Java Management Extensions for application management

by H. Kreger

Modern enterprise systems are composed of both centralized and distributed applications. Many of these applications are business-critical, creating the need for their control and management by existing management systems. A single suite of uniform instrumentation for manageability is needed to make this cost-effective. The Java™ Management Extensions Agent and Instrumentation Specification, v 1.0, describes an isolation layer between an information technology resource and an arbitrary (enterprise-specific) set of management interfaces and systems. It includes a simple, yet sophisticated and extensible management agent that can accommodate communication with private or acquired enterprise management systems. The application programming interface is simple enough that manageability can be achieved in three to five lines of code. Yet, it is flexible enough that complex, distributed applications can be managed, allowing management of Java technologies as well as management through Java technologies. This paper includes an overview of application management issues and technologies. The JMX technology and application program interfaces are discussed in depth using examples pertinent to today's application developer.

Modern enterprise systems are composed of both centralized and distributed applications. The introduction of the Internet and intranets has brought about a new class of applications that connect the end user to existing, traditional, centralized applications. A new class of autonomous, Web-based applications are also being developed and rapidly deployed in the new business environment. Service-based architectures are emerging in which management of applications becomes even more difficult as information technology (IT) resources appear, move, and disappear across the network. These new applications have become business-critical, creating the need for their control and management by existing external management systems. A single suite of uniform instrumentation for manageability is needed to make this cost-effective.

The Java** Management Extensions (JMX**) Agent and Instrumentation Specification, v 1.0,[1] describes an isolation layer between an IT resource and an arbitrary (enterprise-specific) set of management interfaces and systems. The JMX package defines extensions of the language (Java Optional Package for J2SE**[2]) that will allow any Java technology-based resource to be inherently manageable, along with a set of services for managing these objects. It includes a simple, yet sophisticated and extensible management agent that can accommodate communication with private or acquired enterprise management systems. The application programming interface (API) is simple enough that manageability can be achieved in three to five lines of code. Yet, it is flexible enough that complex, distributed applications can be managed, allowing management of Java technologies as well as management through Java technologies.

This paper is divided into four main sections: an overview of application management, an overview of JMX, a few scenarios with programming examples, and a

0018-8670/01/$5.00 © 2001 IBM

detailed review of the JMX components and interfaces. The overview of application management is intended to provide some understanding of the current state of management issues and technologies as well as some common terminology. It covers application management issues and trends, a management technology survey, management system components, and application management approaches. The history and technology survey helps explain the business case and advantages of using JMX technology for application management, while positioning it relative to the application management industry.

The application management problem

The primary challenge in managing applications is diversity. Applications today vary widely in purpose, size, architecture, and criticality. There is very little that is common across all application types, and application architecture trends are increasing this diversity.

Modern enterprise systems are composed of both centralized and distributed applications. Centralized applications, such as payroll and accounting applications, are backed by a database on a high-end server and are usually accessed by a limited set of users—those in the financial department, for example. Distributed applications, such as mail systems, usually require groups of smaller server systems to be running at all times and are accessed throughout the enterprise. These applications represent the "traditional" application in the current enterprise environment.

The introduction of the Internet and intranets has precipitated a new class of applications that connect large numbers of end users to existing, traditional, centralized applications. The new application facilities range from Web-accessible corporate personnel directories to Web-based order-tracking systems that benefit customers and reduce order management costs. These applications make it easier to access corporate information inside traditional applications and reduce the number of personal contacts necessary.

The next generation of autonomous, Web-based applications is rapidly being developed and deployed in the new business environment. These applications embody e-commerce in the form of catalogs, shopping, markets, and auctions. The movement of the supply chain to the Internet will drive the next set of critical, distributed business-based applications. These applications will move the heart of the business—buying supplies and selling products—to the Internet.

Service-based architectures are emerging in which management of applications becomes even more difficult as IT resources appear, move, and disappear across the network. UDDI (universal description, discovery, and integration) is a standard describing how and where services are advertised. Sun Microsystem's Jini**,[3] the Universal Plug and Play (UPnP) initiative,[4] Hewlett-Packard's e-speak,[5] and IBM's Web Services Toolkit[6] are all service-oriented architectures for business environments.

These new classes of applications are executing across multiple hosts, operating systems, and corporations. They incorporate existing traditional and emerging application models. They are no longer just client/server applications; they are now client-middleware-server applications. They are business-critical and need to be uniformly controlled and managed, by a business's existing management systems, with the same diligence as traditional applications have been managed. JMX is flexible and extensible enough to be used in all of these diverse applications.

