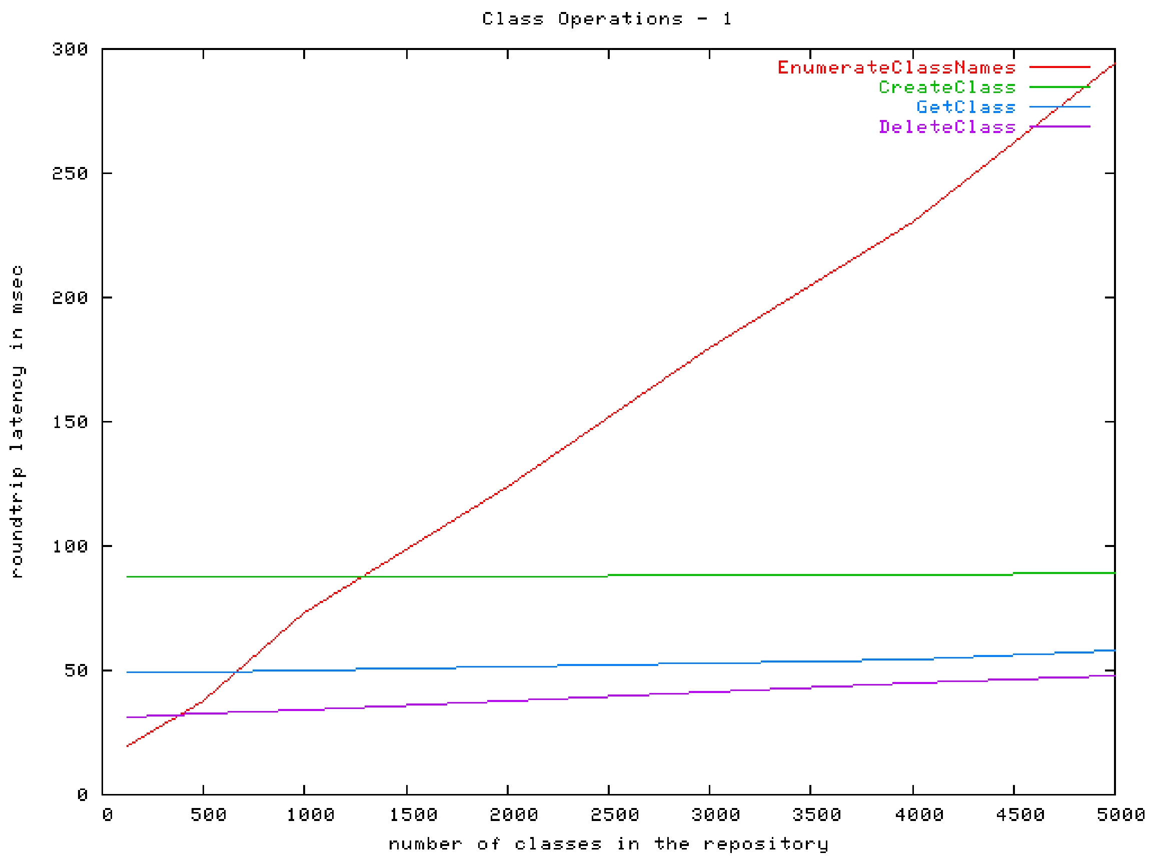


Figure 5: openPegasus WBEM Server Class Operations against Number of Classes with Server/Client on the Same Machine

Test 3: There are 1000 classes in the repository, 980 of which are from standard CIM model and 20 of which are dummy ones. The test class is created as a subclass of a certain superclass. The roundtrip latency time is measured against the total number of properties and qualifiers of the tested class. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openPegasus WBEM server.

Number of qualifiers in the class	24	54	99	158	198	267
EnumerateClassNames	72.84	73.82	73.05	73.43	73.07	73.08
CreateClass	43.84	76.79	130.87	202.19	238.83	319.60
GetClass	30.01	48.64	76.81	113.51	132.12	178.34
DeleteClass	24.02	32.69	47.64	69.77	81.74	106.61

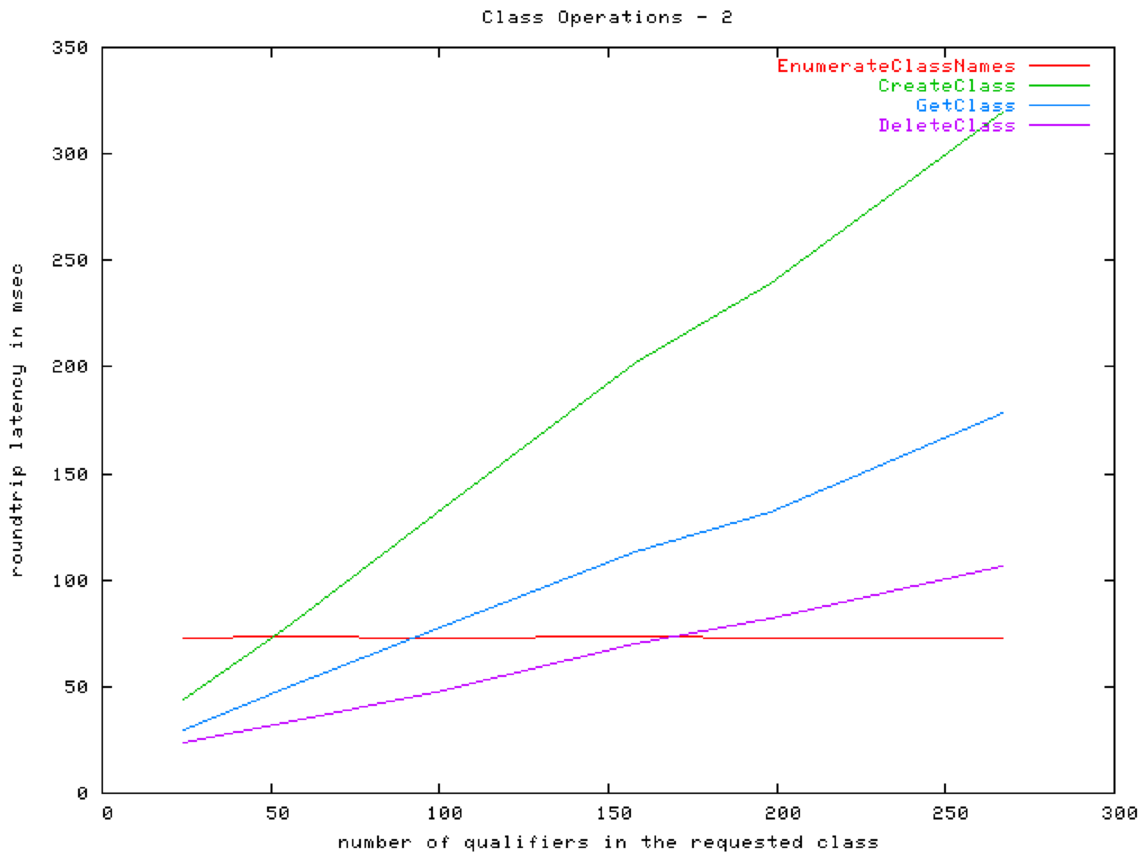


Figure 6: openPegasus WBEM Server Class Operations against Qualifiers with Server/Client on Separate Machines

Test 4: The test class is created with `CIM_System` as super class and the qualifiers of the super class are included in the get operation. Including the qualifiers inherited from `CIM_System`, the test class has total 55 qualifiers. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openPegasus WBEM server.

Number of classes in the repository	123	500	1000	2000	3000	4000	5000
EnumerateClassNames	17.38	36.70	82.93	129.92	186.32	237.98	305.34
CreateClass	77.61	75.48	76.76	76.14	78.06	76.89	76.56
GetClass	48.56	48.35	50.00	54.16	51.63	53.12	54.79
DeleteClass	29.15	29.08	35.21	35.49	39.51	43.82	45.58

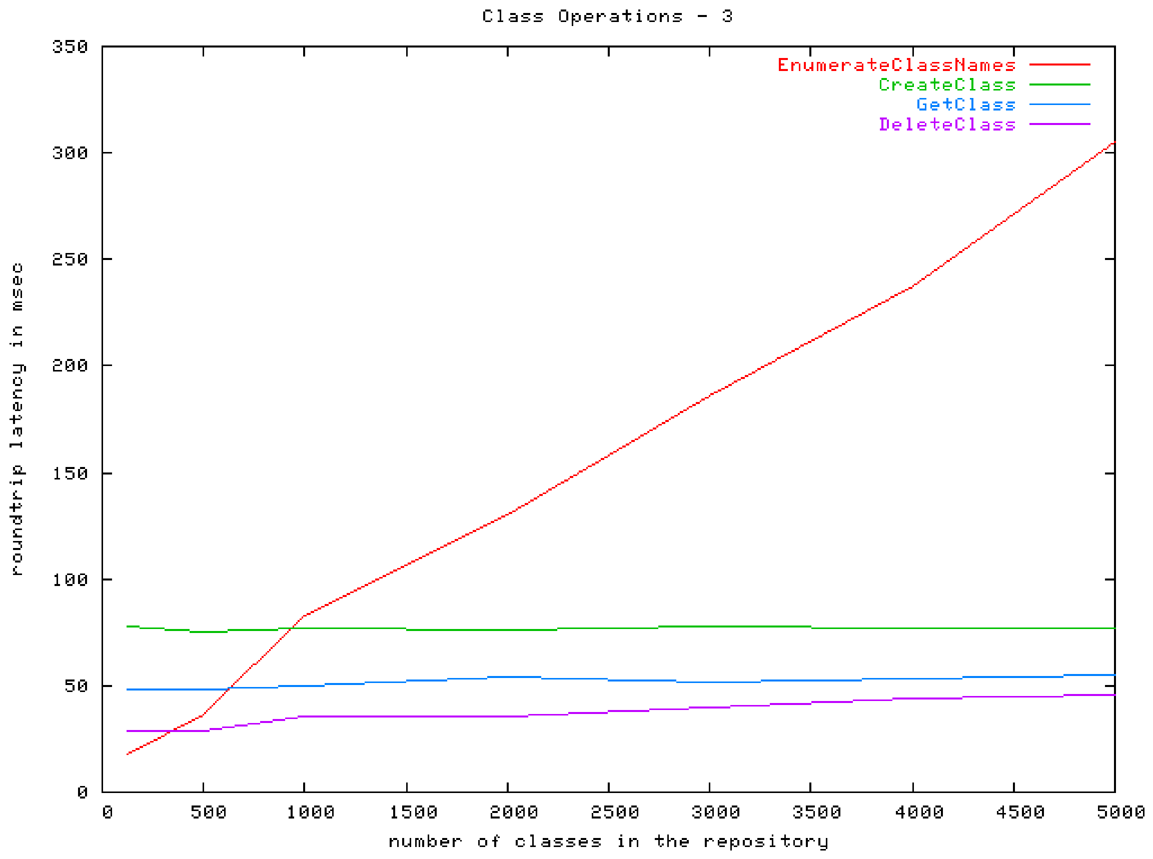


Figure 7: openPegasus WBEM Server Class Operations against Number of Classes with Client/Server on the Same Machine

Test 5: There are 1000 classes in the repository, 980 of which are from standard CIM model and 20 of which are dummy ones. The test class is created as a subclass of a certain superclass. The roundtrip latency time is measured against the total number of properties and qualifiers of the tested class. The protocol is HTTP. The WBEM client and server are on different machine. The server is openPegasus WBEM server.

Number of qualifiers in the class	24	54	99	158	198	267
EnumerateClassNames	74.37	74.13	75.14	73.79	73.62	72.84
CreateClass	43.81	75.74	129.51	200.84	239.12	332.00
GetClass	33.08	47.54	78.19	117.06	137.28	181.12
DeleteClass	26.31	32.00	49.92	71.44	85.11	110.05

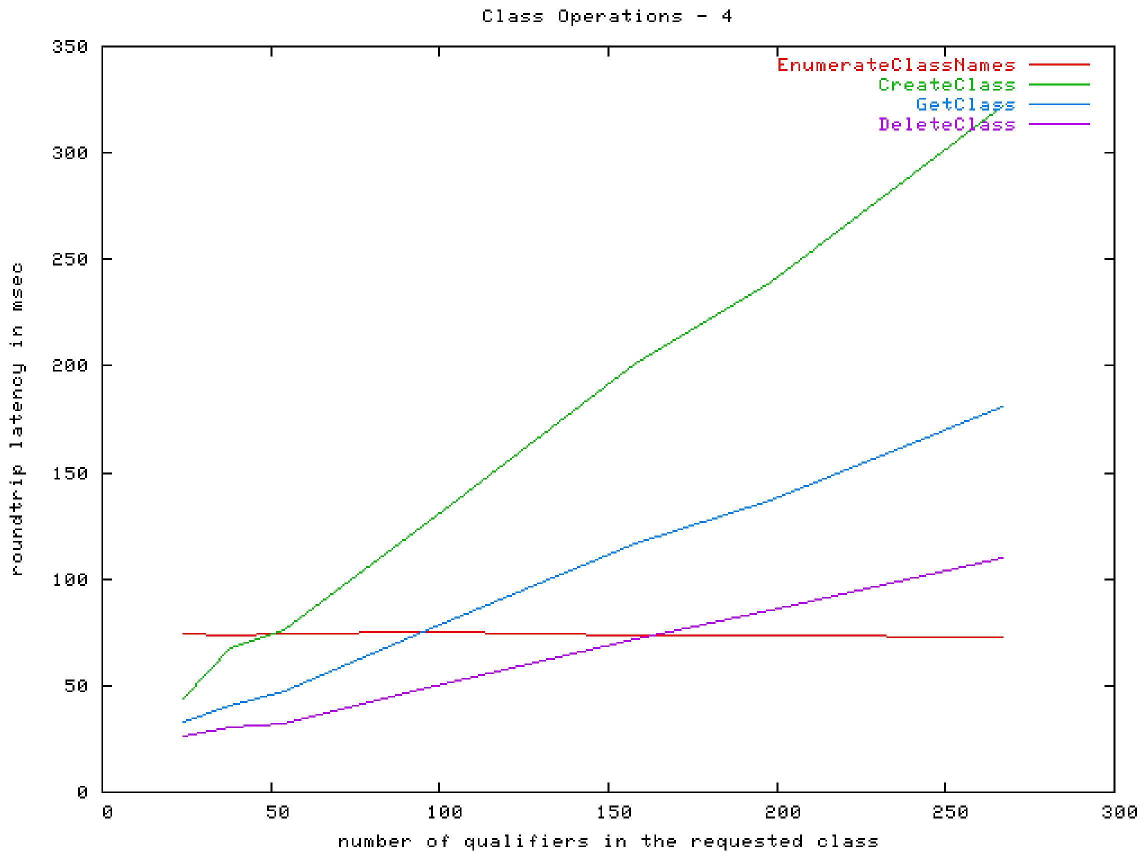
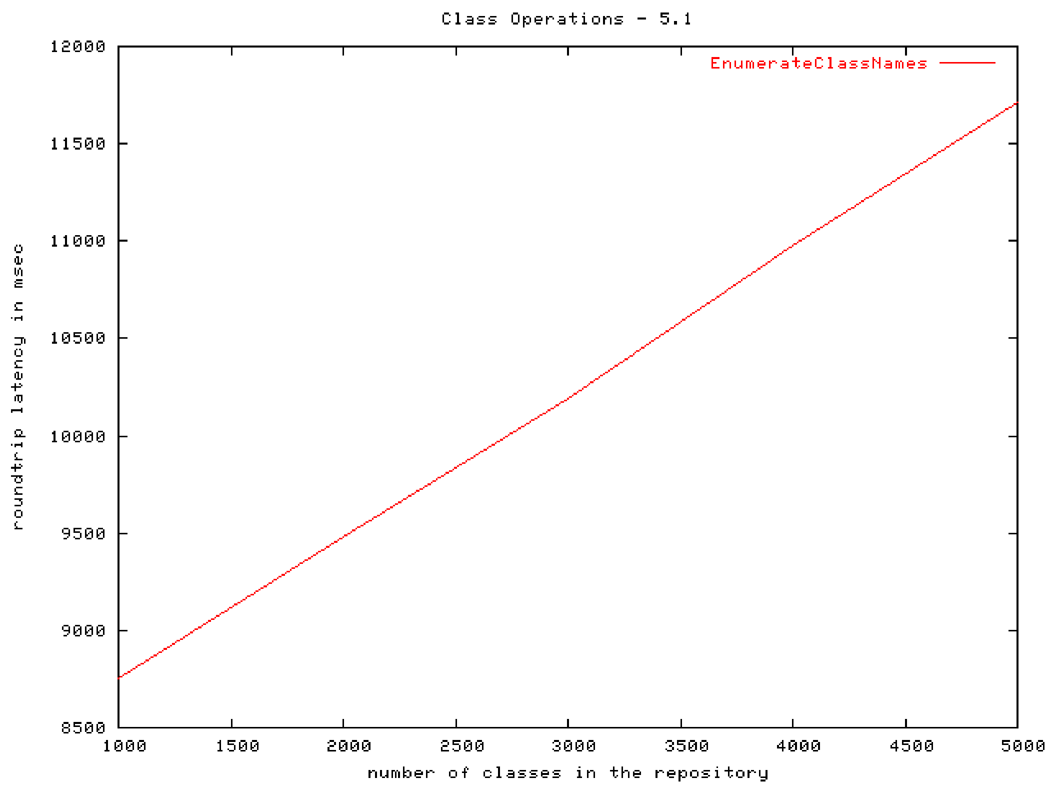


Figure 8: openPegasus WBEM Server Class Operations against Qualifiers with the Client/Server on Separate Machines

Test 6: The test class is created with `CIM_System` as super class and the qualifiers of the super class are included in the get operation. Including the qualifiers inherited from `CIM_System`, the test class has total 55 qualifiers. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openWBEM server.

