15th Triennial World Congress, Barcelona, Spain

WMI BASED PERFORMANCE ANALYSIS FOR DISTRIBUTED SYSTEMS

Mitica Manu[1], Theodor Borangiu[2], Daniel Roman[3]

[1] Visual Studio, Developer Division, Microsoft,One Microsoft Way, Redmond, WA, USA
e-mail: miticam@microsoft.com
[2] Department of Control and Computers, POLITEHNICA University of Bucharest,
Spl. Idependentei 313, sector 6, 77206 Bucharest, ROMANIA,
e-mail: borangiu@icar.cimr.pub.ro
[3] DirectX, Windows Division, Microsoft, One Microsoft Way, Redmond, 98052, USA
e-mail: danielro@microsoft.com

Abstract: Starting from the difficulty to manage enterprise systems, applications, and networks a group of IT companies created the WBEM initiative, base of the Microsoft's WMI implementation. The current paper presents from a practical approach both an architecture design and its implementation for managing distributed systems based on a WMI backbone. The final result was a N-Tier application designed to acquire, store and retrieve functional and performance data from WMI providers without neither overloading targeted systems nor sacrificing the scalability and the reliability. The layers and the dataflow are also presented to provide a better understanding of the whole architecture. *Copyright © 2002 IFAC*

Keywords: Distributed Computer Control Systems, Real Time Systems, Performance, Tests, Monitoring, Management Systems

1. INTRODUCTION

One of the greatest challenges facing information technology managers is managing enterprise systems, applications, and networks as they become larger and more complex. Starting from this problem, the Web-Based Enterprise Management (WBEM) initiative was conceived and created by BMC Software Inc., Cisco Systems Inc., Compaq Computer Corp., Intel Corp., and Microsoft Corp as a possible solution. Later, to this initiative adhered most of the biggest companies in the IT area, including Sun that has recently announced that future versions of Solaris will include a WBEM implementation.

The core concept is the *Common Information Model* that is, as presented in (CIM ***, 1998a; Maston, 1999; ***, 1998c), an object model schema developed within the *Distributed Management Task Force* (DMTF), an industry-based standards body. Development and extension of the CIM model are supported by a number of working groups within the DMTF that focus on different areas of the schema, including systems, devices, network, distributed applications, and cost of ownership. The CIM Specification describes the modeling language, naming, and mapping techniques used to collect and transfer information from data providers and other management models. The CIM schema provides the actual model descriptions and information framework. It defines a set of classes with properties and associations, making it possible to organize information about the managed environment.

The CIM schema serves as the basis for modeling managed objects for the WBEM initiative. WBEM is an industry initiative within the DMTF that leverages many existing management and Internet technologies to deliver a unified way to manage the enterprise computing environment. WBEM is an initiative that proposes a set of standards for managing the enterprise systems and devices.

Windows Management Instrumentation (WMI) is Microsoft's implementation of the WBEM initiative and, at the same time, is the reference WBEM implementation. WMI implements the CIM schema and adds extended information about Windows platforms to the model. WMI serves as a key

component of the Microsoft Management strategy, which includes other technologies such as the Active Directory® and Microsoft Management Console (MMC). WMI is available today as a standard part of Windows® 2000 and any future Windows operating systems while they are also freely available for Windows NT® 4.0, Windows 95/98 and Embedded NT as well.

Figure 1 depicts WMI from an architectural perspective, describing the three-layer model WMI uses, which consists of providers, the CIM Object Manager (CIMOM), and consumers of WMI information.

Working from the lowest level upward, the first tier is the *provider*. A provider is an intermediate agent between the system to be managed (for example, operating system, service, application, device driver, and so on) and the CIM object manager. The job of a provider is to extract management information from the underlying data source using whatever interfaces that software presents for management. The management information and interfaces are then mapped by the provider into the object classes that WMI presents to WMI consumers. Moving forward, new and updated managed systems will use providers as a direct way to expose management APIs without significant intermediate conversions.

Next is CIMOM, the CIM Object Manager, which has its own storage repository and acts as a broker for object requests. CIMOM and its repository are represented on the system by the system service called WinMgmt. Providers plug into CIMOM via a published set of COM interfaces. CIMOM keeps track of what classes are available (their definitions are stored in the repository) and what provider is responsible for supplying instances of those classes. When a request for management information comes from a WMI consumer to CIMOM, it evaluates the request, identifies which provider has the information, and upon getting it returns the data to the consumer.

