

Designing Manageable Applications

Get a Grip on Your Application: How the use of standard-based technologies greatly enhance the manageability of software applications

With the increasing complexity and intricacy of corporate business environments, it's no wonder that IT groups everywhere find costs tougher to control and services more difficult to assure. The answer is designing and developing software applications so they are readily manageable.

by Justin Murray

Development managers familiar with software process improvements know that the cost of finding and fixing development problems increases as an application reaches use by customers. The same principle is true for the cost of managing applications in deployment. The costs of diagnosing runtime problems and determining their solutions increases with respect to the phases of the application life cycle.

To help reduce costs and deliver high-quality applications, developers can design for better manageability in the application itself. This article introduces several ways to develop better programs by designing and implementing management capabilities into programs and encompassing two levels of support, for the *platform* and the *application* level. The article looks at the various industry-standard technologies that are available to the developer to use.

Application Manageability

It's easy to believe that an application is manageable if it can produce messages in a text log file. Log files can be sufficient for the management of very simple types of application deployments, but this is a primitive management technique and does not afford any significant application control. Java developers, for example, build logging into applications through the use of features such as those

found in the emerging standard, Log4J. Logging messages is the base level of manageability of an application, so consider it worth doing in your application, but make sure the operational staff will understand your messages and not be overwhelmed by them. It is easy for management console products to pick up and filter these messages so that operators can make sense of them. Should two copies, or instances, of the application be required to be run in parallel, then the log file method will need more design attention, such as locking of the file between writes done by the separate instances, for example.

Application manageability is not limited to knowing simply whether the application is alive, or whether the application has produced an error message in its log file. For certain applications, it is important to know how many user transactions the application processed, how long each transaction took, and what percentage of them failed to complete, among other matters.

Sophisticated applications may have queues of incoming requests, some of which are outstanding and some being currently processed. An IT system administrator may need to know the condition of each queue, so that problems can be solved quickly. Should a queue of incoming requests be causing delays to the end user, for example, the administrator may wish to allocate some of those requests to another identical process queue, thus avoiding a problem.

It is clear that the term "manageability" can mean many different things to different communities of users.

A framing definition of "manageability" will help developers judge how much of it to build into any one application. The "level of manageability" that is required by an application determines what technologies to use and how much effort to expend on this aspect of the application design.

Definition: "Manageability" is the ability to exercise administrative and supervisory actions and receive information that is relevant to such actions, on a component.

Manageability is further distinguished by three functional parts:

1. Monitoring – the ability to capture runtime and historical events from a particular component, for reporting and notification
2. Tracking – the ability to observe aspects of a single unit of work or thread of execution across multiple components (e.g. tracking messages from senders to receivers)
3. Control – the ability to alter the runtime behavior of a managed component (e.g. changing the logging level of an application).

Manageability provides a mechanism to ensure that an application is both alive and functioning normally, as well as that of checking an application's performance over time. When a runtime problem occurs, management tools enable an engineer to troubleshoot the problem. The three aspects of manageability shown above are used in subsequent sections of this document to assess various technologies.

Further, manageability covers the monitoring, tracking and control of more than one instance of the same software. For example, applications that are built on J2EE-compliant application servers are frequently replicated on several computers for load balancing and fault tolerance. IT operators need to monitor and control the levels of activity of each one of these instances at all times. They can then decide to re-route requests or take other actions to allay problems and attain service level agreements.

Key Design Decisions in Manageability

The decisions the developer must take in applying manageability functionality to his/her software are:

- What level of information is most useful to the operator of the software?
- What actions can be taken on the software as a result of this information?
- What level of detail (fine grained access to individual objects, or coarser grained access to subsystems, modules or otherwise) is best for application manageability?
- Is the management software to be allowed to interact synchronously with the application code or must all actions be taken “after the fact”?

There are varying levels of sophistication that developers can achieve when designing and implementing manageability into an application. For example, HP OpenView Operations (OVO) can “encapsulate” messages written to log files. This level of manageability may provide sufficient details about the application characteristics, but requires some integration effort. This level qualifies as “monitoring” or “tracking” but not “control”.