Number of classes in the repository	123	500	1000	2000	3000	4000	5000
EnumerateClassNames	436.77	751.50	8751.26	9478.80	10194.40	10979.30	11715.07
CreateClass	34.64	35.05	44.30	44.33	44.19	44.30	44.40
GetClass	64.05	64.05	66.03	65.41	65.69	65.20	65.08
DeleteClass	21.33	21.35	30.09	30.29	30.27	30.25	30.51



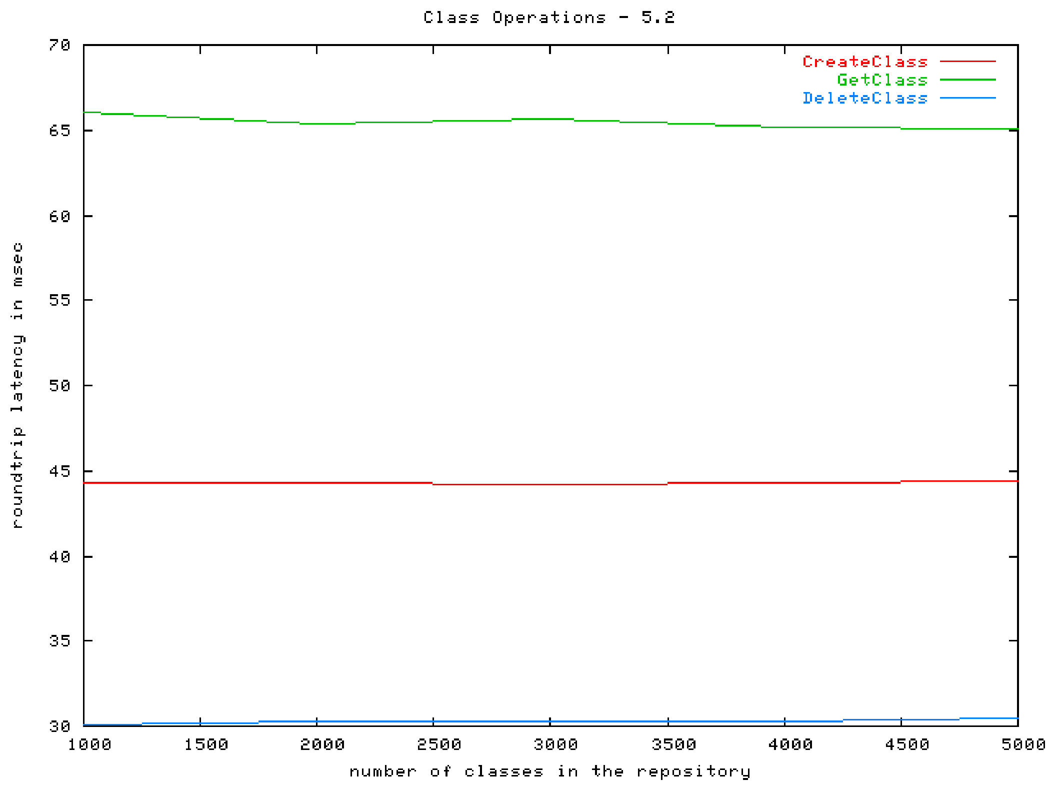


Figure 9: openWBEM Server Class Operations against Number of Classes with Client/Server on the Same Machine

Test 7: There are 1000 classes in the repository, 980 of which are from standard CIM model and 20 of which are dummy ones. The test class is created as a subclass of a certain superclass. The roundtrip latency time is measured against the total number of properties and qualifiers of the tested class. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openWBEM server.

Number of qualifiers in the class	24	37	55	105	163	203	272
EnumerateClassNames	8731.16	8732.16	8748.58	8730.75	8733.46	8726.25	8719.29
CreateClass	35.16	40.60	44.30	56.88	111.59	165.61	303.32
GetClass	22.90	30.11	64.80	51.33	81.22	102.85	127.75
DeleteClass	29.78	29.77	30.09	29.85	30.13	30.32	30.35

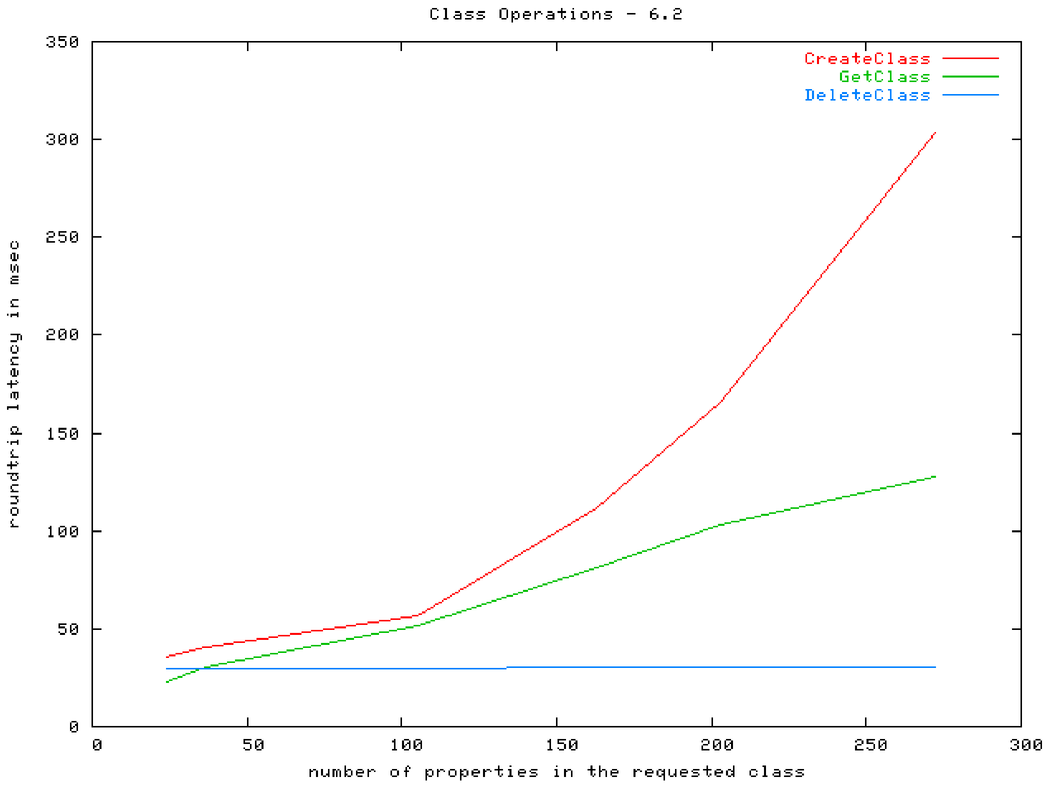
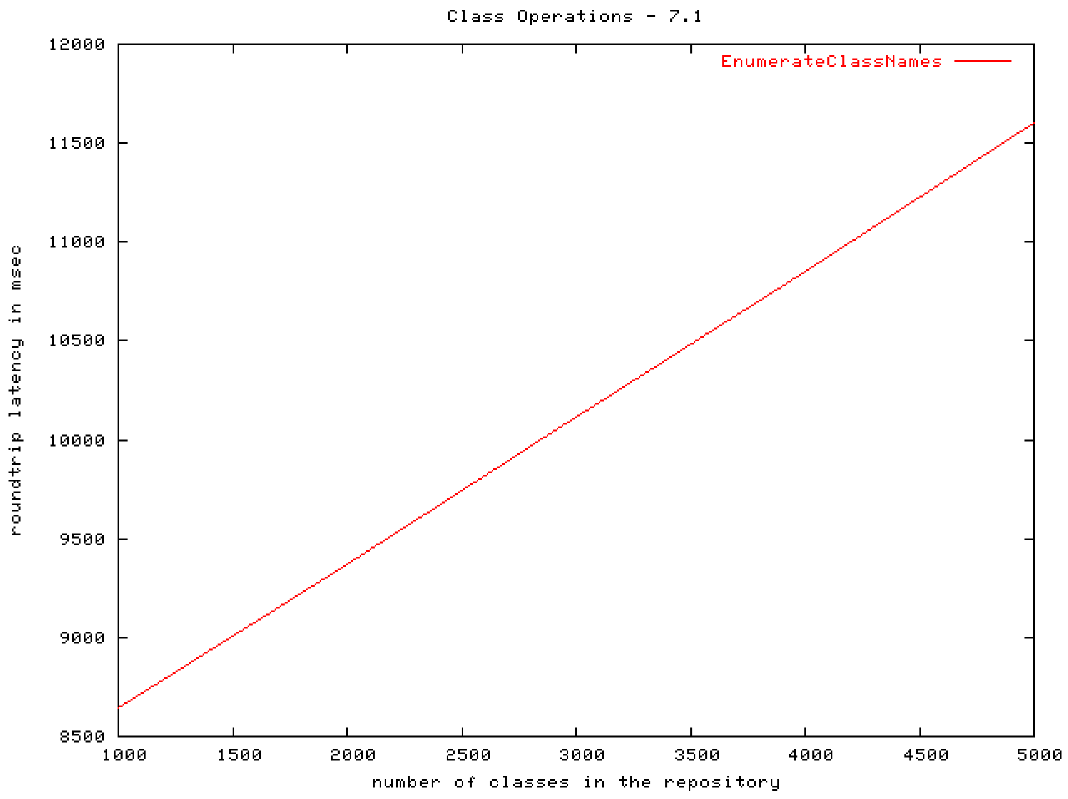


Figure 10: openWBEM Server Class Operations against Number of Qualifiers with Client/Server on the Same Machine

Test 8: The test class is created with `CIM_System` as super class and the qualifiers of the super class are included in the get operation. Including the qualifiers inherited from `CIM_System`, the test class has total 55 qualifiers. The protocol is HTTP. The WBEM client and server are on different machine. The server is openWBEM server.

Number of classes in the repository	1000	2000	3000	4000	5000
EnumerateClassNames	8642.33	9372.08	10113.39	10852.44	11600.98
CreateClass	40.02	40.42	39.96	40.20	40.35
GetClass	30.65	30.97	30.82	30.79	30.76
DeleteClass	27.71	27.75	26.34	27.30	27.53



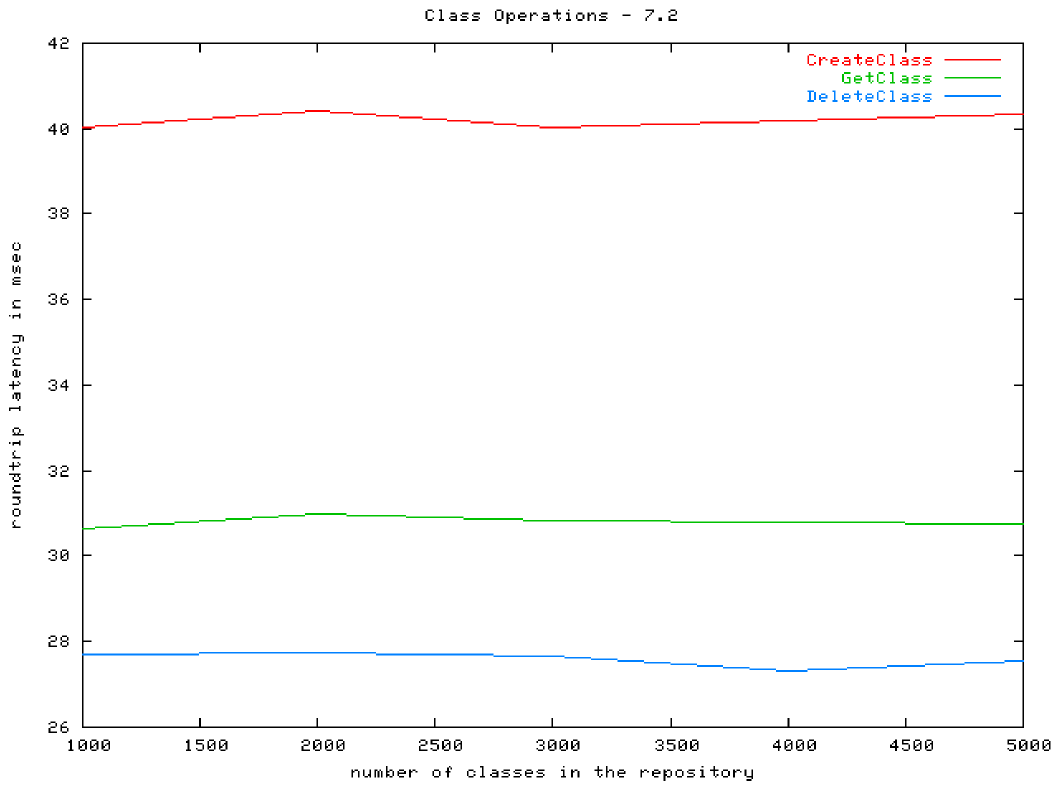


Figure 11: openWBEM Server Class Operations against Number of Classes with Client/Server on the Separate Machines

Test 9: There are 1000 classes in the repository, 980 of which are from standard CIM model and 20 of which are dummy ones. The test class is created as a subclass of a certain superclass. The roundtrip latency time is measured against the total number of properties and qualifiers of the tested class. The protocol is HTTP. The WBEM client and server are on different machines. The server is openWBEM server.

Number of qualifiers in the class	24	55	104	163	203	272
EnumerateClassNames	8644.62	8641.06	8640.81	8638.56	8634.61	8628.82
CreateClass	31.87	40.11	53.62	109.84	163.37	300.52
GetClass	21.32	31.34	48.87	65.41	75.80	97.77
DeleteClass	26.32	27.49	28.35	28.40	27.76	28.80

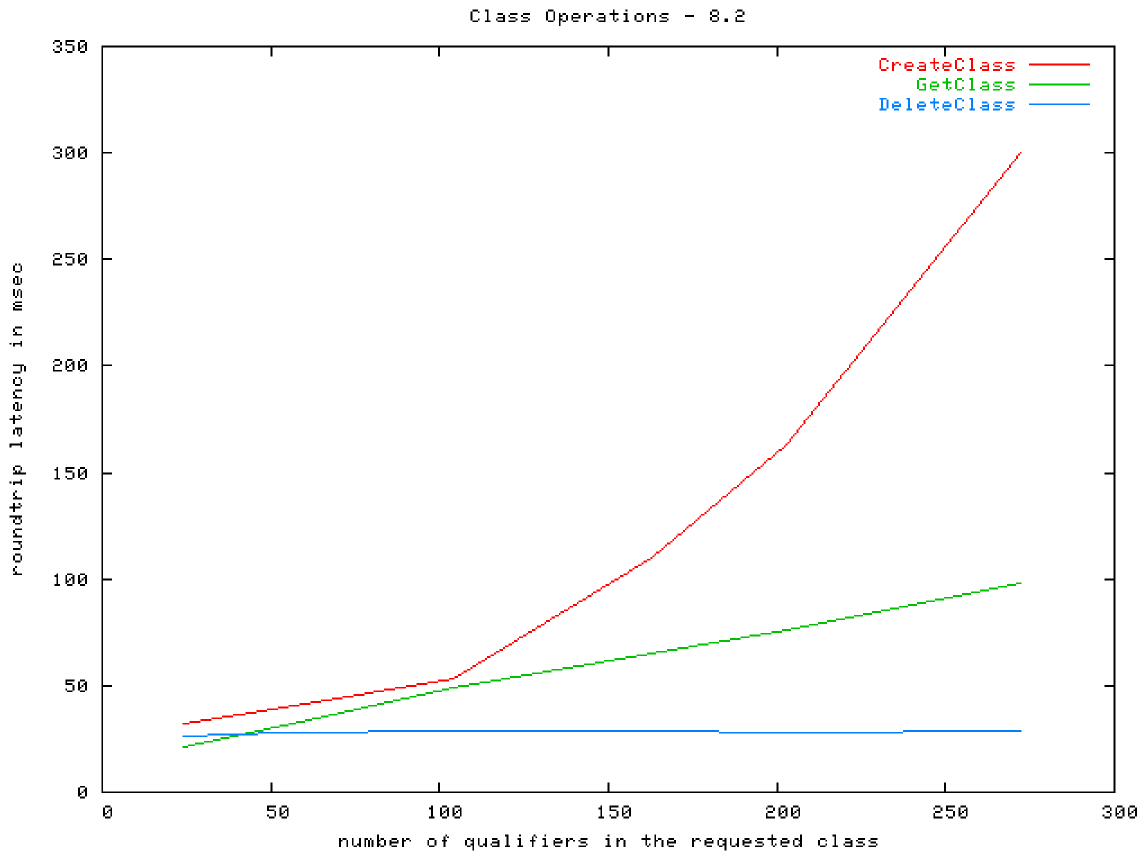


Figure 12: openWBEM Server Class Operations against Number of Qualifiers with Client/Server on the Separate Machines

Observations:

For the openPegasus WBEM server, all class operations behave linearly against some factors.