The consumer only needs to ask for the information it wants but at no time needs to know the exact source of it or any details of how it is extracted from the underlying API.

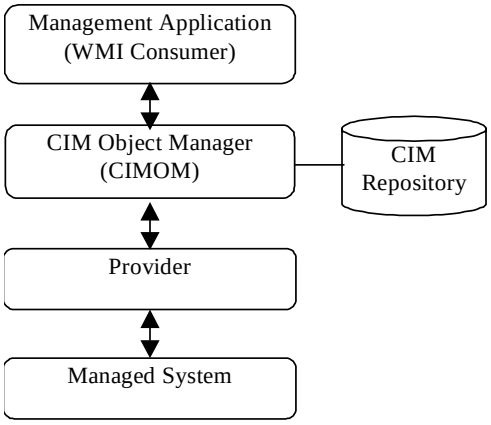

Fig. 1. WMI architecture

It should be noted that static data can be stored in the repository and retrieved without a provider, but the real power of the WMI system is that it supplies *dynamic* information about the managed system, and this is done entirely through providers.

Finally, there are consumers of WMI data. These can be management tools such as MMC snap-ins, management applications like Microsoft Systems Management Server (SMS) or third-party applications or scripts. These consumers, as previously noted, only need to know about the classes of the objects about which they wish to get information.

There are several types of information that can be defined in WMI classes:

- **Properties.** These supply descriptive or operational information about a particular instance of a class.
- **Methods.** For a given instance of a class, these are actions you can execute upon that instance.
- **Events.** These are notifications a consumer can request to receive for interesting occurrences or failures in the system. Any change to a defined property can be used as the basis of an event.
- **Associations.** An association describes a relationship between classes and is in itself defined by a class.

2. DISTRIBUTED SYSTEM MANAGEMENT USING WMI

2.1 Defining the problem

Nowadays we witness a shift from the client-server model to the distributing network computing architecture (Coulouris et al, 1999). The software industry at Microsoft is heading in this direction with its new .NET technology. Distributed computing has promised to make servers work together seamlessly as a unit, yielding better fault-tolerance and easier, less-expensive scalability than can be achieved with a physically centralized computing environment.
The biggest challenge is to manage this distributed environment. Managing networks is a complex and costly process because such diverse environments usually incorporate various operating systems, devices, protocols, and client technologies, making it difficult to troubleshoot and repair malfunctioning hardware and software components. The computer industry is committed to providing methods and technologies to simplify and facilitate network management, and consequently lower TCO of computing environments.

2.2 The WMI solution

The WMI technology and CIM address such management issues by providing a standard, uniform model for representing and accessing management information, to enable developers to easily create

associations between applications, drivers, systems, devices, and network management information (***, 2000).

There are several key features in WMI very valuable in solving the complex management tasks.

- **Remote administration.** Objects managed within WMI are by definition available to applications and scripts both locally and remotely. No additional work is needed to manage remote objects. Connecting to a remote object is similar with accessing a remote database server using a connection string.
- **Query Capability.** WMI treats its managed data much like a relational database and allows for SQL queries to be submitted in order to filter and focus requests for data to only that of interest.
- **Powerful Event Publication and Subscription.** Events can be requested for virtually any change in the managed objects in the system, regardless of whether they support an internal event capability. Event subscribers are able to request notification of very specific events based on their particular interests rather than only being able to get events that were predefined by the original developers. Also, a very flexible architecture allows virtually any user-defined action to be taken upon the receipt of a given event.

These concepts are reflected in diagram presented in Figure 2. The solution developed to manage the distributed system is not linked to a particular machine and it floats above the WMI service. This is the greatest benefit provided by WMI.

3. PERFORMANCE MONITORING, ANALYZING AND EVALUATING

3.1 Defining the problem

Stress testing should always be performed before deploying the application in a production environment. The basic purpose of stress testing a Web-enabled application is to accomplish the following:

- To accurately measure the individual user experience as overall user load is increased on the system
- To determine the maximum capacity of the hardware utilized by the application and thus determine whether a hardware upgrade will be necessary before the deployment of the application in a production environment
- To define the acceptable performance threshold in terms of average or boundary values (for example page response time) for the user of the application
- To ensure that the performance threshold remains at an acceptable level when the estimated maximum concurrent user load is placed on the system

To effectively stress test your data access components and correctly diagnose the results, a method of monitoring and recording operational statistics is of paramount importance. The ideal tool should provide these features:

- *Monitoring* – it should allow the user to select the performance counters he or she is interested in watching. Monitoring the right counters could help locate a problem in the software, even without looking at the code;

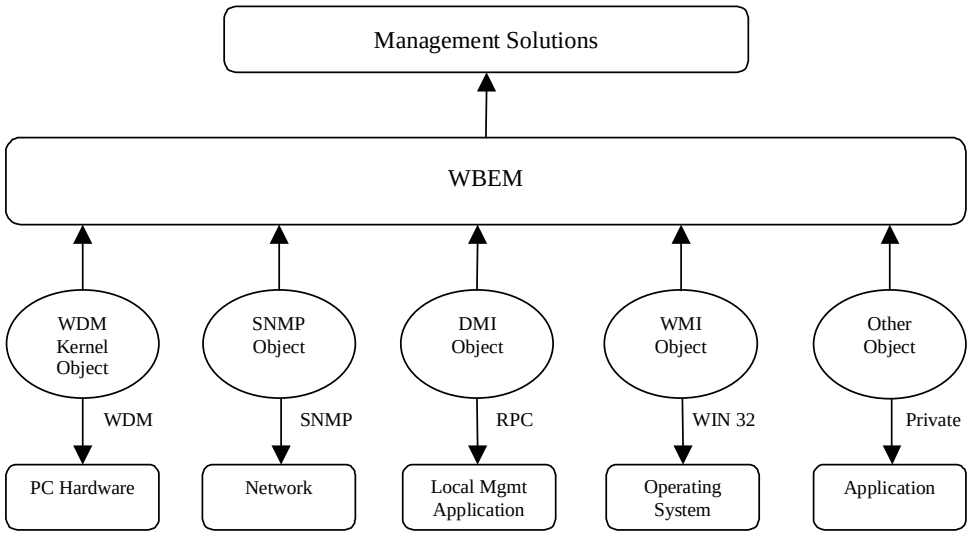

Fig. 2. WMI Architecture.

- *Analyzing* – the user must have a possibility to compare the results from different runs that perhaps used different hardware/software configurations. This is very useful when testing distributed applications, and many scalability and performance issues can be tracked this way;
- *Evaluating* – the amount of data resulted from a performance run could sometimes be really huge, and the tool should provide a smart way to easily detect the problems.

3.2 Dataflow

Inside of a WMI based performance-monitoring application there can be defined two types of data being used:

1. *Performance data*, constituted usually in the form of primitive data types (e.g. int, float, string) and that contains the raw value of a specific performance counter (PCt) for specific WMI Provider occurred in specific conditions
2. *Metadata*, that represents the description of the complex constituted from three structures:

    a. *Data description*: the description of the data type and basic information relative to the provider that generated the data
    b. *Data environment:* the description of the conditions in which the value occurred; here are included not only the information about the data recurrence and WMIP state but also any other system relevant data (e.g. global system state or the state of a different WMIP that interacts with the provider that generated the data)
    c. *Data methods:* actions possible to execute relative to the data itself: the simplest ones are the "Get" and "Set" operations, but we could have here also any other transforming operation that can be executed against the performance counter value

Based on the delimitation made before, the dataflow for a WMI based performance-monitoring application contains actually two different but strongly interdependent dataflow corresponding to the two data type: the performance dataflow and the metadata flow. The main important interactions, as presented in the Figure 3 are corresponding with the main possible actions inside of such a system.

The key element in the structure is the WMI sink: based on the metadata requests from the Data Management Module (DMM), the sink has the responsibility to collect the performance data corresponding to the metadata requested and to send them back to the DMM.