A short history of management technologies. A brief review of management technologies and their histories illustrates how critical it is that a management architecture for Java technology allow applications to be independent of the management technology choice. Businesses invest in enterprise management systems for network and systems management for several reasons:

- There are too many IT resources to be tracked with internally developed tools.
- The IT resources are too distributed to control from a single systems console.
- The IT resources are of diverse types, requiring multiple consoles in the operations center.
- The IT resources need highly available IT infrastructure.

Historically, mainframes such as IBM's System/390*[7] and Amdahl Corporation's Millennium** 2000[8] have had tightly coupled management systems because of the business demand for high availability. In very large businesses, the number of mainframes and dispersed computing centers to be managed placed high demands on operations staff, and new enterprise systems management products were developed by companies like IBM, Candle,[9] and Computer Associ-

ates.[10] As enormous SNA[11] and TCP/IP[12] networks began connecting these computing centers to each other and to distant users, network management products became critical IT investments. Tivoli's NetView** for OS/390*[13] and Hewlett-Packard's OpenView[14] were some of the first successful enterprise network management systems. As many business applications moved into distributed server environments, the demand for enterprise systems managers increased. Business systems management now required external management—the servers were dissimilar, numerous, distributed, and had to be highly available. Tivoli Systems,[15] Computer Associates, and BMC Software[16] are a few of the companies that have supplied management products to meet these challenges.

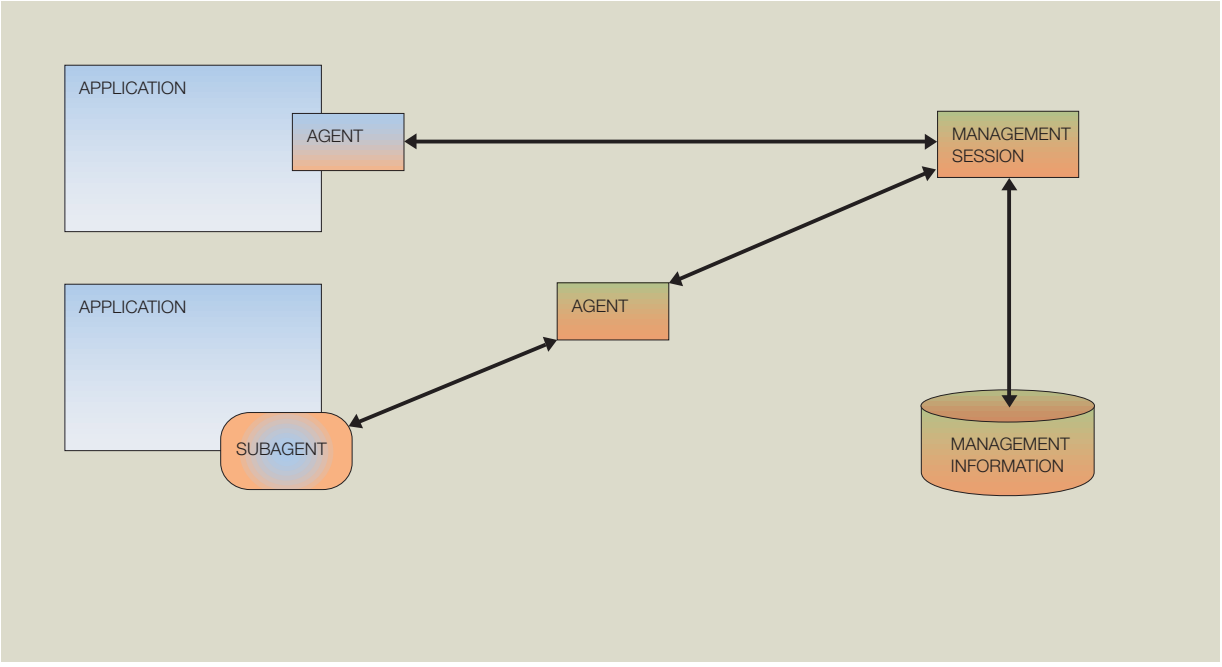
These management systems have been implemented using different protocols and technologies. This is illustrated by a survey of management technologies in use today:

- SNMP is a *de facto* standard for device and network management. However, it has not been widely deployed for application management. It was developed as a temporary solution for management until CMIP was completed. Hewlett-Packard's OpenView** and Tivoli's NetView are popular SNMP-based network management products today.
- CMIP (Common Management Information Protocol)[17] was developed by ISO (International Organization for Standardization) as a management standard. It did not displace SNMP (Simple Network Management Protocol)[18] as expected, but has been used by the telecommunications network management market. Sun Microsystems, Evidian (formerly Bullsoft), and Hewlett-Packard have network management products that support CMIP.
- The DMTF (Distributed Management Task Force)[19] CIM/WBEM (Common Information Model/Web-Based Enterprise Management)[20] standard defines a management information model and an XML (Extensible Markup Language) -based interface to access the information. While the device and system models are fairly complete, the application model is still a work in progress. Although this is still an emerging technology, Microsoft's SMS[21] and Sun's Solstice** provide support and managers for it. SNIA (Storage Networking Industry Association) provides an open source implementation.[22]
- The Open Group's Enterprise Management Program has published several standards that focus on manageability. Recent significant additions include Application Instrumentation and Control (AIC) and Application Response Measurement (ARM) standards. AIC is a C language API (application programming interface) for exposing application metrics and thresholds. ARM is a method for capturing the amount of time it takes to perform units of work inside applications. A language- and platform-independent manageability agent is currently being defined and prototyped. This agent will be interoperable with WBEM.
- Many operating systems, such as IBM's System/390 and Microsoft's Windows** 2000, have their own proprietary management systems and infrastructure.
- The dominant enterprise management system providers, Tivoli, Computer Associates, and BMC Software, use their own proprietary technology for distributed management infrastructure, including manager-to-agent communications. These enterprise management system vendors compete not only on the management system infrastructure and capabilities, but also on the set of products to be managed and the services for installation and customization.

While SNMP dominates the device and network management market, there is no dominant management technology in the application and e-business management markets. Tivoli, Computer Associates, Microsoft, and Hewlett-Packard are all competing for these markets. Most developers of popular cross-platform applications choose a single management technology to accommodate or attract a portion of their target market. In some cases, developers implement multiple management technologies in order to provide more complete coverage of their potential markets, resulting in a significant development cost. With either approach, the potential return on investment does not motivate developers to instrument appropriately for manageability. As a result, more often than not developers choose to "roll their own" application-specific management systems, rather than implement for any external management technology. A single suite of uniform instrumentation for manageability, like JMX, allows cost-effective development of new manageable applications.

Management system components and roles. To understand the set of responsibilities that JMX is intended to fulfill, it is necessary to understand the basic architecture used by most management systems. Despite the wide variety of management technologies and products, most management system infrastructures fall into an architecture pattern referred to as Manager-Agent. There are generally four ba-

Figure 1 Typical management system components



sic components to this architecture: managed resource (application), agent, subagent, and management system (see Figure 1).

Application. The application, or managed resource, is any computer program engaged in business-oriented work. These applications can be running on servers, clients, and even devices. When looking at types of Java technology applications we must be sure to include the management needs of stand-alone applications, client/server applications, and Web applications. Web applications include server-side (servlets, Enterprise JavaBeans**, etc.) as well as client-side (browsers, applets, etc.) elements.

The managed application is responsible for exposing appropriate data for use by a management system, responding to requests from the management agent, recognizing internal errors, and posting events to the management system. Requests from agents may include getting data, changing configurations, or executing operations. The application determines which requests it will support and to what degree these requests are supported. The application communicates with or contains a management agent or subagent.

Agent. The management agent communicates with the management system and the application. It is usually running on the same host or process as the application it is managing. SNMP, CMIP, and WBEM all define standard agent architectures. The enterprise management systems, such as those from Tivoli and Computer Associates, have proprietary agent infrastructures.

The agent is responsible for sending events to the management system, relaying data and command requests from the management system to the application, gathering responses, and returning them to the requester. Some architectures also include midlevel managers between agents and the management system to provide a more scalable solution. Midlevel managers aggregate and filter the information from a set or "domain" of agents, then forward pertinent information on to the management system. This reduces the amount of incoming messages the management system must handle. The agent may be running in the application's address space or may be external to the application, communicating with it via an API or a "subagent" interface. Agents with subagent interfaces can support many applications within the same environment; agents within an ap-

plication can support only that instance of the application.

Subagent. The subagent communicates between the application and the agent and is typically tightly coupled with the application. It is generally delivered as part of the application and runs in the same process. SNMP defines and uses subagents as part of its architecture.

The subagent is responsible for responding to agent requests supported by the application (commands, data requests, and data updates). It registers with the agent when it is initialized, accesses and returns management data to the agent, and sends events from the application to the agent. The communications protocol between the subagent and agent is usually proprietary. For SNMP agents, DPI and SMUX are both widely used subagent protocols.[23] AgentX[24] is being defined by the IETF[25] as a standard subagent protocol for SNMP.