Another level of sophistication is to embed monitoring API calls in the application to communicate directly with the management platform. An example of this is the use of the “operations console message” API to write messages from an application to an OpenView management console. This level can provide finer-grained details of application behavior, but requires more development time. The developer must take care with this API not to flood the operations console with too many detailed messages.

Encapsulation methods are useful when an organization cannot afford to extend an application with new interfaces. Embedding calls to a management framework is preferable when more fine-grained control is needed.

The following sections discuss the standards technologies that developers of manageable applications will consider using.

Simple Network Management Protocol (SNMP)

This network and system management protocol is in widespread use today for software and device management. It was invented in the 1970's, and was enhanced to SNMPv2 in the 1980's. The most prevalent network and system management tools that exist today support SNMP. This proves to be one of the reasons why many developers are attracted to working with SNMP to make their applications manageable — many products are available as the management toolkit.

SNMP defines a client-server relationship. The client program (called the network manager) makes virtual connections to a server program (called the SNMP agent) that executes on a remote network entity (often a device, but this could be an application platform also). Agents provide information to the network manager regarding the remote entity's status. An SNMP agent is therefore required on each machine on which an application is running, to monitor that application or component.

The database of management information that is used and controlled by the SNMP agent is referred to as the SNMP Management Information Base (MIB), and is a standard set of statistical and control values. We can think of this as a database of records that are composed of name-value pairs, each of which can describe a property of a managed object.

SNMP traps are events that are generated when something of interest happens in a managed object that is described in a MIB. To handle a trap, an application sends a request to the management station to have the event forwarded. The programmer does not need to write code to poll for traps — the management agent just sends the traps to the network management station(s) as instructed by the program. Management agents may send repetitive traps. Custom software on the network management station is frequently used to filter these trap messages using certain criteria and deal with the most important ones.

An application cannot, however, rely on SNMP traps alone. For example, if an agent dies, it cannot tell the manager that it is dead. Therefore it is a good idea to poll agents periodically just to see if they are alive and well. This imposes some extra work on the developer. The SNMP trap technology has two significant issues that the developer should be aware of:

1. The unreliability of the transport (UDP) can cause messages to be lost
2. SNMP traps will continue to be sent repeatedly until they are acknowledged by the receiving party—perhaps causing flooding to happen.

There are technologies available on the market that fall under the general category of “extensible SNMP agents”. These usually include a development kit that automates sub-agent development. Based on a Master Agent/Subagent architecture, they allow sub-agents to be loaded and unloaded dynamically at run-time. These technologies can ease the task of developing an SNMP-aware system or application.

SNMP is useful framework for managing applications. As its acronym suggests, the protocol is simple enough that adding SNMP to an application is a reasonable endeavor.

Common Information Model (CIM) and Web-Based Enterprise Management (WBEM)

The Distributed Management Task Force (DMTF) is a standards body of senior representatives from Cisco, Novell, Sun, IBM, HP and several other companies. The DMTF publishes several standards that together provide interoperable management for systems, devices, networks and software applications. Among these standards are the Desktop Management Interface (DMI), the Alert Standard Format (ASF), System Management BIOS (SYMBIOS), Directory Enabled Networking (DEN), WBEM and its associated common information model, CIM. The DMTF has developed a core set of standards for enterprise management that make up WBEM. WBEM includes a data model, called the CIM standard. WBEM is the implementation of the CIM management model specification.

The CIM specification details a language and methodology for describing management data. The CIM schema includes models for systems, applications, networks and devices. The CIM schema enables applications from developers on different platforms to describe management data in a standard format. It can then be shared between varieties of management applications.

The xmlCIM Encoding Specification defines XML elements, in the form of a Document Type Definition (DTD) that can be used to represent CIM classes and instances. The “CIM Operations over HTTP” specification defines a mapping of CIM operations onto HTTP, allowing implementations of CIM to interoperate in a standard manner. We can regard CIM as the specification for which WBEM provides the implementation. WBEM is built into the Microsoft Windows operating system.