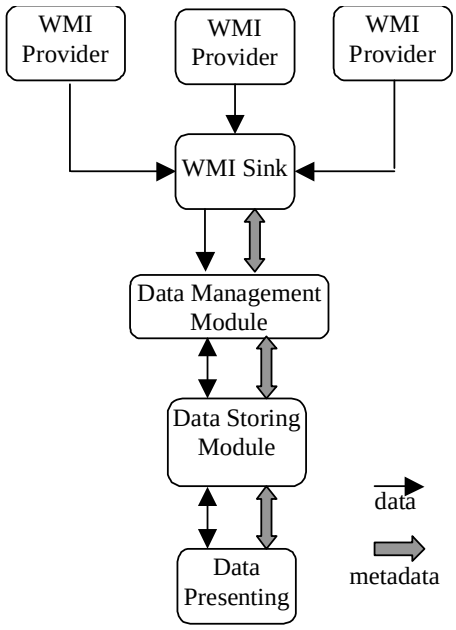

Fig. 3. Typical dataflow for a WMI based performance-monitoring application

This perf data are collected from the corresponding WMIP in the requested conditions based on an event system; for example, if the DMM is interested in collecting the information referring to the processor usage on quad processor system the generic description of the metadata can be like this:

```
<MetaData>
    <WMIProviderLocation>
        EftDev01
    </WMIProviderLocation>
    <WMIProvider>
        Processor
    </WMIProvider>
    <WMIProperty>
        ProcessorTime
    </WMIProperty>
    <Instance>
        <Element> Total </Element>
        <Element> 0 </Element>
        <Element> 2 </Element>
    </Instance>
    <Recurrence>
        <Recurrent> 5 </Recurrent>
    </Recurrence>
    <Method> Get </Method>
    <Method> Set </Method>
</MetaData>
```

So, the WMI Sink will request and, under normal conditions will receive, from the WMIP located on the machine EftDev01 the value of the processor usage on every five seconds for the processors 0 and 2 and for the average value for all the processors

DMM will transform the data received from the WMI Sink (if necessary) and send them to the Data Storing Module (DSM) that represents the area designed to store and retrieve the metadata and

performance data for current process and to store them for future usage. From performance and reliability reasons, this is usually implemented over a SQL Server or an Oracle backend; a MSMQ implementation is also possible.

Finally, the Data Presenting (DP) that is the one responsible with the data formatting and representation to the user; the flow is bi-directional between DP and DSM due to the fact that usually the user could want to save its own statistics and data in DSM for future use.

3.3 Data organizing

Data organizing is one of tasks that can dramatically influence the overall efficiency of the application and, implicitly, its usability in a real world environment with emphasis in the real-time applications as indicated in (Redmond III, 1999).

The structure proposed in the present paper is grouped in 6 main areas (see Figure 4):

1. *Metadata description*: contains structured information about the data, as described in chapter 3.2
2. *Performance data*: the actual data returned from the WMIPs
3. *Data Profile*: collection of relationships between the metadata used during a run
4. *Run information*: run specific data (start and stop data, owner, etc) and the pointers to the respective data profile and stress profile
5. *Stress profile*: collection of data referring the stress conditions and specific load against the targeted computers
6. *Stress data*: data returned by the stress application (Web Application Stress here).

The backend used for supporting this structure was the SQL Server; another thing to mention here is the fact that most of the interactions with the database were integrated in store procedures in order to minimize the data transfer between the application and backend.

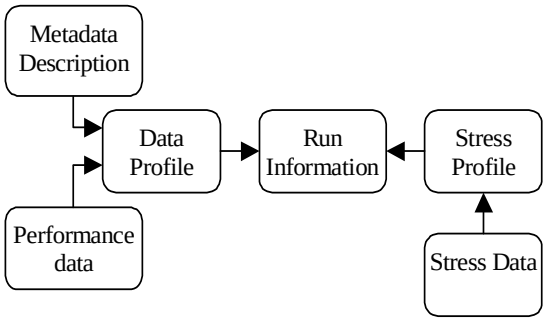

Fig. 4. Data organization (SQL Server backend).

3.4 Architecture design

The performance monitoring application (DNA Watcher) is structured, much like the tested application (Duwamish Books Online), on the WinDNA architecture (Sundblad and Sundblad, 1999). The design of the application followed these guidelines:

- Three-tiered methodology - a three-tiered application is an application whose functionality can be segmented into three *logical* tiers of functionality: presentation services, business services, and data services;
- Independent localization of the components (layers) – the data layer is installed on a remote machine to maximize the reliability and the scalability of the application;
- Each layer must interact only with adjacent layers – it's not permitted, for example, for the user interface to access the database layer. The proper layer verifies this way the correctness of the parameters and data.

The application includes support for a stress tool that simulates a real-world environment. A web stress tool should have the ability to simulate a high number of concurrent connections with sufficient threads to maximize the concurrent connections while throttling the packet sizes sent to the web server. The tool used with DNA Watcher is Microsoft's Web Application Stress Tool which provides all necessary capabilities.

The presentation layer allows the user to select all the entities of the stress test: machines, counters, stress level, run duration etc. The data is routed through the façade layer to the business classes and it's finally stored in the database. When the run starts, the WMI engine connects to WMI service on the machines involved in the test and registers events according to the user selection. The data is collected by the WMI services and stored in real-time during the test by the data services.