Management system. The management system communicates with management agents that support the same management protocol. It may be a sophisticated enterprise manager, like those available from Tivoli or Computer Associates that manage all the critical applications. It may be a domain-specific management system designed to fully manage a particular application, such as IBM's WebSphere* administration console.[26] Or, it may be an application system-specific management system designed to manage a business system, for example banking or accounting systems.

The management system is responsible for providing the infrastructure and user interfaces to manage applications. It receives events from agents, displays them to or notifies operators, and takes any defined automated actions. The management system can initiate a command or data request, send commands to agents to control the application, and request data from the agent (which requests it from the application) for polling, monitoring, trending, or problem determination. The management system may also do the discovery of enterprise devices and applications in order to reduce management system configuration requirements and produce accurate topology and inventory information. The management approach (i.e., activities) can be customized for an application type or instance. This includes defining the components of the application, the commands that are supported, the statistics that are available for monitoring, the values that are indications of a problem, and reactive as well as operational policies. With more sophisticated management applications, service-level agreements and business system contexts can be defined as well.

The JMX agent, or MBean server (discussed in detail later) satisfies the definition and requirements of a management agent or subagent. The JMX instrumentation API provides the interface used by the managed application to communicate with the management agent.

When to manage the application. In order to use JMX in an application, the developer needs to understand and identify its management requirements. It is important to remember that not all applications require a management system. Most need an application-specific manager to allow users or administrators to configure the application, ensure it is working properly, and perhaps even use the application. Some applications need to be managed by an external enterprise management system. Determining how complex the application-specific manager is and how well it integrates with enterprise management systems involves a number of factors:

- *Complexity.* Complex applications typically require more extensive and comprehensive management support.
- *Scale.* Large-scale applications (those with many installations) generally require distributed management support. They are also excellent candidates for integrating with enterprise management systems.
- *Criticality.* Mission-critical applications, even if only one instance exists, or applications that are an integral part of a business-critical system (databases, routers, and name servers), should be supported by the specific management system as well as the enterprise management system.
- *Policy.* Corporate policies may dictate the degree of management support for an application. If the application is being developed in an organization that has invested in an enterprise management system, there may be business policies that govern which applications are to be supported and how extensive that support must be.
- *Target market.* Vendors of applications must consider the needs and expectations of the target market. If a large percentage of the target market uses a particular enterprise management system, then the application should be supported by the same system. If the target market expects the application to be supported by one or more enterprise

managers, then the vendor will want to satisfy those expectations. Customers expect to purchase applications that will be "good citizens" within their existing enterprise management infrastructure.

Vendors of applications with large, diverse target markets may be forced to support multiple management systems on multiple platforms. This can lead to significant, perhaps insupportable, development costs. Developing and maintaining support for any one of the standard or proprietary management technologies requires a steep learning curve and is a significant development effort. JMX mitigates the investment in management during application development by allowing developers to concentrate on supporting one management technology. Multiple enterprise management systems can interact with the JMX agent in order to manage the application.

The application life cycle. Understanding the management issues for an application across its life cycle puts JMX in perspective. JMX is not appropriate for all aspects of application management. According to the DMTF CIM application model,[27] the application life cycle can be segmented into four activities: deploying, installing and configuring, starting, and monitoring and operating an executing application.

Initial application deployment, installation, and configuration solutions are available from several vendors, including Tivoli, Marimba,[28] and Computer Associates. These solutions generally involve moving the necessary files to a set of target systems, running setup programs, and customizing initial configuration files. They can be defined for an application without direct cooperation or interaction with the application's execution environment. The application developer usually does not have to consider the needs of these management systems during development.

Starting applications, followed by monitoring and operating applications during execution, requires a different management infrastructure because the management system must interact directly with the application. The application developer must supply commands and APIs for operations, including starting and stopping. The developer must also define, maintain, and expose configuration and metric information through logs, events, commands, or APIs for the management system to monitor and understand.

JMX provides the infrastructure for the application developer to use to expose this management information. It also provides the APIs for management systems to gain access to the information and invoke the operations.

Application management instrumentation options. In order for any stage of the application's life cycle to be managed by a management system, its management capabilities and requirements must be defined to the management system. This combination of code and definition can be referred to as "management instrumentation." There are two primary types of management instrumentation: external and internal.

External instrumentation. External instrumentation is defined and executed outside the application. It consists of application-specific customization of the management system so that it can effectively manage the application. The management system's external instrumentation includes special files, programs, or utilities that may be necessary for it to control the application or access application data. These files must define the directories, files, libraries, executable modules, installation processes, configurations, events, APIs, and policies that make up the application. The life-cycle stages of deploying, installing, configuring, and starting are satisfied by this type of instrumentation.