EnumerateClassNames: This operation depends solely on the number of classes in the repository. It does NOT depend on class inheritance or the number of qualifiers.

GetClass: Based on the test environment settings, the GetClass operation depends greatly on the number of qualifiers, because the GetClass operation returns the qualifiers of the class itself and its superclass. For example, to get a class which is a subclass of `CIM_System` takes nearly 37msec longer than that to get a class without any superclass or any qualifier. The number of classes in the repository has very little effect on the latency.

CreateClass: This operation depends greatly on the number of qualifiers. For example, to create a class which is a subclass of `CIM_System` takes nearly 73msec longer than that to create a class without any superclass or any qualifier. The number of classes in the repository has very little effect on the latency.

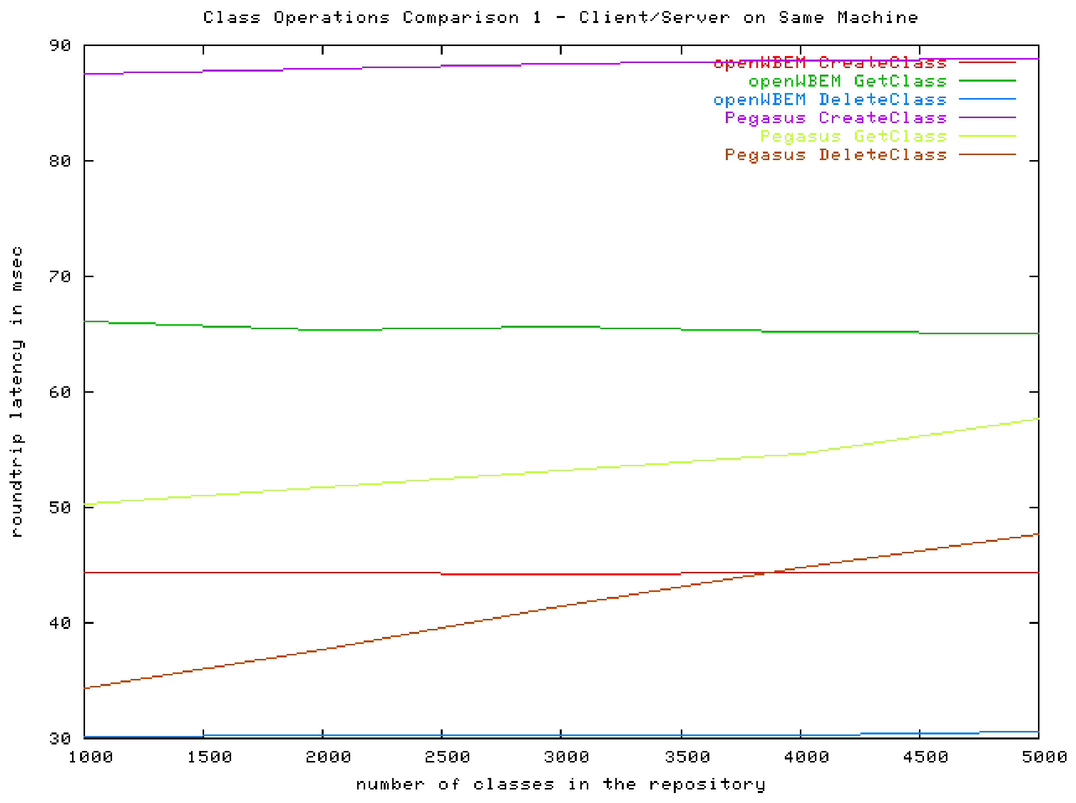
DeleteClass: This operation depends greatly on the number of qualifiers and class inheritance. For example, to delete a class which is a subclass of `CIM_System` takes nearly 28msec longer than that to delete a class without any superclass or any qualifier. The number of classes in the repository has very little effect on the latency.

For the openWBEM server, the behaviours of the class operations are similar to those of the openPegasus, but there are a few noticeable differences.

The EnumerateClassNames latency increases linearly against number of classes in the repository and does not depend on any other factors. The EnumerateClassNames operation has significant longer latency for openWBEM server. This maybe related to the repository implementation. Unlike the OpenPegasus who uses the plain XML files, the openWBEM uses the binary database and the search process in all index (*.ndx) and data (*.dat) files takes a longer period.

The number of classes in the repository does not affect the latency of CreateClass and GetClass operations. Both latencies increase against the number of qualifiers in the specified class. While the GetClass latency increases linearly, the CreateClass latency seems to increase more significantly when the number of qualifiers in the class increases.

Comparisons:



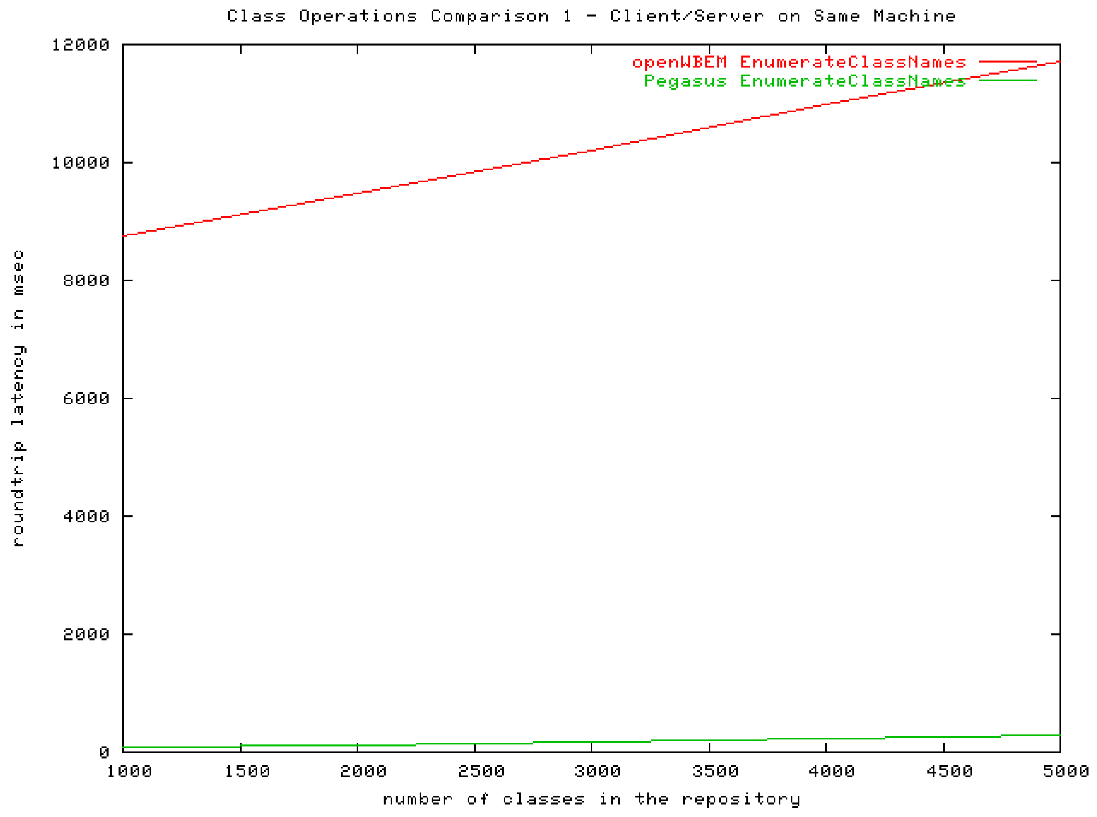
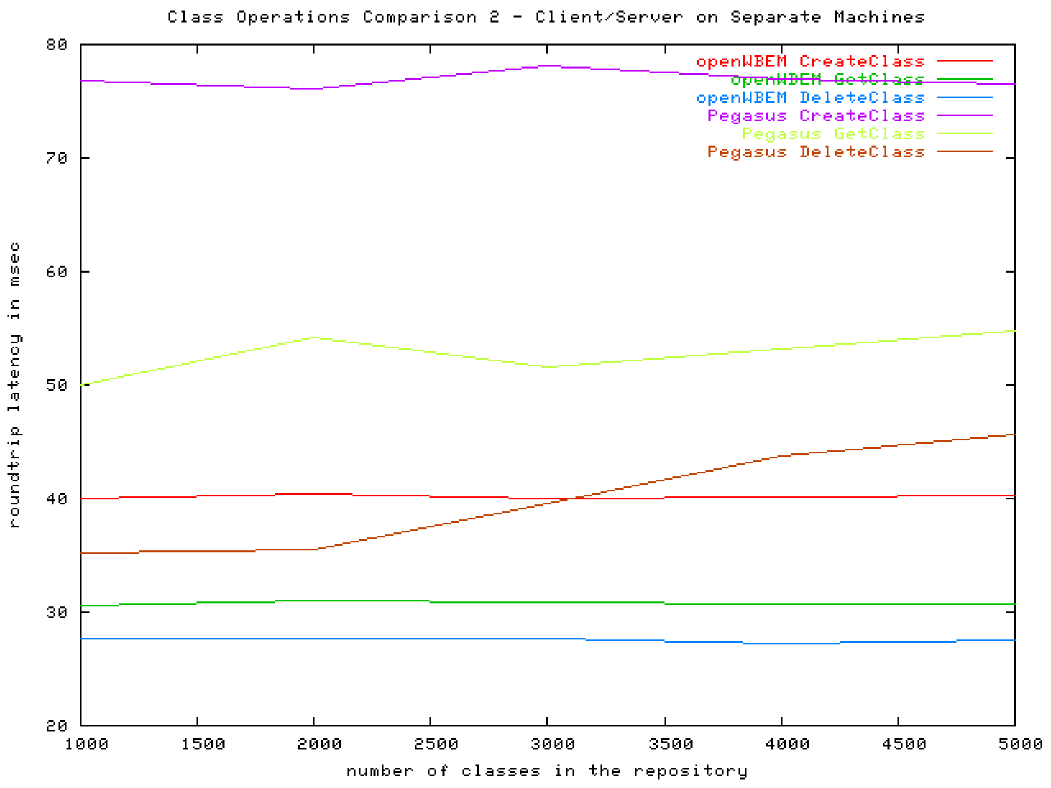


Figure 13: Class Operation Comparison 1 - Client/Server on the Same Machine



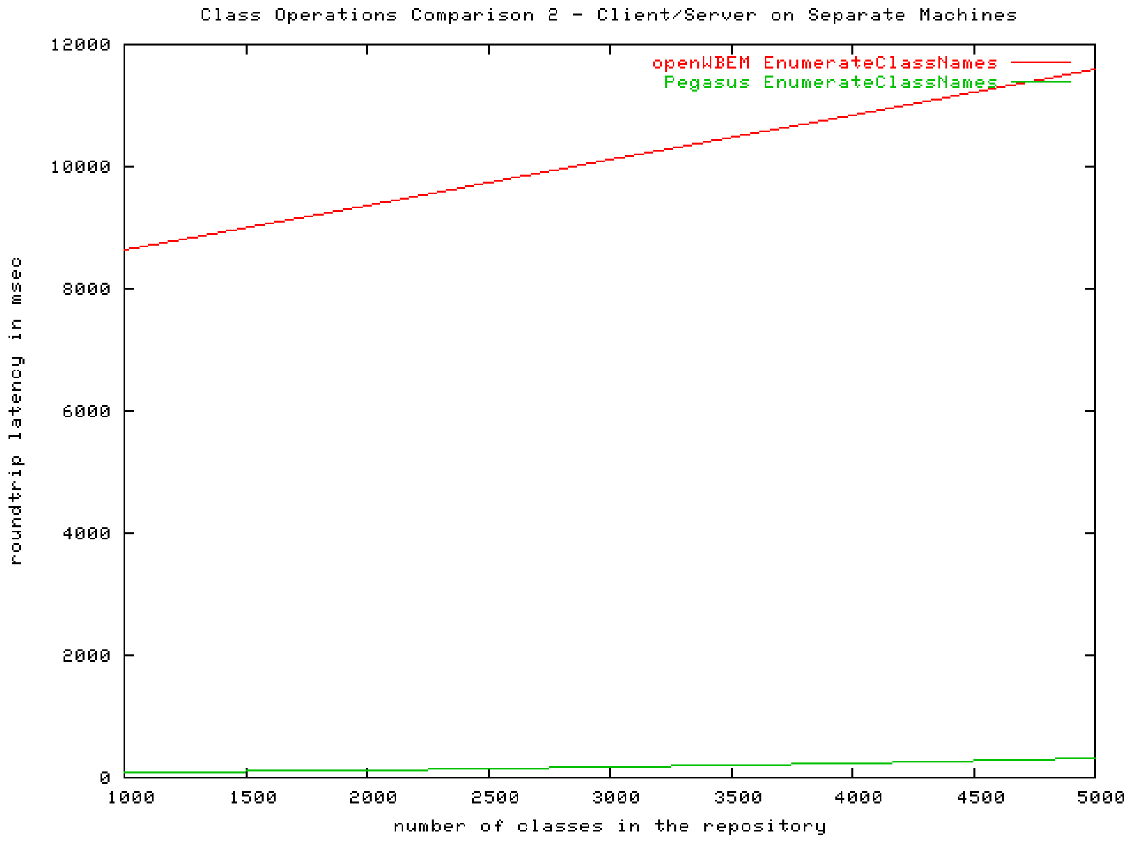
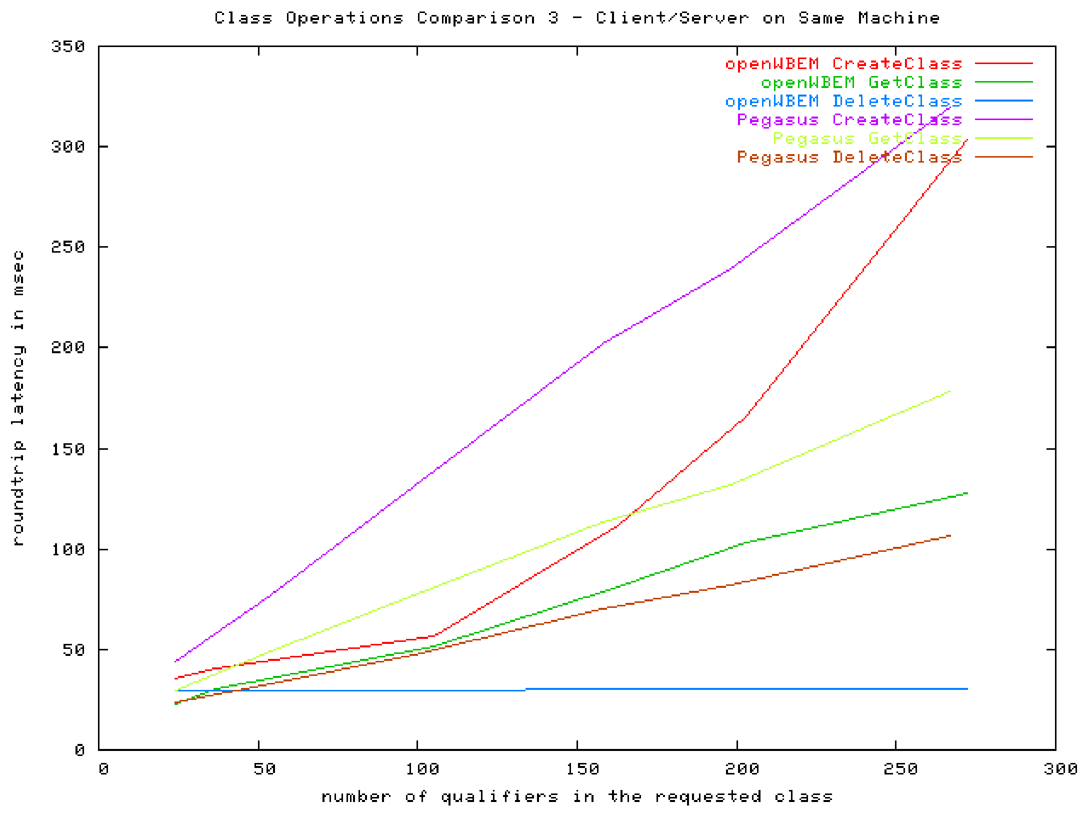