4. IMPLEMENTATION ISSUES

One of the main challenges that the application designer should overpass is the selection of the right programming language for the right component; the choice is more difficult taking into account that the WMI support is offered for all the main programming platforms available at this moment (C++, VB, C#, VBScript and JScript). For the implementation of the system described in the current paper it was used a combination of C++, VB and VBScript.

The kernel of a WMI based application is constituted by the events, but only the non-script languages offer support for an event handling mechanism. The application should enlist to the WMI service by interrogating this service with a WQL query. For example, to receive the data described in paragraph 3.2 there should be used a query like:

```
'Get a pointer to the WMI service on EftDev01
'machine
Set wbemServices =GetObject("winmgmts:
{impersonationLevel=impersonate,(security)}
\\EftDev01\PerfMon")

'Build the recurrent WQL query to enlist for the
event
'to the WMI service
WQLquery = "select * from
__InstanceModificationEvent within 5 where
TargetInstance isa Processor and
(TargetInstance.InstanceName = Total or
TargetInstance.InstanceName = 0 or
TargetInstance.InstanceName = 2)"

'Associate the WMIS object with the WQL query
for
'the defined context that contains metadata
Call wbemServices.ExecNotificationQueryAsync
(sinkWMIEvents, WQLquery, , , , WQLContext)
```

The WMIS is the receiving object for the event and its context; when a new event occurs, it's WMI's responsibility to fire a VB event (***, 1998b) to the WMIS. The application can overwrite the default event handler so the incoming data is processed and sent to the data layers.

Taking into account the huge volume of the performance data during a run the decision was not to store any redundant data, but only the data received for a modification in the value of the counter. By not having complete data in the database, a supplementary interpolation level was added to the DP module; while the WMIS is a real time operating module, the DP module is working offline and it can be overloaded without impacting the overall quality of the solution.

The entire event system is synchronized with the stress tool (WAS) in order to acquire data while the targeted application is running under the WAS load. The WAS object model was used in the business layer to achieve this goal. At the end of the stress test the data collected by WAS is linked with DNA Watcher database and the WMIS is destroyed.

As a part of machine management, a C++ COM object was built to register performance counters to the local WMI service and to the database; the module works remotely to follow the design pattern of not having any local components on the targeted machines.

An additional feature was the capability to work in batch mode. This provides the support for the creation of a list of parameterized runs (duration, load, network configuration, etc.); then the tester can create sets of tests targeted to analyze the different capabilities of the system like extensibility or scalability.

VBScript and ASP were selected for the DP module so the data is accessible on the web; the user is allowed to create cross comparisons between different runs, machines or property either in a statistical approach or in a dynamic representation over the duration of the run(s).

5. RESULTS AND CONCLUSIONS

DNAWatcher, the application implemented conforming with the architectural guidelines listed in the present paper, proved the viability of the solution by running with a high load (up to 80 counters/run) on long time runs (up to 72 hours) and using high level of stress threads (up to 1600) without losing the coherence of the data acquired and without significant lowering of the overall distributed system performance (running on both intranet and VPN). The targeted platforms ranged from single processor machines to web farms composed by dual and quad processor computing systems.

The chosen data processing system (acquiring, storing and representing) allowed to discover a large area of insidious problems. The dynamical analysis tool displayed the history of the monitored counter; for an usability example, a counter with a good average for the run, was discovered to have a high fluctuant evolution, with abrupt transitions between extreme values. The abnormal behavior of such counters could be precisely located in time, making it possible to narrow down the problem.

At the same time, the batch mode feature greatly reduces the total cost of ownership (TCO) providing all the benefits of automated testing.

REFERENCES

*** (1998a) WMI Background and Overview, *MSDN*

*** (1998b) *Visual Basic 6.0 Programmer's Guide*, Microsoft Press

*** (1998c) Microsoft Windows Management Instrumentation: Advantages to Developer, *MSDN*

*** (2000) WMI SDK, Microsoft

Coulouris G., J. Dollimore and K. Kinderberg (1999) *Distributed Systems; Concepts and Design*, Addison Wesley

Redmond III, F. E. (1999) Designing and Building Windows DNA Applications, *MSDN*

M. Maston, M. (1999) Managing Windows with WMI, *MSDN*

Sundblad S. and P. Sundblad (1999) *Scalable Visual Basic and MTS Application*, ADB-Arkitectur AB, Uppsala, Sweden