Tivoli's NetView uses AMS (Application Management Specification)[29] files that are generically defined as an application's definition file. Some application writers and management system vendors provide tools for creation of an application's definition files. For SNMP, the MIB (management information base) file represents this type of instrumentation. For CIM/WBEM, the instrumentation is defined as a schema in a MOF (managed object format) file.[30]

Internal instrumentation. Internal instrumentation is developed by the application developer and executed as part of the application. It is specifically engineered to meet management system requirements and includes a command tool or a management agent that can perform operational support and obtain the application status. The life-cycle stage of monitoring and operating an executing application may require this basic instrumentation. Extensive internal instrumentation may be required in order to communicate with a single management system.

Monitoring and operating an executing application requires that the application support instrumentation for several different types of management data and functions. For example, a SNMP subagent must be developed for each application that supports a defined SNMP MIB. Likewise, with WBEM, a CIM provider may need to be developed to support the management schema for an application. There is currently no standard instrumentation data or API approach; however, a management model defining the data for managing executing applications is currently being developed in the DMTF.

A combination of external and internal instrumentation methodologies is optimal. A single common application management API simplifies and minimizes the internal instrumentation required. This is the essence of the requirement for a common management system application-level API, as defined by the JMX API set, described later. JMX is designed to satisfy the requirements of managing the executing application from the application developer's point of view as well as the management vendor's point of view. JMX, an internal instrumentation API, is used to expose and access the execution time management information that is necessary for an application to be manageable by an arbitrary management system. Tools can be used to generate the external instrumentation for a specific management system from the information available through the JMX APIs.

Managing an executing application. Management systems need to access management data about the application, monitor the application, and operate an application. Different types of internal instrumentation are necessary to support these management requirements. By examining each of these needs from the management system's point of view we can identify the information required to manage executing applications. JMX is capable of supporting all of these application management aspects.

Access to application management data. The information about the application that a management system uses is called "management data." The management system must be able to determine the application identity, how it will run, and what application statistics are required in order to monitor the application. This information generally falls into one of three categories:

- *Identification data.* Identification data uniquely identify the current instance of the application. These data are used by management systems with discovery, inventory (audit), and enterprise configuration components. If the management system cannot find identification data for the application, then the application must be manually defined to these components.
- *Configuration data.* Configuration data may include the complete configuration or a subset. Read access is useful for problem determination and dynamic management policy. It gives the management system real-time management guidance about threshold data, logs and components to be managed, and so on. If the configuration data can be set by the management system, then an update to these data may cause the application to dynamically update and abide by the new configuration values.
- *Statistical data.* Statistical data generally represent the current state of the application. Most management systems monitor application statistics to determine application "health." Depending on the quality and quantity of statistical data available from the application, management systems may also be able to use these data for load and resource balancing, tuning, trend analysis, capacity forecasting, billing, and understanding application usage. Service-level agreements and the systems that report and enforce them use statistical data to define the expectations, thresholds, and responses for an application.

Monitoring the application. Management systems monitor applications on a regular basis to make sure the application is functional, catch problems before they become fatal, and gather statistics for analysis. The management data just discussed can be used to track the following:

- *Availability.* The management system reports when the application components are not available or not responding. Application availability is affected by the ability of users to traverse the network to the application and marshal sufficient system(s) resources to run the application. Availability can be determined by monitoring application components, statistical data, and events. Alternatively, the management system can execute an application-provided "test" command to determine application availability.
- *Performance and health status.* The management system reports when an application is not performing within defined limits. It determines this by monitoring application statistics or by running and evaluating the results of a test command that will exercise the critical application functions that must

perform well. Management systems can also test for thresholds, draw graphs, and report trends, using historical statistical data. The results of these activities indicate application health and status. The management system can be sensitized to practical application run-time health characteristics from the specifications provided in the application's definition file. A sophisticated management system may monitor application statistics to establish trend lines to guide tuning for optimal operational results over time.

- *Events.* The management system asynchronously receives or "pulls" events from the application and then reacts to the events according to policy. These events may indicate that there is a current or imminent problem to which the management system must react. Reactions include applying recovery or tuning policy, notifying operators, logging, or filtering. Certain events indicate that the application is healthy; these are usually called "heartbeats." When these events are not received as expected, the management system will react.