Figure 14: Class Operation Comparison 2 - Client/Server on the Separate Machines



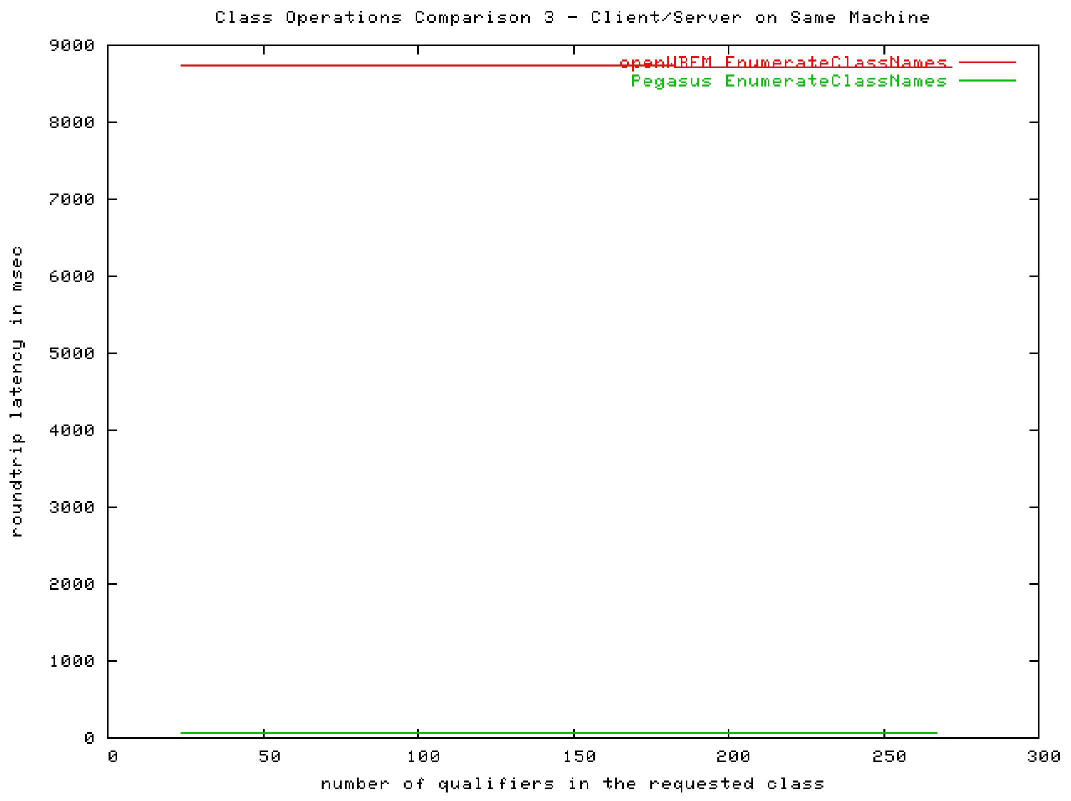
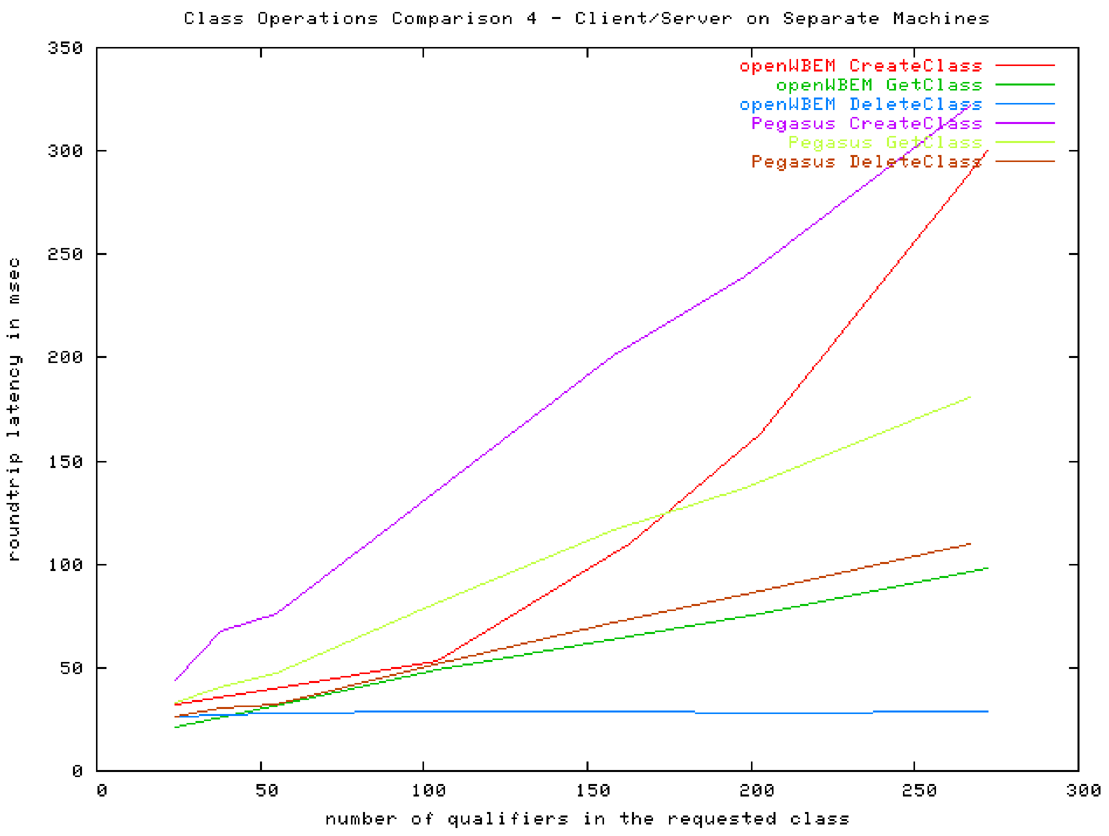


Figure 15: Class Operation Comparison 1 - Client/Server on the Same Machine



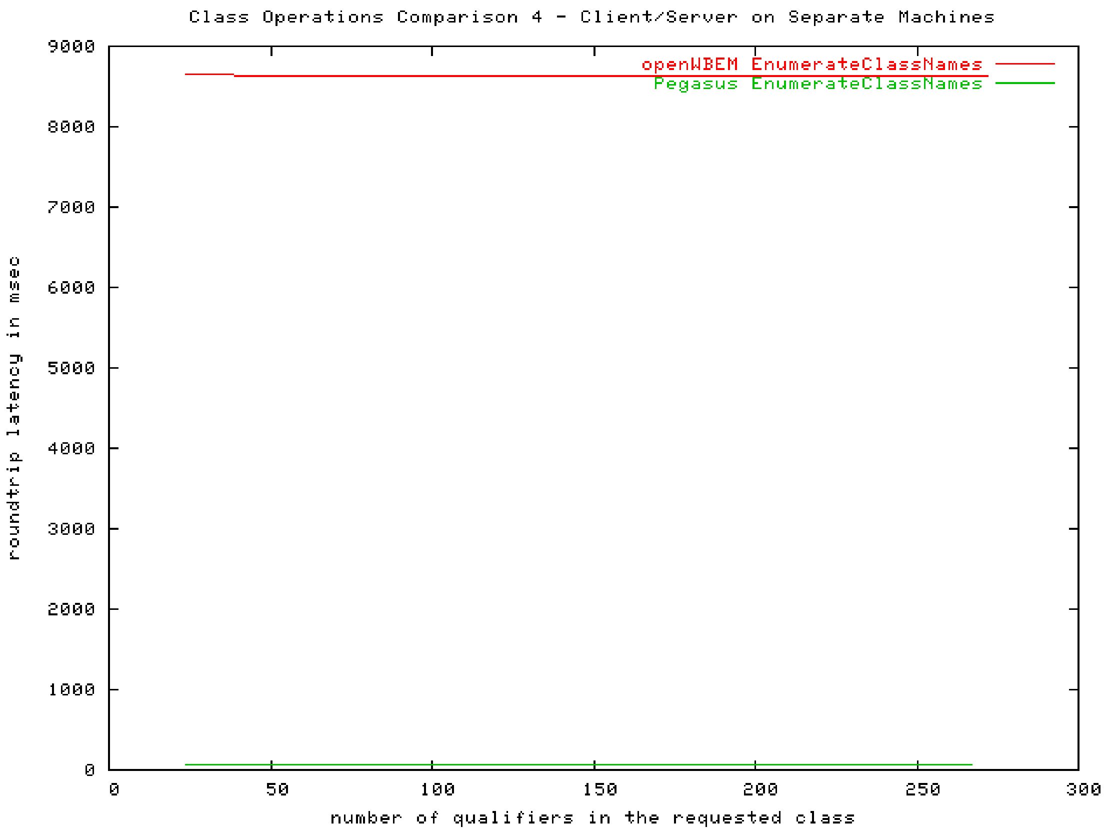


Figure 16: Class Operation Comparison 2 - Client/Server on the Separate Machines

6.3.2 *Instance Operations*

The instance operations are tested with the following settings:

```
IncludeQualifiers = FALSE
IncludeClassOrigins = FALSE
LocalOnly = FALSE
DeepInheritance = FALSE
```

All roundtrip latency values use msec as unit and they are measured including the HTTP/HTTPS session setup time.

Test 1: Instance operations against number of keys of the instance, and the key type is string. The test class is a subclass of CIM_EnabledLogicalElement. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openWBEM server.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	38.30	40.68	44.76	48.28	53.01	58.36
EnumerateInstanceNames	170.13	247.56	346.66	397.36	523.16	617.81
EnumerateInstances	815.79	942.16	1205.31	1457.01	1786.34	2081.13
GetInstance	36.66	36.61	38.95	41.62	44.97	47.63
ModifyInstance	48.75	47.74	53.93	59.02	65.39	72.10
DeleteInstance	42.43	42.56	46.58	50.88	55.22	60.40

Test 2: Instance operations against number of keys of the instance, and the key type is uint32. The test class is a subclass of CIM_EnabledLogicalElement. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openWBEM server.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	38.45	39.85	42.99	45.86	49.78	54.09
EnumerateInstanceNames	169.52	250.88	306.81	404.19	523.91	620.24
EnumerateInstances	809.04	935.14	1199.18	1476.47	1749.59	2093.73
GetInstance	35.52	37.61	41.83	45.14	49.79	54.40
ModifyInstance	46.34	50.74	53.63	64.61	73.52	81.57
DeleteInstance	31.89	44.64	50.63	56.35	63.60	69.91

Test 3: Instance operations against number of keys of the instance, and the key type is string. The test class is a subclass of CIM_EnabledLogicalElement. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openPegasus server.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	97.36	104.54	135.83	151.29	183.37	197.73
EnumerateInstanceNames	76.05	90.07	147.78	176.76	237.86	270.46
EnumerateInstances	3929.09	3987.60	4148.43	4287.81	4464.81	4598.71
GetInstance	64.20	61.37	72.34	67.71	78.68	71.44
ModifyInstance	100.23	101.79	123.05	128.71	149.99	154.41
DeleteInstance	50.55	61.08	82.19	103.09	124.99	147.79

Test 4: Instance operations against number of keys of the instance, and the key type is uint32. The test class is a subclass of CIM_EnabledLogicalElement. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openPegasus server.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	89.67	109.27	116.36	123.19	146.94	153.33
EnumerateInstanceNames	67.17	110.02	133.12	155.83	210.28	237.92
EnumerateInstances	3915.41	4010.27	4105.21	4251.51	4378.14	4523.48
GetInstance	60.84	74.18	69.22	64.37	74.90	67.95
ModifyInstance	94.45	111.17	112.55	113.71	131.29	131.80
DeleteInstance	46.27	52.92	65.63	78.38	91.86	106.52