WBEM goes beyond similar network management standards such as SNMP and DMI, and defines the following specifications:

- a rich model of manageable entities featuring inheritance and associations (the Common Information Model, or CIM)
- an extensible set of operations that can be performed on these objects (CIM Operations)
- a protocol to encode the objects and operations for communication over a network (xmlCIM)

To use a WBEM/CIM approach, the developer is required to first model the required managed objects in an object-oriented style. Then the developer maps his/her managed application objects into the common information model, or a derived extension of it.

There are many other details and initiatives that the DMTF is pursuing. These standards activities will help put meaningful structure to the use of management technologies in specific domains. Space does not permit a full treatment of this subject in this article, but readers are encouraged to learn more at the DMTF website (www.DMTF.org).

The Java Management Extensions (JMX) Standard

The Java Management Extensions have become the accepted standard in the industry for managing Java applications. The specification for JMX within the Java Community Process (JCP) is a Java Submission Request (JSR) (JSR3). A separate specification, JSR77, which is part of the J2EE 1.4 specification, is a description of the "model" of objects that each application server must expose through JMX. The JMX specification provides for the construction of manageability of Java applications in a standard way.

The JMX environment is composed of three levels of software:

- the instrumentation level
- the agent level
- the distributed services level

These levels are shown in the Figure 1 (the distributed services layer is the topmost one).

The Instrumentation level contains managed bean objects, or MBeans. These are Java objects that conform to either:

- a simple style of object construction in the Java world called JavaBeans (for simple MBeans)
- a JMX standard called "Dynamic MBean" for more flexible management
- the JMX "Model MBeans", which are an extension to Dynamic MBeans that provide more generic templates for management instrumentation

Conforming to the JavaBeans format is straightforward. It requires that an object is serializable and has a null constructor. It is common practice to implement "getter" and "setter" methods on all that object's attributes that must be visible to external management tools. Creating MBeans in this style means that once the MBeans are registered with the JMX server, called the MBean server, those MBeans can be reached for monitoring and control purposes by the tools at the distributed- services layer at the top of the diagram.

JMX requires the use of an MBean server shown at the Agent Level in the diagram. An application server process supports this MBean server functionality. The MBean server handles the management messages that are flowing to and from objects that have been previously registered with it as MBeans. Certain properties of the object that conforms to the MBean interface can then be viewed by management tools and, provided the developer of the MBean has allowed it, the behavior of the object can be changed, using functionality of management consoles. This is done through

the connectors and other adapters that allow a management console to extract data from a JMX server, not through direct manipulation of the MBeans themselves.

The MBean mechanism is powerful in the sense that it provides an industry standard method for "wrapping" a business object, with another MBean object, the latter being dedicated to manageability. The alternative approach is to allow the business object to conform to MBean interfaces itself.

A JMX-compatible Smart Plug-in (SPI) for HP OpenView Operations

HP enhances application manageability by providing a specific Smart Plug-in (SPI) module that integrates the management of an application environment with the management of the platforms on which it depends (operating system, JVM, application server). The SPI integrates several sources of information into one management tool. It:

- Gathers data from any supplied MBeans, such as those supplied by an application server
- Performs calculations on that data to be more meaningful to the end operator
- Brings filtered application logging messages to bear on the management task at the same time.

This provides a highly desirable management integration, as many IT operations departments need to monitor the health of their computers, networks, application servers and any applications that depend on them from the same console, using the same, familiar tools.

This HP SPI makes use of the JMX MBean server supplied by the application server and derives some of its information from the MBeans registered with that server. Developers can use the SPI in three ways:

- Existing information (contained in existing MBeans) may be used to provide information through existing OpenView Operations-supplied metrics. There are 55 such metrics available for use by the IT operator. Examples of such metrics are the number and percentage of available JDBC connections.
- Existing information may be used to define new user-defined metrics (UDMs).
- New, custom-built MBeans may be designed by the application developer to provide new information through new metrics. This last approach applies in those situations where business-specific metrics require action by operations personnel. An example would be the number of customers who did or did not purchase items at a website in the past day.