Operating the application. The management system needs to control (i.e., start, stop, etc.) the application. It must be able to invoke management commands that control, locally or remotely, all components of the application in response to events, schedules, and user requests. The management system invokes commands from scripts, programs, remote connections, and command prompts, as well as those initiated by management agents. The application's user interface may use these commands for operator-initiated intervention as well as for testing.

The application uses JMX APIs to expose its management data, operations, and events. Tools can be used to generate the external instrumentation with the information available through the JMX APIs, rather than require every application to define its own. The external instrumentation describes the application to the management system. The JMX APIs can be used by the management system to access the management data, invoke the operations, and receive the events from a JMX-enabled application.

In summary, JMX does not address the definition or process requirements of application deployment, installation, or initial configuration. JMX does address the infrastructure required to monitor and operate the application. JMX provides a common set of metadata and APIs that can be used to implement internal instrumentation, generate external instrumentation, and access the management information. This helps automate and minimize the amount of system- or technology-specific definition that would have to be done to integrate the application's management needs with an arbitrary management system.

Java Management Extensions

JMX provides accessible management data while simultaneously shielding the application from management protocols. JMX isolates the application management data from the management system. The API allows applications to surface data, operations, and events to application-specific or enterprise management systems. It allows management systems to support JMX-enabled applications generically with little or no customer integration (i.e., manual generation of external instrumentation). The technical review of the JMX technology introduces its organization into instrumentation and agent levels for specification and compliance along with the components of each level. A few scenarios illustrate how these JMX components could be used in a typical e-business application. This is followed by an in-depth discussion of how the JMX components work and their APIs.

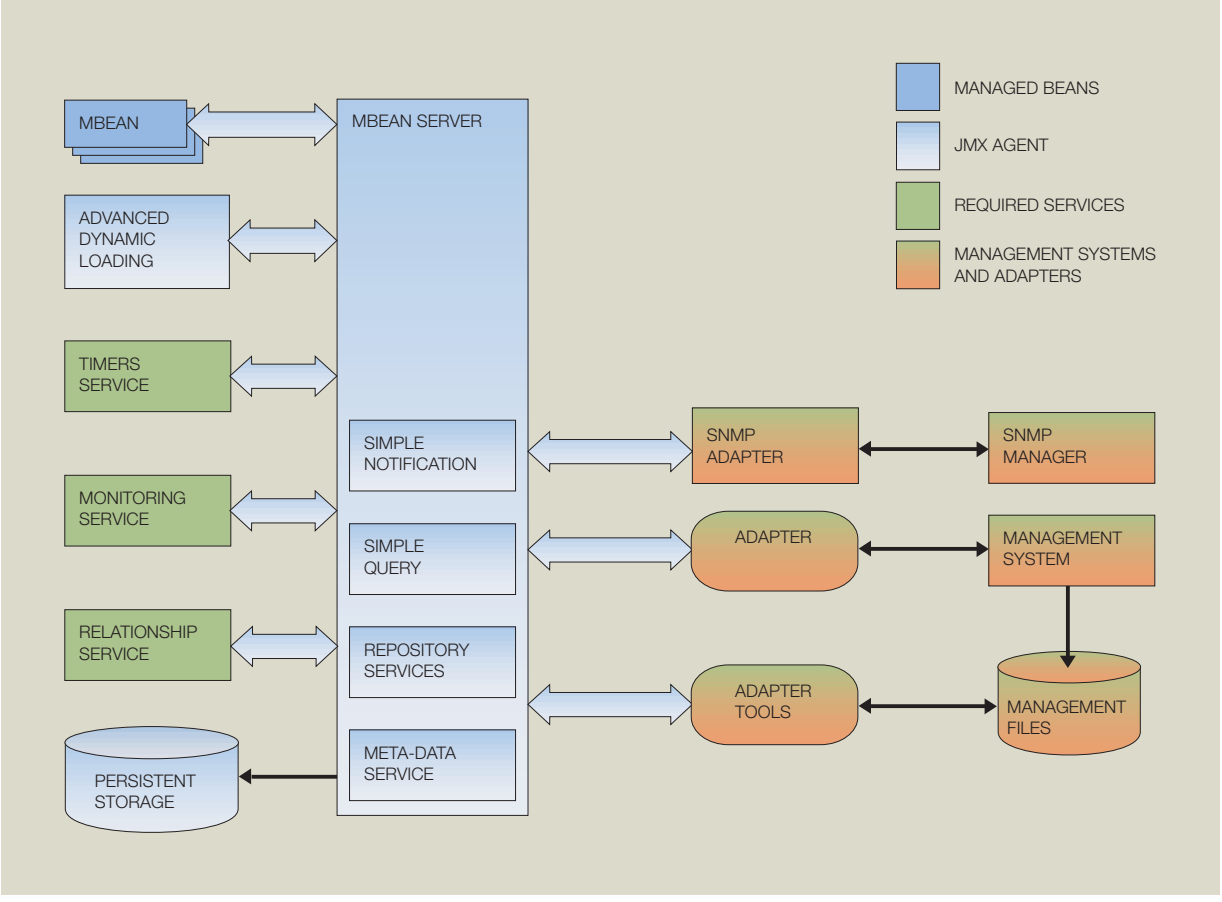
Specification organization and compliance levels. The JMX specification, reference implementation, and compliance requirement support management at three levels: instrumentation level, agent level, and manager level.