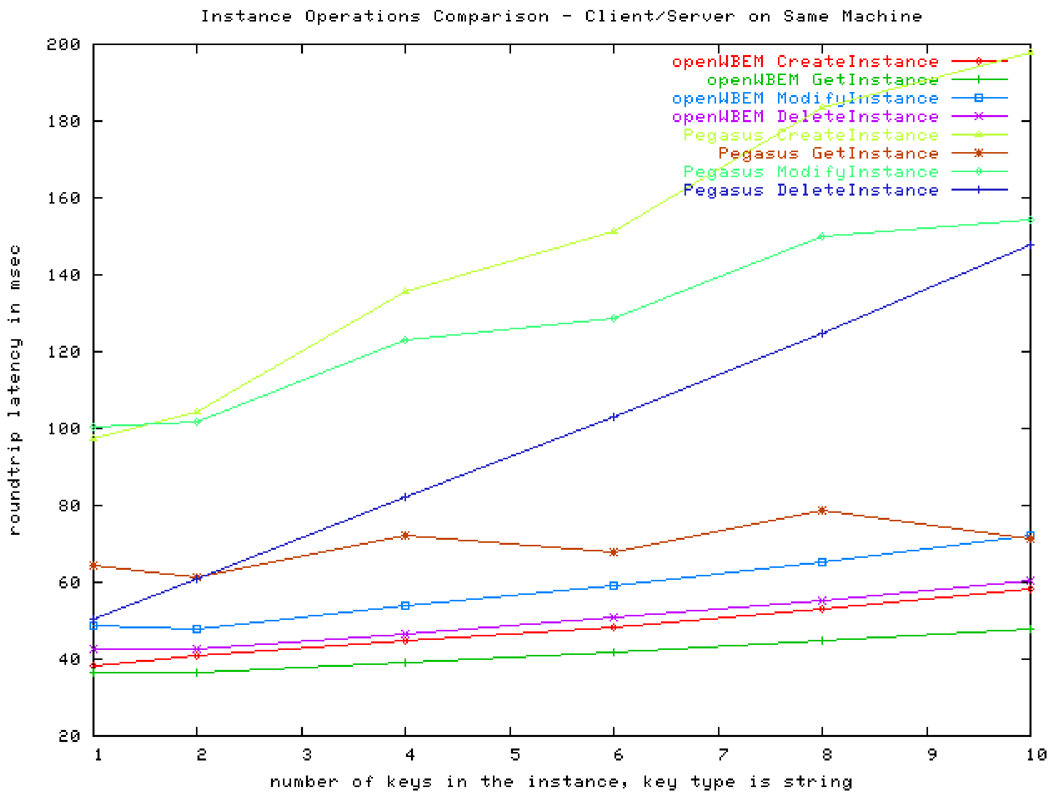


Figure 17: Instance Operation Comparison 1 - Client/Server on the Same Machines

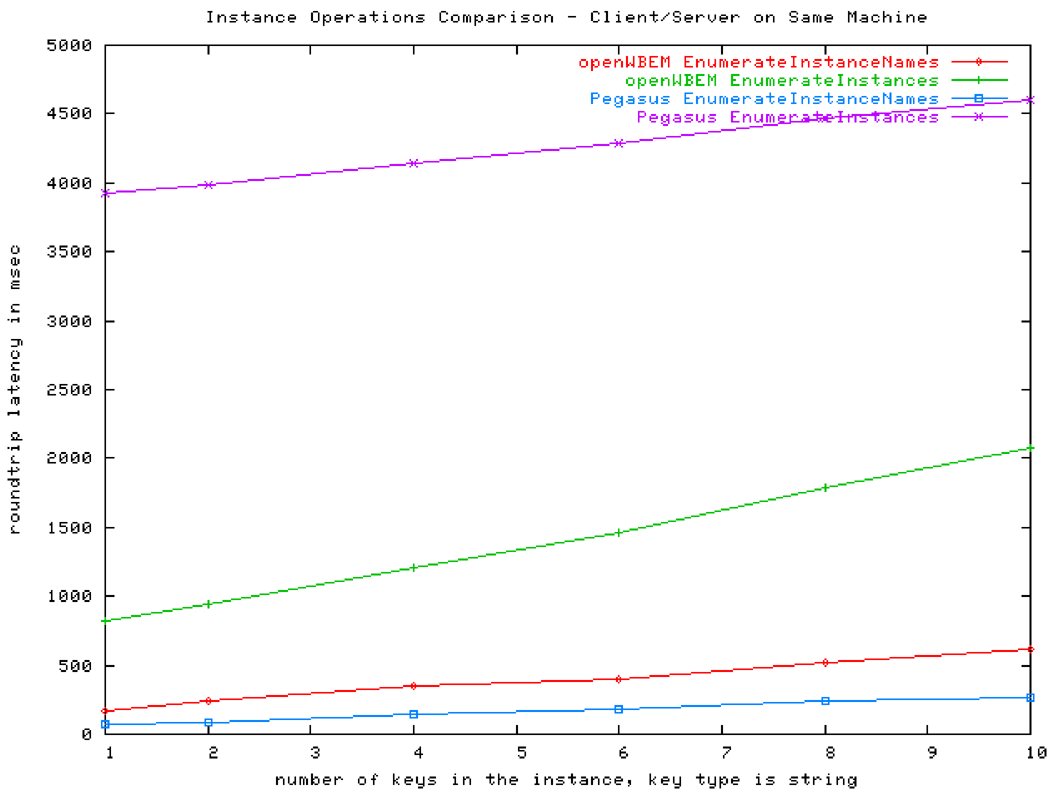


Figure 18: Instance Operation Comparison 2 - Client/Server on the Same Machines

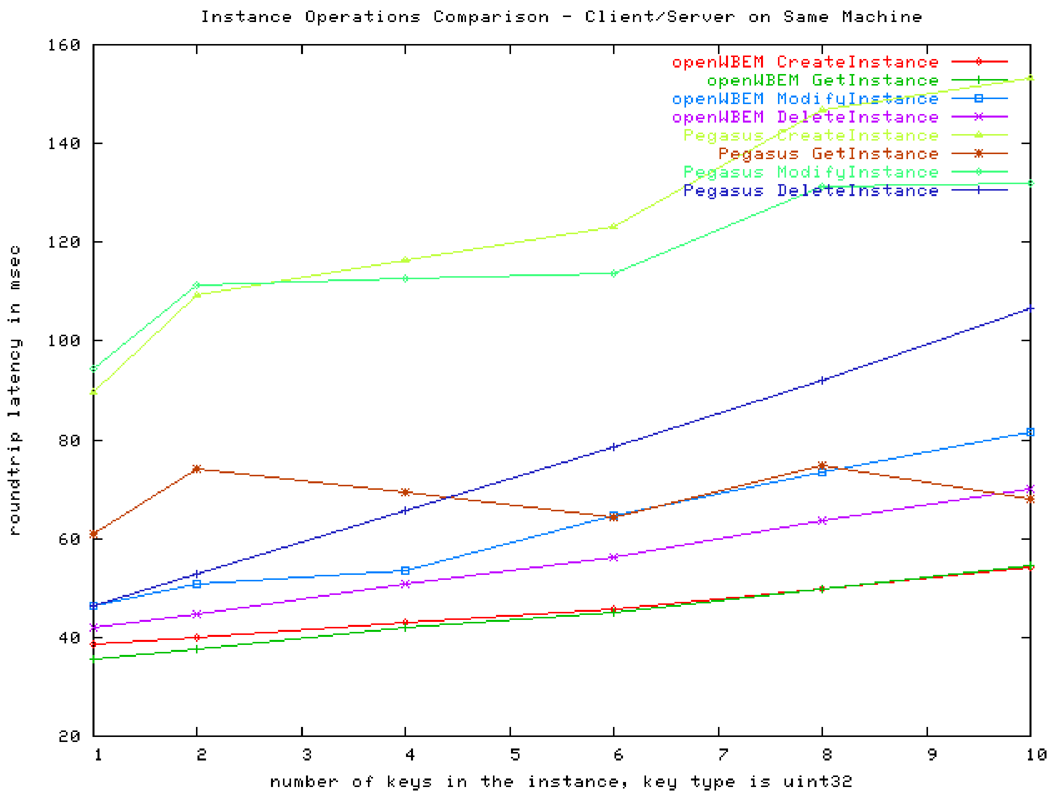


Figure 19: Instance Operation Comparison 3 - Client/Server on the Same Machine

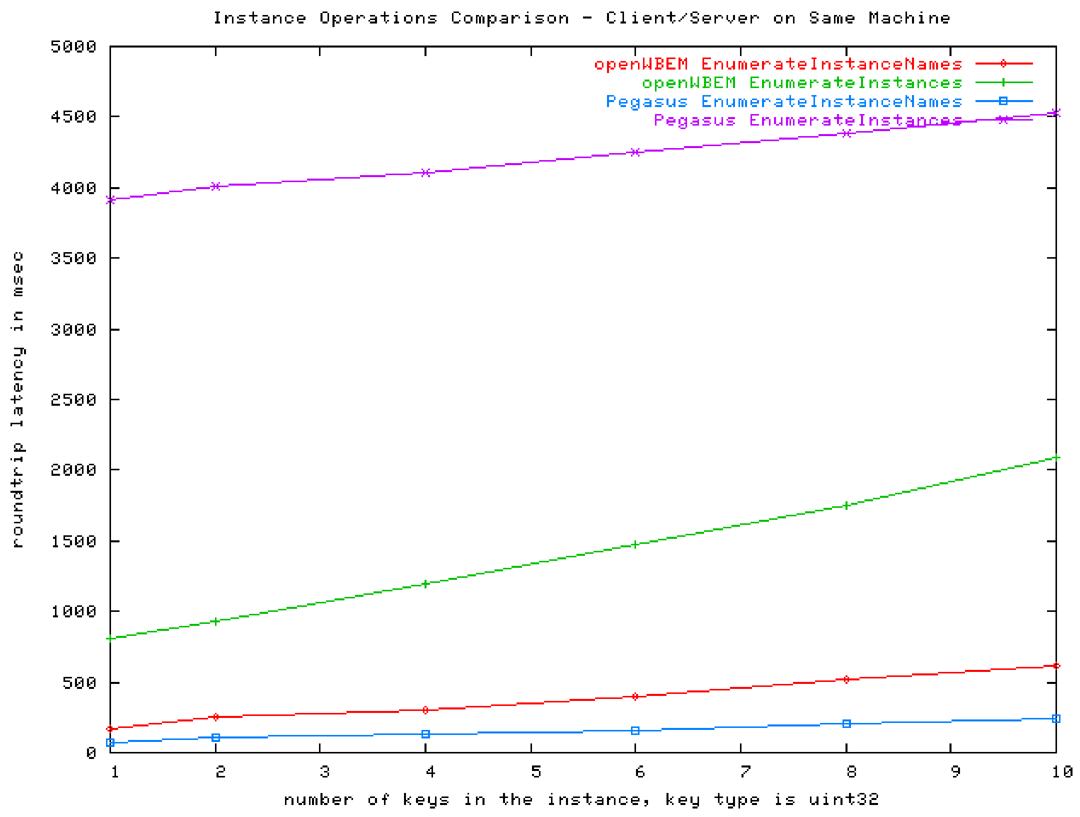


Figure 20: Instance Operation Comparison 4 - Client/Server on the Same Machine

Test 5: Instance operations against number of instances of the requested class, and the key type is string. The test class is a subclass of CIM_EnabledLogicalElement and the instance has four keys. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openWBEM server.

Number of keys in the instance	0	50	100	200	500	1000
CreateInstance	35.21	35.16	35.62	35.54	36.25	37.14
EnumerateInstanceNames	13.37	189.07	336.84	653.33	1460.91	2935.62
EnumerateInstances	23.56	628.07	1191.72	2315.10	5790.18	11537.80
GetInstance	30.78	31.47	31.45	31.56	31.50	31.55
ModifyInstance	45.00	45.14	45.04	45.52	46.09	47.02
DeleteInstance	38.34	38.74	38.18	38.22	38.69	39.57

Test 6: Instance operations against number of instances of the requested class, and the key type is uint32. The test class is a subclass of CIM_EnabledLogicalElement and the instance has four keys. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openWBEM server.

Number of keys in the instance	0	50	100	200	500	1000
CreateInstance	33.07	33.15	33.10	33.15	33.30	33.18
EnumerateInstanceNames	13.41	194.91	302.20	614.66	1485.66	1931.26

EnumerateInstances	23.45	620.18	1197.85	2338.36	5770.42	11479.70
GetInstance	34.04	33.80	34.06	34.07	34.04	34.05
ModifyInstance	48.38	48.21	48.14	48.22	48.54	48.59
DeleteInstance	42.24	41.97	42.04	41.98	42.20	42.00

Test 7: Instance operations against number of instances of the requested class, and the key type is string. The test class is a subclass of CIM_EnabledLogicalElement and the instance has four keys. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openPegasus server.

Number of keys in the instance	0	50	100	200	500	1000
CreateInstance	81.28	107.56	135.76	197.70	380.05	678.05
EnumerateInstanceNames	46.71	96.27	151.94	255.84	584.72	1137.98
EnumerateInstances	83.75	2119.74	4161.97	8296.74	20672.30	41263.00
GetInstance	72.60	72.97	72.30	72.43	72.60	116.22
ModifyInstance	99.19	111.03	123.16	148.83	226.07	354.00
DeleteInstance	24.71	53.86	82.02	143.99	325.58	625.30

Test 8: Instance operations against number of instances of the requested class, and the key type is uint32. The test class is a subclass of CIM_EnabledLogicalElement and the instance has four keys. The protocol is HTTP. The WBEM client and server are on the same machine. The server is openPegasus server.

Number of keys in the instance	0	50	100	200	500	1000
CreateInstance	73.64	96.05	116.48	162.25	295.74	515.34
EnumerateInstanceNames	37.59	85.75	133.21	228.94	526.02	1024.74
EnumerateInstances	77.18	2098.55	4125.32	8172.65	20510.60	40889.00
GetInstance	69.15	70.04	69.25	69.28	69.36	69.35
ModifyInstance	94.65	104.54	112.69	131.26	186.75	279.53
DeleteInstance	24.04	45.39	65.76	111.21	244.79	464.47

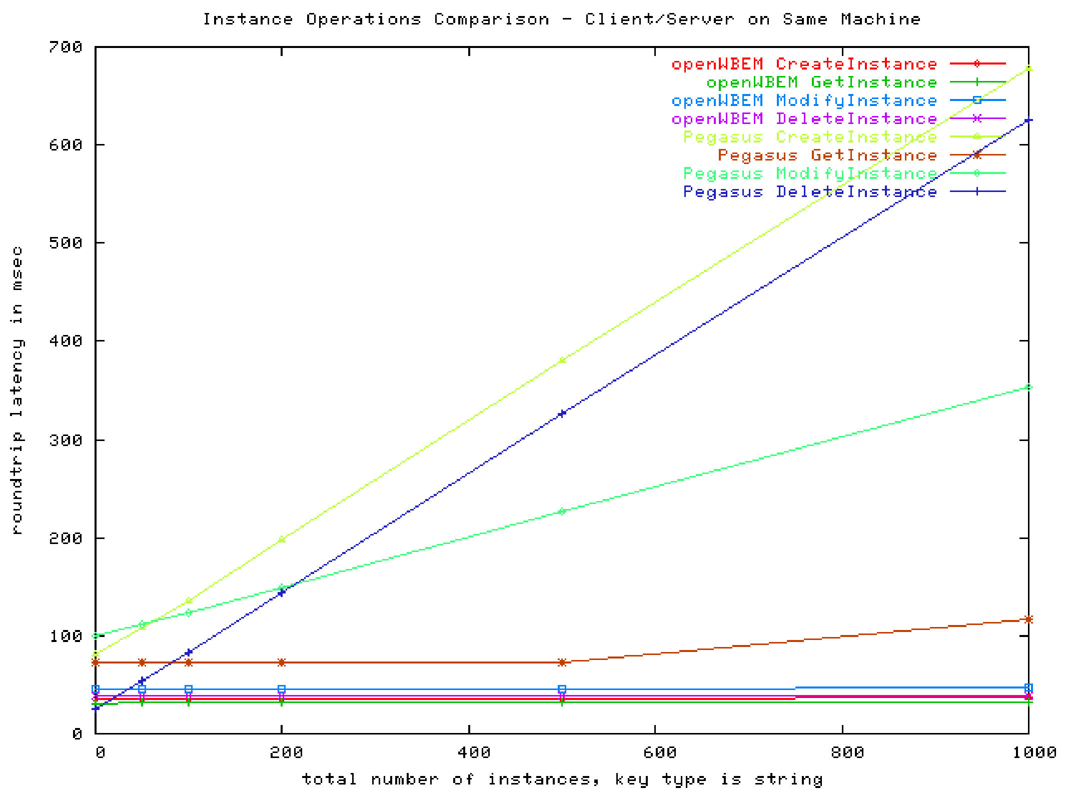


Figure 21: Instance Operation Comparison 5 - Client/Server on the Same Machine

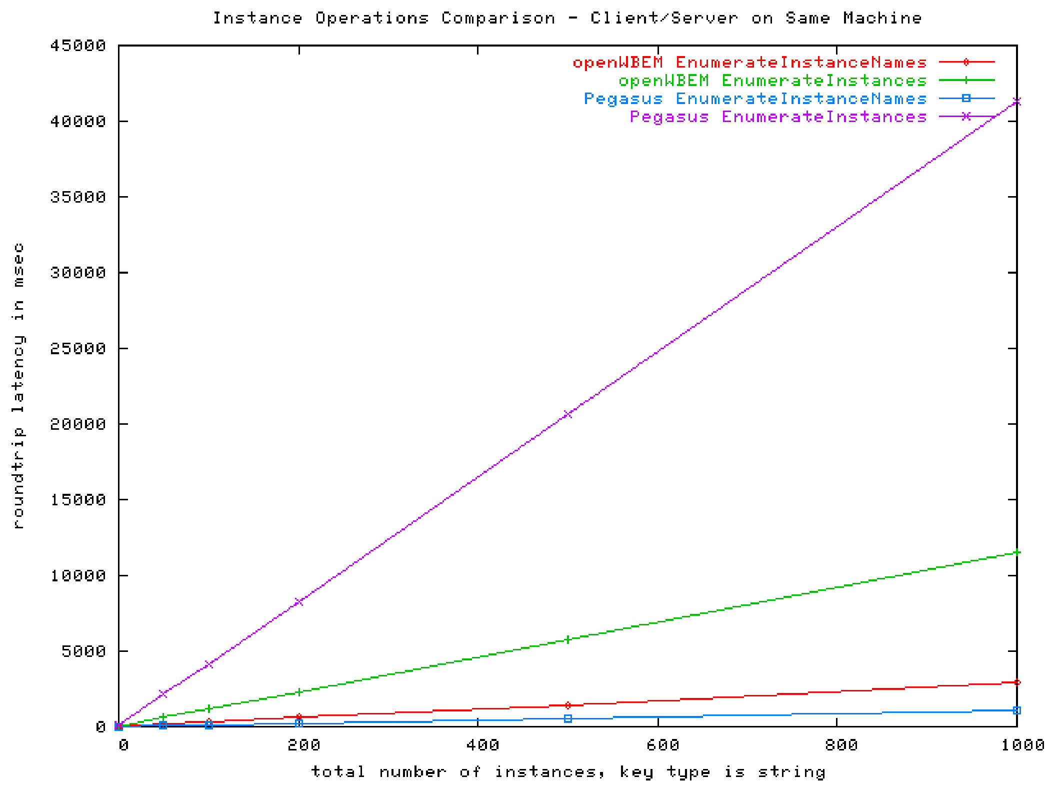


Figure 22: Instance Operation Comparison 6 - Client/Server on the Same Machine

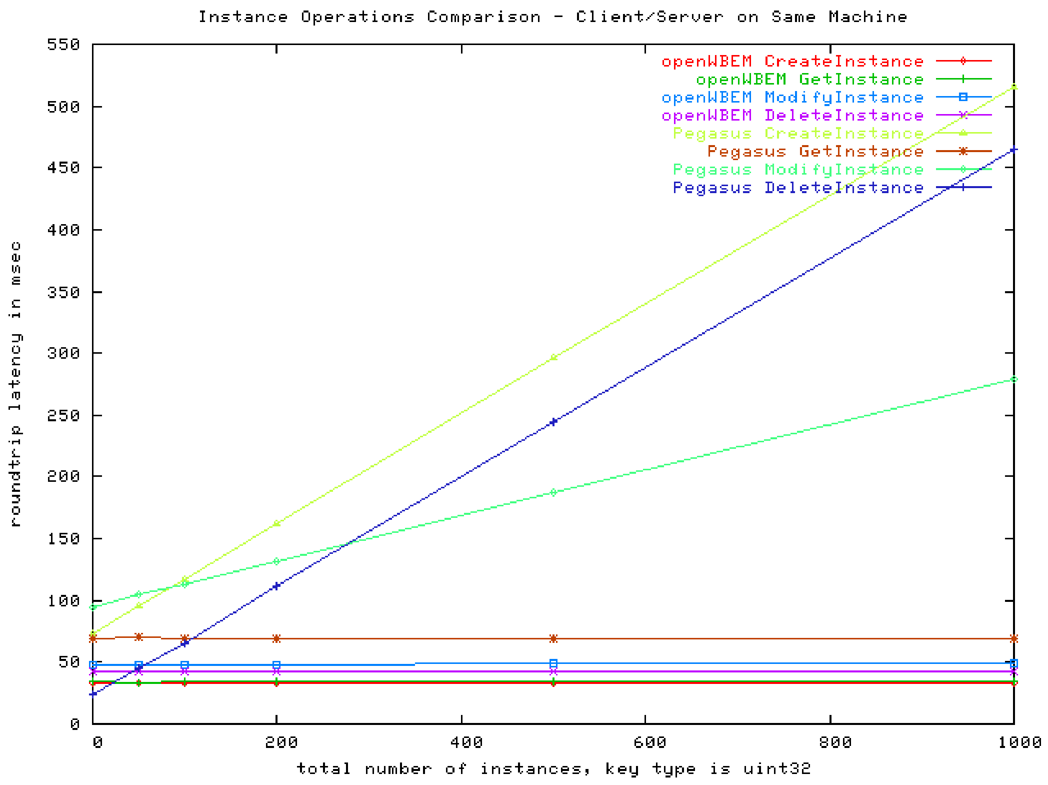


Figure 23: Instance Operation Comparison 7 - Client/Server on the Same Machine

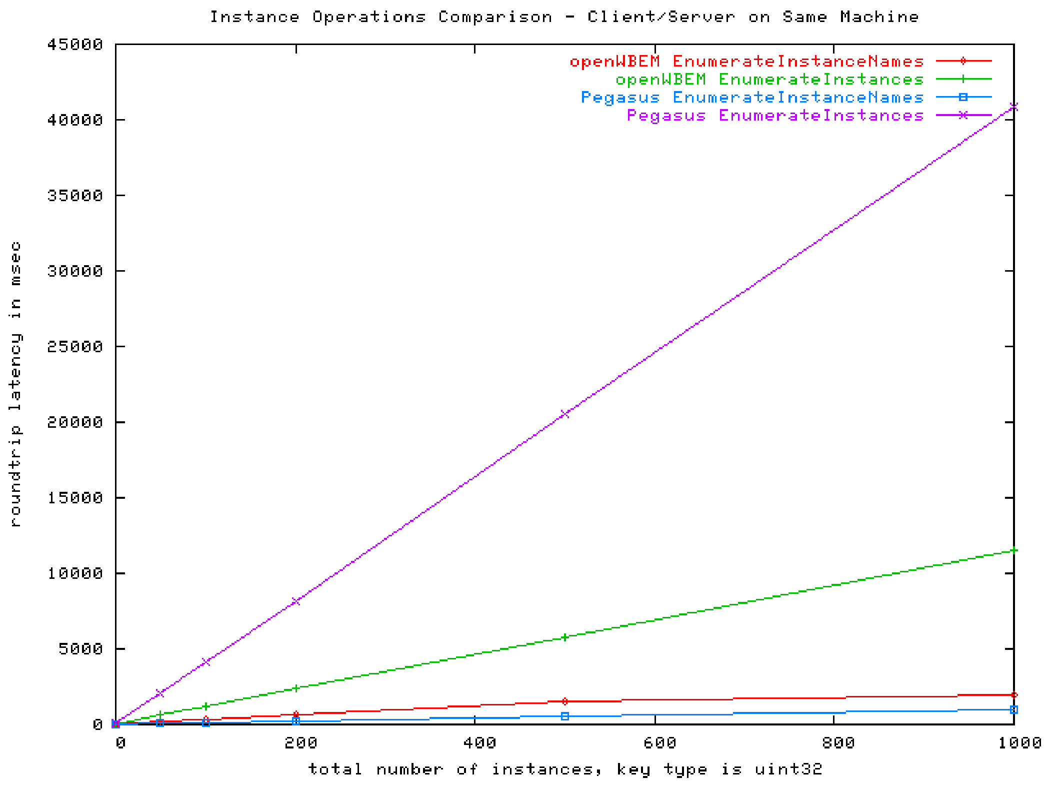


Figure 24: Instance Operation Comparison 8 - Client/Server on the Same Machine

6.4 Provider Operations

The test results listed in this section are obtained under test **condition A** described in section 5.2.3. All latency units are in msec.