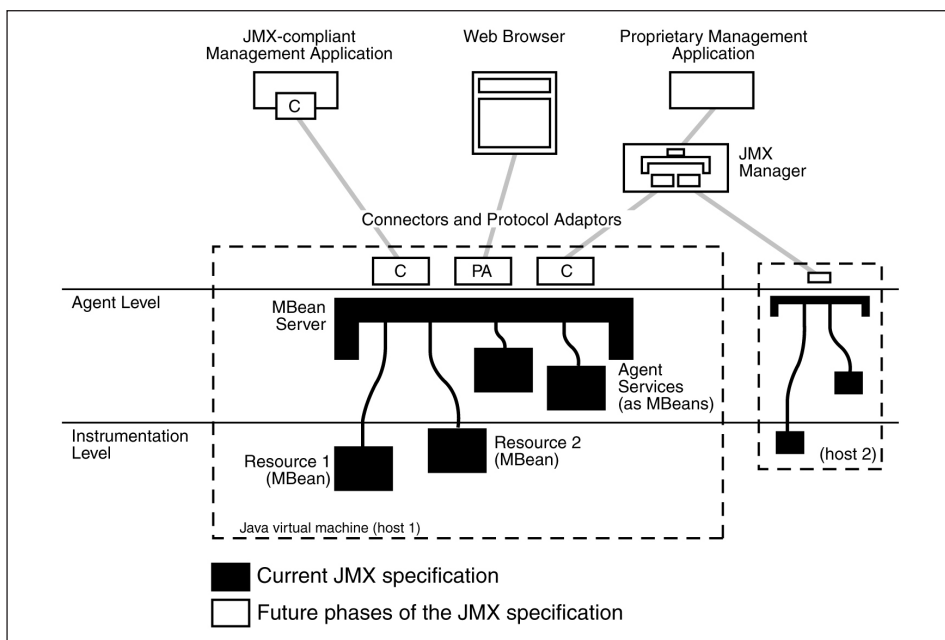


Figure 1. A model of the JMX specification.

The application developer can, if necessary, add his/her own metrics using the features of the OVO console. By defining one's own metrics, a developer helps the operator to monitor the applications by allowing new combinations of data to be derived from existing sources. We see therefore that this tool can monitor and control both the application server platform and with some user coding, the application itself.

Writing Custom Application-Specific MBeans

The JMX capabilities leave the software designer with the problem of deciding which application objects are in need of management visibility and which are not. Should all business objects be capable of being manipulated through their associated MBean features, or just a chosen few?

The software designer has the option of mapping one MBean to one or to many business objects, depending on the need for management of them. Each business object can alternatively be constructed to inherit the MBean capabilities, but this will mean that each such object will need to register with the MBean server. This can lead to scalability issues.

If one business object depends completely on another to get the work done (i.e., they are in a "uses" relationship), then there is an argument for managing them using one MBean. Correspondingly, if objects are entirely independent of each other in the business context, the designer can choose to have a management view of each one, using separate MBeans or MBean interfaces for those objects.

It is unlikely that all objects in any system will need a dedicated MBean of their own. The software designer chooses the Java object or EJB to visualize and control using a management tool. Finally, the developer will register the new MBean with the appropriate MBean server process.

From that point onward, the MBean is manageable using JMX-aware tools. The associated business object is manageable to the degree that its guardian MBean or interface determines its behavior. As an example, the HP OpenView Smart Plug-In technology provides management of the J2EE application server's internal objects, without the application developer having to do extra coding work. Using the OpenView SPI to extend the metrics it supplies to the operator gives you a powerful tool for enhancing the manageability of your application. There are more details available on the customization of MBeans in the HP OpenView developer documentation.