The *instrumentation* level defines (and provides APIs to support) implementation of Java technology-based resources so that they are manageable by any arbitrary management system through JMX managed beans (MBeans). Resources that conform to this level are *JMX manageable resources* (previously defined as applications or managed resources). The JMX instrumentation layer is defined in the *JMX Instrumentation and Agent Specification*.[1]

The *agent* level provides the repository of management data and basic common management interfaces used by JMX-managed resources, services, and adapters representing management systems. Resources that conform to this level are *JMX agents* (previously defined as management agents). The JMX agent layer is also defined in the *JMX Instrumentation and Agent Specification*.

The *manager* level provides the local and distributed management services used and implemented by man-

Figure 2 JMX components



agement systems. These services communicate with the JMX agent via adapters. Resources that conform to this level are *JMX managers*. The JMX specification does not define JMX managers at this time.

Component overview. The JMX architecture is based on four component types: managed resources and MBeans that comprise the instrumentation level, agents that comprise the agent level, and adapters that represent the manager level (see Figure 2). These map well to the management system components discussed earlier:

- *JMX managed resources* are instrumented using MBean objects. JMX specifies four types of MBeans: standard, dynamic, open, and model. Application instrumentors use or implement one or more of these types, exposing their management interface.

–*Standard MBeans* allow registration of any Java bean with the JMX agent. A definition of the management interface as a Java interface must be created during development and provided with the Java class.
–*Dynamic MBeans* allow the application- or domain-specific manager to define or generate the management interface for a resource at run time. This provides a simple way to wrap existing non-bean or even non-Java resources.
–*Open MBeans* are dynamic MBeans with restricted data types, so that class loading to use the bean is not necessary.
–*Model MBeans* implement a dynamic MBean that comes with the JMX agent. Any application can instantiate and customize the model MBean with management interface information, making it immediately useful. This drastically reduces the amount of code to be written for manageability,

and it protects the resource from Java virtual machine version and JMX agent implementation variances.

- A *JMX agent* consists of an MBean server and service MBeans. The MBean server runs in the local application environment (the Java virtual machine or a server known to the application) and is responsible for maintaining access to application management data, query support, and event-notification forwarding and generation. Applications communicate data and events to management systems through the MBean server. The MBean server can function as an instance factory for any MBean class. The JMX agent includes a monitoring service, relationship service, and MBean class loader. Additional management services can be added dynamically as service MBeans by applications or management systems. Thus the JMX agent is flexible and extensible.
- *JMX adapters* communicate between the JMX agent and their corresponding management systems. The adapter is responsible for translating between JMX and the manager and taking care of any issues related to remote communication. Since the adapter can be implemented to mimic the manager's supported agent technology, the management system may not even be aware that JMX is in the picture. There is at least one specific adapter for each management protocol or technology required to support different management systems.

Implementations of the JMX instrumentation API, JMX agent, and JMX adapters are available from a number of vendors, including Sun Microsystems, IBM, and AdventNet.

Scenarios

The following two scenarios illustrate JMX execution flow and include code samples. The scenarios use an SNMP management system because SNMP systems are more widely used and understood.