6.4.1 Instance Operations

6.4.1.1 Tests against States in openWBEM

Test 1: The provider returns a success response for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is string. The WBEM server is openWBEM server.

The minimum actions required in the methods: EnumerateInstanceNames and EnumerateInstances return empty list; the CreateInstance returns an CIMObjectPath from the incoming instance; the GetInstance returns an instance constructed by the given className; ModifyInstance and DeleteInstance immediately return with no action taken.

Test state	The initial state of the WBEM server	The state of the first-time provider invocation after the WBEM server has been already started for a while	The state after a long provider idle period if a provider timeout is defined	The normal operation state
Library conditions	The provider library and the request handler library are NOT loaded yet	The provider library is NOT yet loaded, but the request handler library is already loaded	The provider library is previously loaded and unloaded. It is to be RE-loaded, while the request handler library is already loaded	Both provider library and the request library are already loaded
CreateInstance	103.85	43.55	35.23	29.34
EnumerateInstanceNames	53.06	24.86	18.50	12.64
EnumerateInstances	54.74	26.53	19.97	13.85
GetInstance	102.22	40.90	34.53	28.04
ModifyInstance	105.89	45.05	36.56	30.80
DeleteInstane	101.01	40.89	32.50	26.73

Test 2: The provider returns a success response for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is uint32. The WBEM server is openWBEM server.

The minimum actions required in the methods: EnumerateInstanceNames and EnumerateInstances return empty list; the CreateInstance returns an CIMObjectPath from the incoming instance; the GetInstance returns an instance constructed by the given className; ModifyInstance and DeleteInstance immediately return with no action taken.

Test state	The initial state of the WBEM server	The state of the first-time provider invocation after the WBEM server has been already started for a while	The state after a long provider idle period if a provider timeout is defined	The normal operation state
Library conditions	The provider library and the request handler library are NOT loaded yet	The provider library is NOT yet loaded, but the request handler library is already loaded	The provider library is previously loaded and unloaded. It is to be RE-loaded, while the request handler library is already loaded	Both provider library and the request library are already loaded
CreateInstance	101.75	41.21	33.51	28.12
EnumerateInstanceNames	52.73	24.82	18.48	12.64
EnumerateInstances	54.46	26.63	19.43	13.82
GetInstance	103.81	42.53	35.72	29.63
ModifyInstance	107.67	47.21	37.13	32.04
DeleteInstane	103.40	42.30	34.39	28.39

Test 3: The provider returns an exception for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is string. The WBEM server is openWBEM server.

The minimum actions required in the methods: all operations performs no action except to throw an FAILED exception with a message.

Test state	The initial state of the WBEM server	The state of the first-time provider invocation after the WBEM server has been already started for a while	The state after a long provider idle period if a provider timeout is defined	The normal operation state
Library conditions	The provider library and the request handler library are NOT loaded yet	The provider library is NOT yet loaded, but the request handler library is already loaded	The provider library is previously loaded and unloaded. It is to be RE-loaded, while the request handler library is already loaded	Both provider library and the request library are already loaded
CreateInstance	61.34	33.39	26.91	20.42
EnumerateInstanceNames	51.69	23.77	17.33	11.20

EnumerateInstances	53.58	25.75	18.57	13.23
GetInstance	55.80	27.48	21.05	14.57
ModifyInstance	56.42	28.05	20.95	14.86
DeleteInstane	54.49	26.12	19.56	13.63

Test 4: The provider returns an exception for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is uint32. The WBEM server is openWBEM server.

The minimum actions required in the methods: all operations performs no action except to throw an FAILED exception with a message.

Test state	The initial state of the WBEM server	The state of the first-time provider invocation after the WBEM server has been already started for a while	The state after a long provider idle period if a provider timeout is defined	The normal operation state
Library conditions	The provider library and the request handler library are NOT loaded yet	The provider library is NOT yet loaded, but the request handler library is already loaded	The provider library is previously loaded and unloaded. It is to be RE-loaded, while the request handler library is already loaded	Both provider library and the request library are already loaded
CreateInstance	59.97	31.59	25.10	18.62
EnumerateInstanceNames	51.72	23.75	18.34	11.45
EnumerateInstances	53.83	25.74	19.18	13.09
GetInstance	57.75	28.96	22.45	16.19
ModifyInstance	58.01	29.59	23.02	16.43
DeleteInstane	56.23	27.91	21.28	14.76

6.4.1.2 Tests against Number of Keys in openWBEM

Test 1: The provider returns a success response for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is string. The WBEM server is openWBEM server.

The minimum actions required in the methods: EnumerateInstanceNames and EnumerateInstances return empty list; the CreateInstance returns an CIMObjectPath from the incoming instance; the GetInstance returns an instance constructed by the given className; ModifyInstance and DeleteInstance immediately return with no action taken.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	25.36	26.16	29.34	33.29	36.64	40.22
EnumerateInstanceNames	11.49	11.79	12.64	13.28	13.75	14.53
EnumerateInstances	13.26	13.51	13.85	14.56	15.60	16.64
GetInstance	26.19	26.85	28.04	29.65	31.23	32.51
ModifyInstance	28.42	29.01	30.80	32.55	33.98	36.04
DeleteInstance	24.49	25.09	26.73	28.17	29.71	31.71

Test 2: The provider returns an exception for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is string. The WBEM server is openWBEM server.

The minimum actions required in the methods: all operations performs no action except to throw an FAILED exception with a message.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	16.30	17.90	20.42	24.00	27.36	30.70
EnumerateInstanceNames	11.46	11.47	11.20	11.34	11.33	11.24
EnumerateInstances	13.17	13.10	13.23	13.08	13.21	13.06
GetInstance	13.91	14.27	14.57	15.33	16.06	16.19
ModifyInstance	14.20	14.42	14.86	15.68	16.29	16.71
DeleteInstance	13.04	12.79	13.63	14.07	14.35	15.19

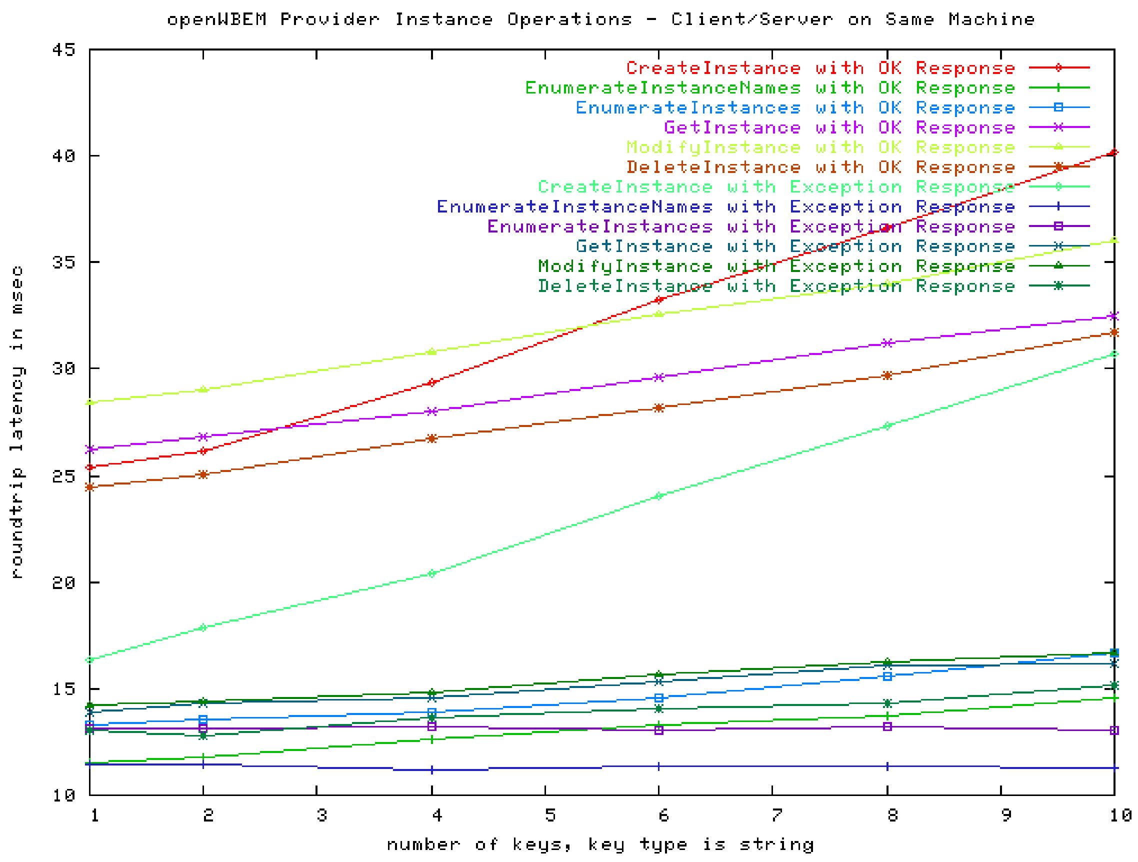


Figure 25: openWBEM Provider Instance Operation - Key Type is String

Test 3: The provider returns a success response for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is uint32. The WBEM server is openWBEM server.

The minimum actions required in the methods: EnumerateInstanceNames and EnumerateInstances return empty list; the CreateInstance returns an CIMObjectPath from the incoming instance; the GetInstance returns an instance constructed by the given className; ModifyInstance and DeleteInstance immediately return with no action taken.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	24.33	25.71	28.12	30.72	33.46	37.79
EnumerateInstanceNames	11.26	11.65	12.64	13.18	13.73	14.47
EnumerateInstances	13.05	13.27	13.82	14.69	15.53	16.35
GetInstance	26.47	27.25	29.63	31.77	33.83	36.05
ModifyInstance	28.60	29.80	32.04	34.88	36.85	39.19
DeleteInstance	24.92	26.04	28.39	31.01	33.33	35.91

Test 4: The provider returns an exception for all operations. The instance class is a subclass of `CIM_EnabledManagedElement` and it has four keys whose type is uint32. The WBEM server is openWBEM server.

The minimum actions required in the methods: all operations performs no action except to throw an FAILED exception with a message.

Number of keys in the instance	1	2	4	6	8	10
CreateInstance	16.05	17.02	18.62	21.39	23.68	26.48
EnumerateInstanceNames	11.46	11.46	11.45	11.28	11.08	11.40
EnumerateInstances	13.10	12.95	13.09	13.03	13.16	13.16
GetInstance	14.37	14.66	16.19	17.71	18.54	20.00
ModifyInstance	14.42	15.84	16.43	18.10	18.86	20.14
DeleteInstance	13.02	13.50	14.76	16.54	17.61	19.20

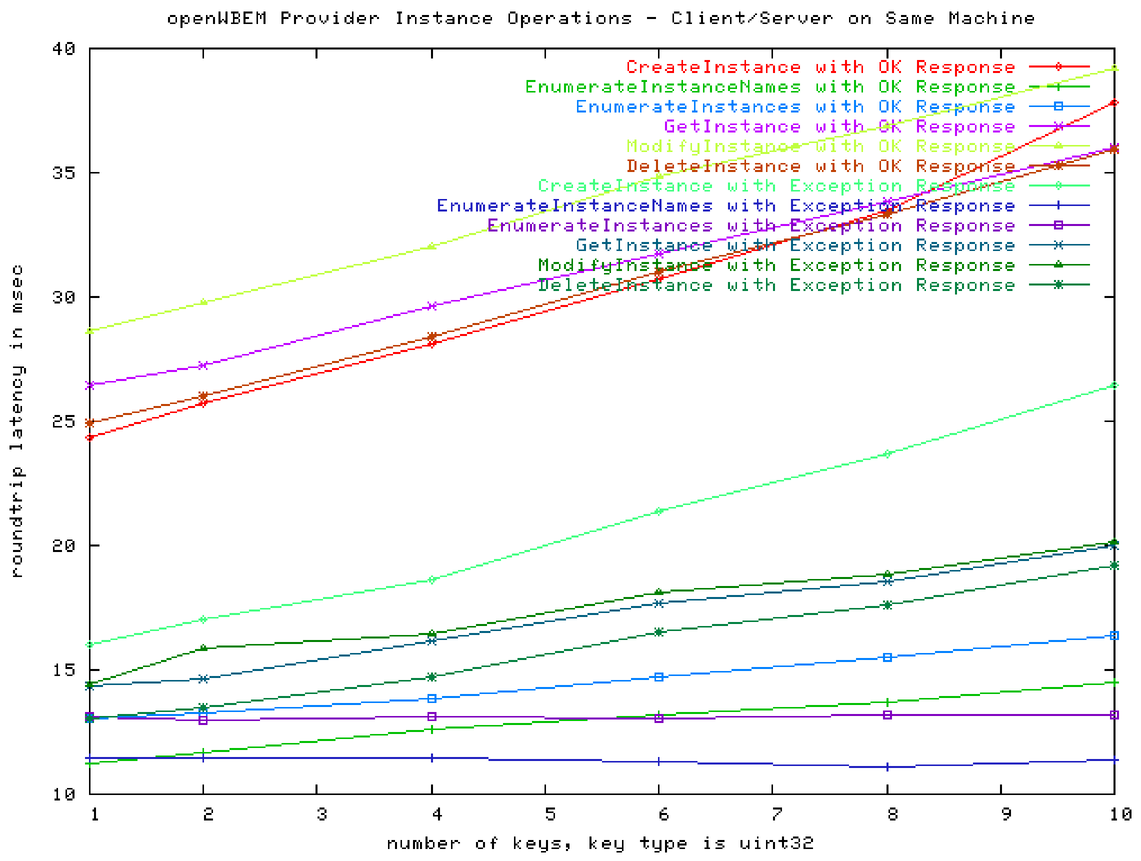


Figure 26: openWBEM Provider Instance Operation - Key Type is Uint32

Under the test condition 1, the openPegasus server has the following results: TODO

6.4.1.3 Tests in openPegasus

TODO: In Pegasus release 2.2, the providers are automatically unloaded after a very short random timeout. This is a known bug and is expected to be fixed in release 2.3. Thus, we are pending the bug fix and the provider tests for openPegasus are NOT performed.

Notes (Nov. 27, 2003): Release 2.3 has been downloaded and installed. The initial tests proved that the providers are NOT unloaded after a short random period. However, extensive tests are required to verify the bug fix.

6.4.2 Property Operations

TODO

6.4.3 Association Operations

TODO

6.4.4 Indication Operations

TODO

7 Future Work

The benchmark tests described in previous sections are performed under the condition of a single client connected to the WBEM server. The future work would include the benchmark tests under the condition of multiple clients connected to a WBEM server simultaneously. In this case, the throughput (how many messages per second) is measured and the result is compared with that of a single client.