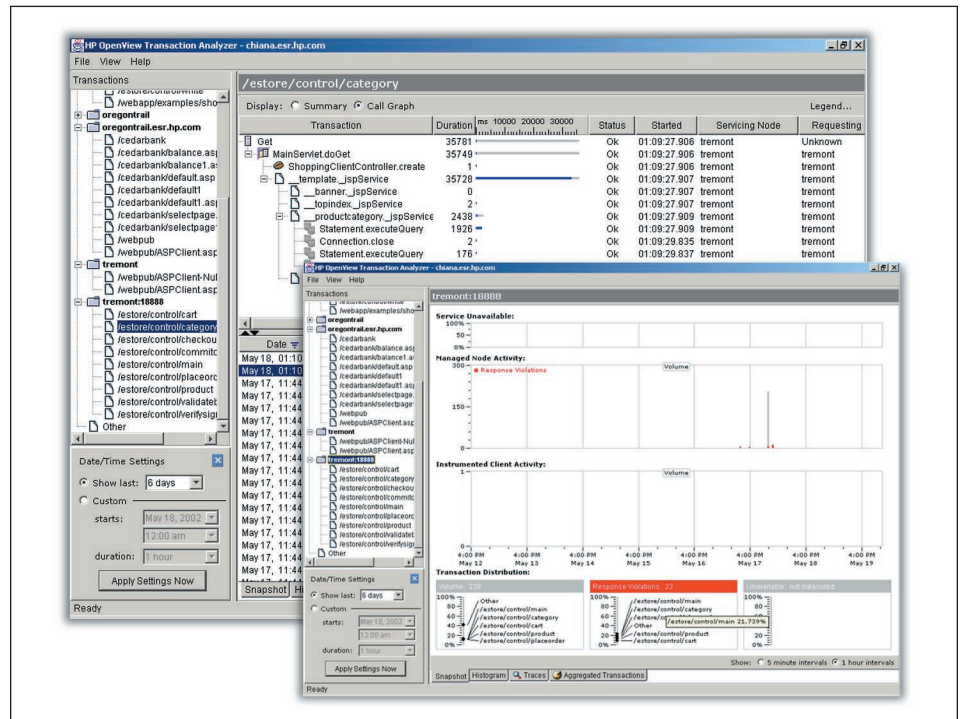


Figure 2. OpenView Transaction Analyzer screens. Highlighted in red is the key culprit for performance concerns within this particular application. This tool fulfills the final requirement for the application developer in building manageability into an application; that of tracking the paths through the code.

Tracking in a J2EE Application with OpenView Transaction Analyzer

Many developers are concerned with finding the bottlenecks in their applications that inhibit better performance. This is not an easy task, due to the distributed nature of these applications. One mechanism to instrument your application is to embed calls to the Application Response Measurement (ARM) API in an application. This API is designed to allow for detailed accounting of CPU time spent in sections of the application. Use of this API can be intrusive on the application design, however, and require attention from the developer in areas where he/she is not an expert.

There are now tools on the market, such as the OpenView Transaction Analyzer (OVTA), that make use internally of ARM's accurate measurement and logging techniques without the need for explicit calls to the ARM API and without application code changes. These tools exploit purpose-built Java class loaders and byte-code manipulation to intercept JSP, EJB, and servlet method entry and exit points. The ARM API is used at these points in the instrumented code and is therefore completely hidden from the application developer, yet the degree of timing accuracy provided by ARM is maintained.

This proves to be a very powerful method of analyzing the full J2EE application at development time, or even later at deployment time, for

performance bottlenecks. The application developer has to make no changes to code to carry this process out. Figure 4 gives a snapshot of the types of screen that HP's OVTA product produces to aid the developer in performance analysis. It can quickly trace down to the point of the problem in a complex application.

Summary

To reduce IT operations costs and deliver high-quality applications, developers must design their applications for manageability. Applications supporting management capabilities will perform better and production issues can be diagnosed and cured more quickly.

Varying levels of manageability are available through the SNMP and JMX standards and through integration with management tools like OpenView Operations

Armed with this suite of management capabilities, developers can make their applications better suited for deployment. In the long run, this helps both IT operations staff and development organizations.

About the Author

Justin Murray is a technical consultant at HP. He has taught and delivered consulting to HP customers and staff on a variety of subjects including optimizing performance of Java and web-services applications.