8 References

[1] Chris Hobbs, "*Using Web-Based Enterprise Management*", Nortel Networks, 2003.

[2] Chris Hobbs and Ying Zeng, "*CIM Exploration – Writing a Pegasus CIM Provider*", IEEE Nortel Networks, October 2002.

[3] Ying Zeng, "*Writing a CIM Provider in openWBEM*", Nortel Networks, July 2003.

9 Appendix A - Sample XML Messages

Some sample XML requests for operations:

- EnumerateClassNames

The EnumerateClassNames operation returns the class names within the namespace specified in the request. A sample request in CIMXML would be as follows:

```
<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
 <MESSAGE ID="7873" PROTOCOLVERSION="1.0">
  <SIMPLEREQ>
   <IMETHODCALL NAME="EnumerateClassNames">
    <LOCALNAMESPACEPATH>
     <NAMESPACE NAME="root"/>
     <NAMESPACE NAME="widget"/>
    </LOCALNAMESPACEPATH>
    <IPARAMVALUE NAME="DeepInheritance">
     <VALUE>TRUE</VALUE>
    </IPARAMVALUE>
   </IMETHODCALL>
  </SIMPLEREQ>
 </MESSAGE>
</CIM>
```

- CreateClass

The CreateClass operation create a new class in the namespace in the repository. A sample request in CIMXML would be as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
 <MESSAGE ID="7" PROTOCOLVERSION="1.0">
  <SIMPLEREQ>
   <IMETHODCALL NAME="CreateClass">
    <LOCALNAMESPACEPATH>
     <NAMESPACE NAME="root"/>
     <NAMESPACE NAME="widget"/>
    </LOCALNAMESPACEPATH>
    <IPARAMVALUE NAME="NewClass">
     <CLASS NAME="BmTestClass" SUPERCLASS="CIM_System">
      <QUALIFIER NAME="Version" TYPE="string" >
       <VALUE>4.5.7</VALUE>
      </QUALIFIER>
      <QUALIFIER NAME="Description" TYPE="string" >
       <VALUE>Widget is a class which defines the Widget developed
       for use within all left-handed thronge units. Note that it
       should not be used as a superclass of any device which has
       been developed to support brillig dews.
       </VALUE>
      </QUALIFIER>
      <PROPERTY NAME="colour" TYPE="uint8" >
       <QUALIFIER NAME="Write" TYPE="boolean" >
        <VALUE>true</VALUE>
       </QUALIFIER>
       <QUALIFIER NAME="Description" TYPE="string" >
        <VALUE>Colour of the external box</VALUE>
       </QUALIFIER>
       <QUALIFIER NAME="ValueMap" TYPE="string" >
        <VALUE.ARRAY>
```

```
         <VALUE>0</VALUE>
         <VALUE>1</VALUE>
         <VALUE>2</VALUE>
        </VALUE.ARRAY>
       </QUALIFIER>
       <QUALIFIER NAME="Values" TYPE="string" >
        <VALUE.ARRAY>
         <VALUE>Red</VALUE>
         <VALUE>Green</VALUE>
         <VALUE>Blue</VALUE>
        </VALUE.ARRAY>
       </QUALIFIER>
      </PROPERTY>
      <PROPERTY NAME="packetCounter" TYPE="uint64" >
       <QUALIFIER NAME="Read" TYPE="boolean" >
        <VALUE>true</VALUE>
       </QUALIFIER>
       <QUALIFIER NAME="Description" TYPE="string" >
        <VALUE>Counter of Transmitted Packets</VALUE>
       </QUALIFIER>
      </PROPERTY>
      <PROPERTY NAME="style" TYPE="uint8" >
       <QUALIFIER NAME="Write" TYPE="boolean" >
        <VALUE>true</VALUE>
       </QUALIFIER>
       <QUALIFIER NAME="Description" TYPE="string" >
        <VALUE>Style of the outer casing</VALUE>
       </QUALIFIER>
       <QUALIFIER NAME="ValueMap" TYPE="string" >
        <VALUE.ARRAY>
         <VALUE>0</VALUE>
         <VALUE>1</VALUE>
         <VALUE>2</VALUE>
        </VALUE.ARRAY>
       </QUALIFIER>
       <QUALIFIER NAME="Values" TYPE="string" >
        <VALUE.ARRAY><VALUE>Modern</VALUE>
         <VALUE>Classical</VALUE>
         <VALUE>Baroque</VALUE>
        </VALUE.ARRAY>
       </QUALIFIER>
      </PROPERTY>
     </CLASS>
    </IPARAMVALUE>
   </IMETHODCALL>
  </SIMPLEREQ>
 </MESSAGE>
</CIM>
```

- GetClass

The GetClass operation retrieve the inheritance, properties, qualifiers and methods of the specified class. A sample CIMXML request is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
 <MESSAGE ID="1001" PROTOCOLVERSION="1.0">
  <SIMPLEREQ>
   <IMETHODCALL NAME="GetClass">
    <LOCALNAMESPACEPATH>
     <NAMESPACE NAME="root"/>
     <NAMESPACE NAME="widget"/>
    </LOCALNAMESPACEPATH>
    <IPARAMVALUE NAME="ClassName">
     <CLASSNAME NAME="BmTestClass"/>
    </IPARAMVALUE>
    <IPARAMVALUE NAME="LocalOnly">
     <VALUE>FALSE</VALUE>
    </IPARAMVALUE>
    <IPARAMVALUE NAME="IncludeQualifiers">
     <VALUE>TRUE</VALUE>
    </IPARAMVALUE>
   </IMETHODCALL>
  </SIMPLEREQ>
 </MESSAGE>
</CIM>
```

- DeleteClass

The DeleteClass operation removes a class from the given namespace in the repository. A sample CIMXML request is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
 <MESSAGE ID="1001" PROTOCOLVERSION="1.0">
  <SIMPLEREQ>
   <IMETHODCALL NAME="DeleteClass">
    <LOCALNAMESPACEPATH>
     <NAMESPACE NAME="root"/>
     <NAMESPACE NAME="widget"/>
    </LOCALNAMESPACEPATH>
    <IPARAMVALUE NAME="ClassName">
     <CLASSNAME NAME="BmTestClass"/>
    </IPARAMVALUE>
   </IMETHODCALL>
  </SIMPLEREQ>
 </MESSAGE>
</CIM>
```

10 Appendix B - Instrumentation in the wbemexec Utility

The wbemexec is a simple command line utility to read the CIMXML request from standard input, send CIMXML message, receive the CIMXML response and output the response to standard output. The following is the function (method) in which wbemexec command sends CIMXML request and receives the CIMXML response.

```
/**

    Executes the command using HTTP.  A CIM request encoded in XML is read
    from the input, and encapsulated in an HTTP request message.  A channel
    is obtained for an HTTP connection, and the message is written to the
    channel.  The response is written to the specified outPrintWriter, and
    consists of the CIM response encoded in XML.

    @param   outPrintWriter     the ostream to which output should be
                                written
    @param   errPrintWriter     the ostream to which error output should be
                                written

    @exception  WbemExecException  if an error is encountered in executing
                                   the command

 */
void WbemExecCommand::_executeHttp (ostream& outPrintWriter,
                                    ostream& errPrintWriter)
    throw (WbemExecException)
{
    Uint32                       size;
    Array <Sint8>                content;
    Array <Sint8>                contentCopy;
    Array <Sint8>                message;
    Array <Sint8>                httpHeaders;
    Array <Sint8>                httpResponse;
    WbemExecClient client;

    client.setTimeout( _timeout );

    // Read in the CIMXML request from a file and perform syntax checks, etc.
    // ……

    //
    //  Make a copy of the content because the XmlParser constructor
    //  modifies the text
    //
    contentCopy << content;

    XmlParser parser ((char*) contentCopy.getData ());

    // Here is the place to insert the timestamp as the beginning of the
operation

    try
    {
        _connectToServer( client, outPrintWriter );

        //
        //  Encapsulate XML request in an HTTP request
        //
        message = XMLProcess::encapsulate( parser, _hostName,
                                           _useMPost, _useHTTP11,
                                           content, httpHeaders );
        if (_debugOutput1)
        {
          outPrintWriter << message.getData () << endl;
        }
    }
    catch (XmlException& xe)
    {
        WbemExecException e(WbemExecException::INVALID_XML, xe.getMessage ());
        throw e;
    }
    catch (WbemExecException& e)
    {
        throw e;
    }
    catch (Exception& ex)
    {
        WbemExecException e(WbemExecException::CONNECT_FAIL, ex.getMessage ());
        throw e;
    }

    try
    {
        httpResponse = client.issueRequest( message );
    }
    catch (ConnectionTimeoutException& ex)
    {
        WbemExecException e
            (WbemExecException::TIMED_OUT);
        throw e;
    }
    catch (UnauthorizedAccess& ex)
    {
        WbemExecException e
            (WbemExecException::CONNECT_FAIL, ex.getMessage ());
        throw e;
    }
    catch (Exception& ex)
    {
        WbemExecException e
            (WbemExecException::CONNECT_FAIL, ex.getMessage ());
        throw e;
    }

    // Here is the place to insert the timestamp as the end of the operation

    //
    // Process the response message
    //
    _handleResponse( httpResponse, outPrintWriter, errPrintWriter );
}
```

A Perl script invokes the wbemexec command for n (n=10000?) times. The roundtrip time is accumulated each time when the operation is successfully executed. When the script finishes invoking wbemexec, the accumulated time is divided by the total number of successful operations. The final result for a single roundtrip time would be the average of the time from all the successful operations.

11 Appendix C - Bugs/Problems in WBEM Implementations

In this appendix, we created a list of bugs and potential problems with the WBEM servers. The listed bugs and problems may not critical and some of them have workarounds, but they can sometimes cause strange CIMOM behaviours or affect the CIMOM performance significantly. Thus, we generated such a list to let developers be aware of these problems.

	Problem	**Affected Features**	**WBEM Server**	**Status**
1	The incoming instance key array and property array were NOT synchronized properly in CIMOM	CreateInstance in repository	openWBEM 2.0.6	Bug/Fixed in 2.0.12
2	The incoming instance key array and property array were NOT synchronized properly in CIMOM	CreateInstance in provider	openWBEM 2.0.6	Bug/Fixed in 2.0.12
3	The EnumerateClassNames operation performs the unnecessary de-serialization of classes	EnumerateClassNames in repository	openWBEM 2.0.6	Bug/Fixed 2.0.12
4	The repository is implemented using XML files, which cause the repository operations become very inefficient when there are a large number of class or instances	All repository operations except EnumerateClassNames, EnumerateInstanceNames	openPegasus 2.2	Can be a big potential problem in future design; good for debug but not practical. No current proposal on fix.
5	The providers are unloaded in a very short period of time automatically in Pegasus.	All types of provider operations including indications	openPegasus 2.2	Bug/Fixed in release 2.3; not yet tested locally.
6	OpenPegasus allows a provider for a standard CIM class, the repository and the provider can both have instance with the same keys. The getInstance operation returns two instances with the same keys: one from repository and one from the provider.	Instance operations in repository and instance provider	openPegasus 2.2	Not investigated yet.
7	OpenPegasus repository do NOT handle association functions of an instance correctly	Association operations on an instance in the repository	openPegasus 2.2	Bug with Nortel's own fixes.

12 Appendix D - Sample Test Providers and MOFs

12.1 Sample MOF for Instance Provider Tests

```
[Description (
       "Benchmark Instance Provider Test"),
        provider("c++::InstanceProvider") ]
class BM_4_str_Class : CIM_EnabledLogicalElement
    {
    [Key, Description("Key 1 for benchmark test") ]
    string Key_1;

    [Key, Description("Key 2 for benchmark test") ]
    string Key_2;

    [Key, Description("Key 3 for benchmark test") ]
    string Key_3;

    [Key, Description("Key 4 for benchmark test") ]
    string Key_4;

    [Min(0), Max(2),
    Description ("For benchmark test purpose. If its value is set to 0, "
    "then the provider processing elapsed time is accumulated. "
    "If its value is set to 1, "
    "it marks the end of the benchmark tests, and the provider will print "
    "the elapsed time for internal provider data processing. "
    "If the value is set to 2, then the elapsed time inside the provider is
reset to 0.") ]
    uint32 testProperty;
    };


[Description (
       "Benchmark Instance Provider Test"),
        provider("c++::InstanceProvider") ]
class BM_4_int_Class : CIM_EnabledLogicalElement
    {
    [Key, Description("Key 1 for benchmark test") ]
    uint32 Key_1;

    [Key, Description("Key 2 for benchmark test") ]
    uint32 Key_2;

    [Key, Description("Key 3 for benchmark test") ]
    uint32 Key_3;

    [Key, Description("Key 4 for benchmark test") ]
    uint32 Key_4;

    [Min(0), Max(2),
    Description ("For benchmark test purpose. If its value is set to 0, "
    "then the provider processing elapsed time is accumulated. "
    "If its value is set to 1, "
    "it marks the end of the benchmark tests, and the provider will print "
    "the elapsed time for internal provider data processing. "
    "If the value is set to 2, then the elapsed time inside the provider is
reset to 0.") ]
    uint32 testProperty;
    };
```

12.2 OpenWBEM Sample Instance Provider for Condition A

The following is an instance provider sample used in the instance provider tests under condition A:

```
// *****************************************************
//  file    instanceProvider.cpp for class BmTestClass
//           for the CIMOM benchmark tests
//  written September 2003
//  author  Ying Zeng
//  history
//        v1.0 September 2003
// *****************************************************

#include "instanceProvider.h"

namespace
{

// *************************************************
// class    InstanceProvider
// method   InstanceProvider
// purpose  constructor
// *************************************************

InstanceProvider::InstanceProvider(void)
    {
    return;
    }

// *************************************************
// class    InstanceProvider
// method   ~InstanceProvider
// purpose  destructor
// *************************************************

InstanceProvider::~InstanceProvider(void)
    {
    return;
    }

// *************************************************
// class    InstanceProvider
// method   initialize
// purpose  initialises the environment
//           reference
// input    reference to originator of
//           this provider's CIMOM handle
// note          called by the CIM server after we are
//           created
// *************************************************

void
InstanceProvider::initialize(const OW_ProviderEnvironmentIFCRef& env)
    {
    (void)env;

    return;
    }

// *************************************************
// class    InstanceProvider
// method   cleanup
// purpose  frees up all the memory that this
//           provider has used
// note          called by the CIM server before we are
//           destroyed
// ************************************************

void InstanceProvider::cleanup()
    {
    return;
    }

// ************************************************
// class    InstanceProvider
// method   enumInstanceNames
// purpose  standard openWBEM function to handle
//           the enumInstanceNames call
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

void InstanceProvider::enumInstanceNames(
          const OW_ProviderEnvironmentIFCRef& env,
          const OW_String& ns,
          const OW_String& className,
          OW_CIMObjectPathResultHandlerIFC& result,
          const OW_Bool& deep,
          const OW_CIMClass& cimClass)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables

    (void)env;
    (void)deep;
    (void)cimClass;
    (void)className;
    (void)result;
    (void)ns;

    return;
    }

// ************************************************
// class    InstanceProvider
// method   enumInstances
// purpose  handle standard openWBEM enumInstances
//           call
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

void InstanceProvider::enumInstances(
          const OW_ProviderEnvironmentIFCRef& env,
          const OW_String& ns,
          const OW_String& className,
          OW_CIMInstanceResultHandlerIFC& result,
          const OW_Bool& deep,
          const OW_CIMClass& cimClass,
          const OW_Bool& localOnly)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables

    (void)env;
    (void)ns;
    (void)className;
    (void)result;
    (void)deep;
    (void)cimClass;
    (void)localOnly;

    return;
    }


// ************************************************
// class    InstanceProvider
// method   getInstance
// purpose  standard openWBEM call for handling
//           getInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

OW_CIMInstance InstanceProvider::getInstance(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_CIMObjectPath& instanceName,
           const OW_CIMClass& cimClass,
           const OW_Bool& localOnly)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables
    (void)env;
    (void)ns;
    (void)instanceName;
    (void)cimClass;
    (void)localOnly;

    OW_CIMInstance rval = OW_CIMInstance(instanceName.getObjectName());
    rval.setClassName(instanceName.getObjectName());

    return rval;
    }

// ************************************************
// class    InstanceProvider
// method   createInstance
// purpose  standard openWBEM function for handling
//           call to createInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

OW_CIMObjectPath InstanceProvider::createInstance(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_CIMInstance& cimInstance)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables
    (void)env;
    (void)ns;
    (void)cimInstance;
    OW_CIMObjectPath op(ns, cimInstance);
    op.setKeys(cimInstance.getKeyValuePairs());

    return op;
    }

// ************************************************
// class    InstanceProvider
// method   deleteInstance
// purpose  standard openWBEM function for handling
//           call to deleteInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

void InstanceProvider::deleteInstance(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_CIMObjectPath& cop)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables

    (void)env;
    (void)ns;
    (void)cop;

    return;
    }

// ***********************************************
// class    InstanceProvider
// method   modifyInstance
// purpose  standard openWBEM function for handling
//           call to modifyInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ***********************************************

void InstanceProvider::modifyInstance(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_CIMInstance& modifiedInstance)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables
    (void)env;
    (void)ns;
    (void)modifiedInstance;

    return;
    }

}
// specify the actual provider library for the CIMOM
OW_PROVIDERFACTORY(InstanceProvider, InstanceProvider)
```

12.3 OpenWBEM Sample Instance Provider for Condition B

The following is an instance provider sample used in the instance provider tests under condition B:

```
// ***************************************************
//  file    instanceProvider.cpp for class BmTestClass
//           for the CIMOM benchmark tests
//  written September 2003
//  author  Ying Zeng
//  history
//          v1.0 September 2003
// ***************************************************

#include "instanceProvider.h"

namespace
{

// ***********************************************
// class    InstanceProvider
// method   InstanceProvider
// purpose  constructor
// ***********************************************

InstanceProvider::InstanceProvider(void)
    {
/*
    cout << "********** In constructor for InstanceProvider " <<
                 " for the first time *********" << endl;
*/
    bmDataPath=OW_CIMObjectPath(dataclassname);
    bmDataPath.addKey("Name", OW_CIMValue("Benchmark"));
    bmDataPath.addKey("CreationClassName", OW_CIMValue(dataclassname));
    FILE    *fp;
    fp = fopen("/usr/local/lib/openwbem/c++providers/bmResult.txt","w");
    fclose(fp);

    }

// ************************************************
// class    InstanceProvider
// method   ~InstanceProvider
// purpose  destructor
// ************************************************

InstanceProvider::~InstanceProvider(void)
    {
/*
    cout<<"********** In InstanceProvider destructor **********"<<endl;
*/
    return;
    }

// ************************************************
// class    InstanceProvider
// method   initialize
// purpose  initialises the environment
//           reference
// input    reference to originator of
//           this provider's CIMOM handle
// note          called by the CIM server after we are
//           created
// ************************************************

void
InstanceProvider::initialize(const OW_ProviderEnvironmentIFCRef& env)
    {
    (void)env;

    env->getLogger()->logDebug("InstanceProvider initialize called");
    cout<<"InstanceProvider::initialize"<<endl;
    myEnv = env;
    OW_CIMOMHandleIFCRef hdl = myEnv->getCIMOMHandle();
    OW_CIMInstance inst=hdl->getInstance(defaultns, bmDataPath);
    inst.setProperty("CiElapsedTime",OW_CIMValue(0));
    inst.setProperty("ECiElapsedTime", OW_CIMValue(0));
    inst.setProperty("GiElapsedTime", OW_CIMValue(0));
    inst.setProperty("EGiElapsedTime", OW_CIMValue(0));
    inst.setProperty("EnElapsedTime", OW_CIMValue(0));
    inst.setProperty("EEnElapsedTime", OW_CIMValue(0));
    inst.setProperty("EiElapsedTime", OW_CIMValue(0));
    inst.setProperty("EEiElapsedTime", OW_CIMValue(0));
    inst.setProperty("MiElapsedTime", OW_CIMValue(0));
    inst.setProperty("EMiElapsedTime", OW_CIMValue(0));
    inst.setProperty("DiElapsedTime", OW_CIMValue(0));
    inst.setProperty("EDiElapsedTime", OW_CIMValue(0));
    inst.setProperty("CiCount", OW_CIMValue(0));
    inst.setProperty("ECiCount", OW_CIMValue(0));
    inst.setProperty("GiCount", OW_CIMValue(0));
    inst.setProperty("EGiCount", OW_CIMValue(0));
    inst.setProperty("EnCount", OW_CIMValue(0));
    inst.setProperty("EEnCount", OW_CIMValue(0));
    inst.setProperty("EiCount", OW_CIMValue(0));
    inst.setProperty("EEiCount", OW_CIMValue(0));
    inst.setProperty("MiCount", OW_CIMValue(0));
    inst.setProperty("EMiCount", OW_CIMValue(0));
    inst.setProperty("DiCount", OW_CIMValue(0));
    inst.setProperty("EDiCount", OW_CIMValue(0));
    hdl->modifyInstance(defaultns, inst);

    return;
    }
```

```
// ************************************************
// class    InstanceProvider
// method   cleanup
// purpose  frees up all the memory that this
//            provider has used
// note           called by the CIM server before we are
//            destroyed
// ************************************************

void InstanceProvider::cleanup()
    {

    myEnv->getLogger()->logDebug("InstanceProvider cleanup called");
    cout<<"InstanceProvider::cleanup"<<endl;

    }

// ************************************************
// class    InstanceProvider
// method   enumInstanceNames
// purpose  standard openWBEM function to handle
//            the enumInstanceNames call
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

void InstanceProvider::enumInstanceNames(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_String& className,
           OW_CIMObjectPathResultHandlerIFC& result,
           const OW_Bool& deep,
           const OW_CIMClass& cimClass)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables

    (void)env;
    (void)deep;
    (void)cimClass;
    (void)className;
    (void)result;
    (void)ns;

    timeval startT;
    timeval endT;
    OW_UInt64 elapsedT;
    // OW_CIMObjectPath op;
    unsigned int i;

    //cout << "In InstanceProvider::enumInstanceNames " << endl;
    gettimeofday(&startT, NULL);
    OW_CIMOMHandleIFCRef hdl = myEnv->getCIMOMHandle();
    OW_CIMInstance inst = hdl->getInstance(defaultns, bmDataPath);

    for (i=0; i<myInstances.size(); i++)
    {
        OW_CIMObjectPath op(ns, myInstances[i]);
        result.handle(op);
    }

    OW_UInt32 n;
    inst.getProperty("EnCount").getValue().get(n);
    inst.setProperty("EnCount",OW_CIMValue(++n));
    OW_UInt64 t;
    inst.getProperty("EnElapsedTime").getValue().get(t);
    gettimeofday(&endT, NULL);
    elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-startT.tv_usec;
    inst.setProperty("EnElapsedTime",OW_CIMValue(t+elapsedT));
    hdl->modifyInstance(defaultns, inst);

    return;
    }

// ***********************************************
// class    InstanceProvider
// method   enumInstances
// purpose  handle standard openWBEM enumInstances
//           call
// input    as defined by openWBEM
// output   as defined by openWBEM
// ***********************************************

void InstanceProvider::enumInstances(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_String& className,
           OW_CIMInstanceResultHandlerIFC& result,
           const OW_Bool& deep,
           const OW_CIMClass& cimClass,
           const OW_Bool& localOnly)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables

    (void)env;
    (void)ns;
    (void)className;
    (void)result;
    (void)deep;
    (void)cimClass;
    (void)localOnly;

    timeval startT;
    timeval endT;
    OW_UInt64 elapsedT;
    unsigned int i;

    //cout << "In InstanceProvider::enumInstances " << endl;
    gettimeofday(&startT, NULL);
    OW_CIMOMHandleIFCRef hdl = myEnv->getCIMOMHandle();
    OW_CIMInstance inst = hdl->getInstance(defaultns, bmDataPath);

    for (i=0; i<myInstances.size(); i++)
    {
        result.handle(myInstances[i]);
    }

    OW_UInt32 n;
    inst.getProperty("EiCount").getValue().get(n);
    inst.setProperty("EiCount",OW_CIMValue(++n));
    OW_UInt64 t;
    inst.getProperty("EiElapsedTime").getValue().get(t);
    gettimeofday(&endT, NULL);
    elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-startT.tv_usec;
    inst.setProperty("EiElapsedTime",OW_CIMValue(t+elapsedT));
    hdl->modifyInstance(defaultns, inst);

    return;
    }


// ***********************************************
// class    InstanceProvider
// method   getInstance
// purpose  standard openWBEM call for handling
//           getInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ***********************************************

OW_CIMInstance InstanceProvider::getInstance(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
            const OW_CIMObjectPath& instanceName,
            const OW_CIMClass& cimClass,
            const OW_Bool& localOnly)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables
    (void)env;
    (void)ns;
    (void)instanceName;
    (void)cimClass;
    (void)localOnly;

    OW_CIMInstance rval = OW_CIMInstance(instanceName.getObjectName());
    rval.setClassName(instanceName.getObjectName());

    timeval startT;
    timeval endT;
    OW_UInt64 elapsedT;
    OW_CIMObjectPath op;
    unsigned int i;
    unsigned int flag=0;
    OW_CIMObjectPath tp;

    //cout << "In InstanceProvider::getInstance " << endl;
    gettimeofday(&startT, NULL);
    OW_CIMOMHandleIFCRef hdl = myEnv->getCIMOMHandle();
    OW_CIMInstance inst = hdl->getInstance(defaultns, bmDataPath);

    for (i=0; i<myInstances.size(); i++)
    {
        op=OW_CIMObjectPath(ns, myInstances[i]);
        // This is to avoid the default root namespace problem in openWBEM
        tp=instanceName;
        tp.setNameSpace(defaultns);
        if (op.equals(tp))
        {
            OW_UInt32 n;
            inst.getProperty("GiCount").getValue().get(n);
            inst.setProperty("GiCount",OW_CIMValue(++n));
            flag = 1; // found the instance
            rval= myInstances[i];
            break;
        }
    }

    if (flag ==0)
    {
        OW_UInt32 n;
        inst.getProperty("EGiCount").getValue().get(n);
        inst.setProperty("EGiCount", OW_CIMValue(++n));
        OW_UInt64 t;
        inst.getProperty("EGiElapsedTime").getValue().get(t);
        gettimeofday(&endT, NULL);
        elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-
startT.tv_usec;
        inst.setProperty("EGiElapsedTime", OW_CIMValue(t+elapsedT));
        hdl->modifyInstance(defaultns, inst);
        OW_THROWCIMMSG(OW_CIMException::NOT_FOUND,
             "The requested test instance is not found");
    }

    OW_UInt64 t;
    inst.getProperty("GiElapsedTime").getValue().get(t);
    gettimeofday(&endT, NULL);
    elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-startT.tv_usec;
    inst.setProperty("GiElapsedTime",OW_CIMValue(t+elapsedT));
    hdl->modifyInstance(defaultns, inst);

    return rval;
    }

// **************************************************
// class    InstanceProvider
// method   createInstance
// purpose  standard openWBEM function for handling
//            call to createInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

OW_CIMObjectPath InstanceProvider::createInstance(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_CIMInstance& cimInstance)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables
    (void)env;
    (void)ns;
    (void)cimInstance;
    OW_CIMObjectPath op(ns, cimInstance);
    op.setKeys(cimInstance.getKeyValuePairs());

    timeval startT;
    timeval endT;
    OW_UInt64 elapsedT;
    OW_CIMObjectPath tmpop;
    unsigned int i;

    //cout << "In InstanceProvider::createInstance " << endl;
    gettimeofday(&startT, NULL);
    OW_CIMOMHandleIFCRef hdl = myEnv->getCIMOMHandle();
    OW_CIMInstance inst = hdl->getInstance(defaultns, bmDataPath);
    OW_CIMObjectPath op(ns, cimInstance);
    for (i=0; i<myInstances.size(); i++)
    {
        tmpop=OW_CIMObjectPath(ns, myInstances[i]);
       if (tmpop.equals(op))
        {
            OW_UInt32 n;
            inst.getProperty("ECiCount").getValue().get(n);
            inst.setProperty("ECiCount", OW_CIMValue(++n));
            OW_UInt64 t;
            inst.getProperty("ECiElapsedTime").getValue().get(t);
            gettimeofday(&endT, NULL);
            elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-
startT.tv_usec;
            inst.setProperty("ECiElapsedTime", OW_CIMValue(t+elapsedT));
            hdl->modifyInstance(defaultns, inst);
            OW_THROWCIMMSG(OW_CIMException::ALREADY_EXISTS,
                 "Unable to create instance: instance already exists");
        }
    }

    myInstances.append(cimInstance);
    OW_UInt32 n;
    inst.getProperty("CiCount").getValue().get(n);
    inst.setProperty("CiCount",OW_CIMValue(++n));
    OW_UInt64 t;
    inst.getProperty("CiElapsedTime").getValue().get(t);
    gettimeofday(&endT, NULL);
    elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-startT.tv_usec;
    inst.setProperty("CiElapsedTime",OW_CIMValue(t+elapsedT));
    hdl->modifyInstance(defaultns, inst);

    return op;
    }

// ************************************************
// class    InstanceProvider
// method   deleteInstance
// purpose  standard openWBEM function for handling
//            call to deleteInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

void InstanceProvider::deleteInstance(
           const OW_ProviderEnvironmentIFCRef& env,
           const OW_String& ns,
           const OW_CIMObjectPath& cop)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables

    (void)env;
    (void)ns;
    (void)cop;

    timeval startT;
    timeval endT;
    OW_UInt64 elapsedT;
    unsigned int i;

    //cout << "In InstanceProvider::deleteInstance " << endl;
    gettimeofday(&startT, NULL);
    OW_CIMOMHandleIFCRef hdl = myEnv->getCIMOMHandle();
    OW_CIMInstance inst = hdl->getInstance(defaultns, bmDataPath);
    OW_CIMObjectPath tmpop = cop;
    tmpop.setNameSpace(ns);

    for (i=0; i<myInstances.size(); i++)
    {
        OW_CIMObjectPath op(ns, myInstances[i]);
        if (op.equals(tmpop))
       {
            myInstances.remove(i);
            OW_UInt32 n;
            inst.getProperty("DiCount").getValue().get(n);
            inst.setProperty("DiCount", OW_CIMValue(++n));
            OW_UInt64 t;
            inst.getProperty("DiElapsedTime").getValue().get(t);
            gettimeofday(&endT, NULL);
            elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-
startT.tv_usec;
            inst.setProperty("DiElapsedTime", OW_CIMValue(t+elapsedT));
            hdl->modifyInstance(defaultns, inst);
            return;
        }
    }
    OW_UInt32 n;
    inst.getProperty("EDiCount").getValue().get(n);
    inst.setProperty("EDiCount", OW_CIMValue(++n));
    OW_UInt64 t;
    inst.getProperty("EDiElapsedTime").getValue().get(t);
    gettimeofday(&endT, NULL);
    elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec -
startT.tv_usec;;
    inst.setProperty("EDiElapsedTime", OW_CIMValue(t+elapsedT));
    hdl->modifyInstance(defaultns, inst);
    OW_THROWCIMMSG(OW_CIMException::FAILED,
             "Unable to delete the requested instance: instance not found");

    return;
    }

// ************************************************
// class    InstanceProvider
// method   modifyInstance
// purpose  standard openWBEM function for handling
//           call to modifyInstance
// input    as defined by openWBEM
// output   as defined by openWBEM
// ************************************************

void InstanceProvider::modifyInstance(
          const OW_ProviderEnvironmentIFCRef& env,
          const OW_String& ns,
          const OW_CIMInstance& modifiedInstance)
    {
    // silly statements to get rid of compiler
    // warnings about unused variables
    (void)env;
    (void)ns;
    (void)modifiedInstance;

    timeval startT;
    timeval endT;
    OW_UInt64 elapsedT;
    unsigned int i;
    unsigned int flag=0;

    //cout << "In InstanceProvider::modifyInstance " << endl;
    gettimeofday(&startT, NULL);
    OW_CIMOMHandleIFCRef hdl = myEnv->getCIMOMHandle();
    OW_CIMInstance inst = hdl->getInstance(defaultns, bmDataPath);
    OW_CIMObjectPath op(defaultns, modifiedInstance);
    for (i=0; i<myInstances.size(); i++)
    {
        OW_CIMObjectPath tmp(defaultns, myInstances[i]);
        if (tmp.equals(op))
        {
            flag=1;
            myInstances[i].setProperties(modifiedInstance.getProperties());
            OW_UInt32 n;
            inst.getProperty("MiCount").getValue().get(n);
            inst.setProperty("MiCount", OW_CIMValue(++n));
            OW_UInt64 t;
            inst.getProperty("MiElapsedTime").getValue().get(t);
            gettimeofday(&endT, NULL);
            elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-
startT.tv_usec;
            inst.setProperty("MiElapsedTime", OW_CIMValue(t+elapsedT));
            hdl->modifyInstance(defaultns, inst);
            return;
        }
    }
    if (flag == 0)
    {
        OW_UInt32 n;
        inst.getProperty("EMiCount").getValue().get(n);
        inst.setProperty("EMiCount", OW_CIMValue(++n));
        OW_UInt64 t;
        inst.getProperty("EMiElapsedTime").getValue().get(t);
        gettimeofday(&endT, NULL);
        elapsedT = (endT.tv_sec-startT.tv_sec)*10000000+endT.tv_usec-
startT.tv_usec;
        inst.setProperty("EMiElapsedTime", OW_CIMValue(t+elapsedT));
        hdl->modifyInstance(defaultns, inst);
        OW_THROWCIMMSG(OW_CIMException::FAILED,
                 "Unable to modify the requested instance: instance not found");
    }

    return;
    }

}
// specify the actual provider library for the CIMOM
OW_PROVIDERFACTORY(InstanceProvider, InstanceProvider)
```