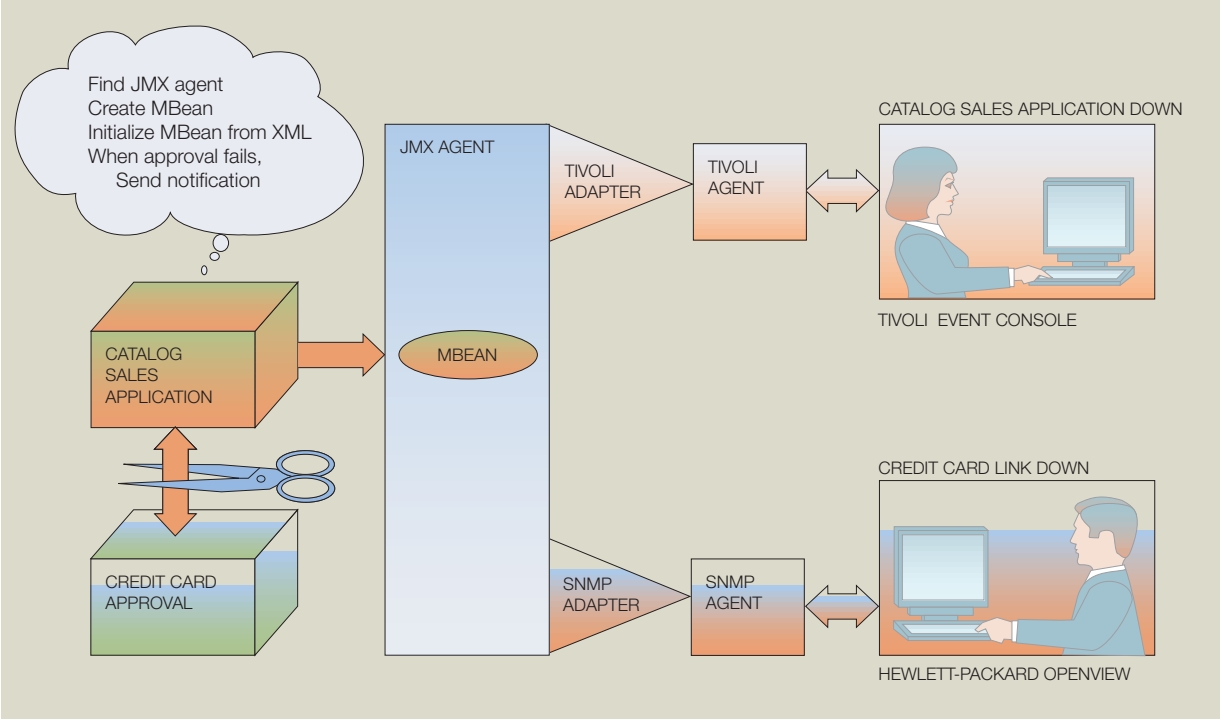
Event scenario. In this scenario our customer, BigCo, has a catalog sales application running on its Web site. This application accepts credit cards as payment for sales. The credit card information must be approved by the credit card company. BigCo uses the services of the Credit Card Approval (CCA) company. CCA gave BigCo a link to a Web site and servlet to call for credit card approval. Since no sales can be made without CCA's approval, BigCo wants to send an event to its management system whenever it receives an "HTTP 404" or "unable to connect" response from CCA's link.

BigCo uses both the Tivoli Management Environment (TME**) and Hewlett-Packard's OpenView for systems management, as shown in Figure 3. Operators of both systems want to be notified if the connection to CCA goes down. TME operators argue that it is a critical part of a revenue-producing business system, which it is. OpenView operators argue that the connection is over the network, which it is. Fortunately, BigCo uses JMX for its e-business management infrastructure and can easily pass events to both management systems using generic SNMP and Tivoli TEC[31] JMX adapters. If BigCo did not have JMX, its application developers would have to define an SNMP MIB and add an SNMP subagent to the catalog application. They would also have to develop a TEC adapter and definition files for each notification.

When the connection to CCA does not return an appropriate response, BigCo's catalog sales application will invoke its ModelMBean instance to send a notification to all registered listeners. In this case, both the Tivoli adapter and SNMP adapter are instantiated and registered with the MBean server. They are registered as listeners for events from the catalog sales application and both will receive the CCA link failure notification. The Tivoli adapter will translate the notification into a TEC event and send it to the Tivoli agent, which will send it to the Tivoli event console. At the same time, the SNMP adapter will transform the notification into an SNMP trap and, acting as an SNMP subagent, send it to the SNMP master agent, which will in turn send it to OpenView, the SNMP management system. Now both the system operations center and network operations center are aware that the link is not functioning properly and business is being lost.

As we can see in the simple program fragment in Figure 4, during application initialization the catalog application finds the JMX agent's MBean server by calling the static *FindMBeanServer* method for the *MBeanServerFactory* object. Using this reference, the application creates its model MBean named "csa." The catalog application includes an XML file that contains definitions for its management interface. It uses this file to create the meta-data object—ModelMBeanInfo—for itself. This object will contain the attributes, operations, and notifications in the catalog's management interface (see Figure 5). In the *runOrder* method of the catalog application, it catches an exception indicating that the credit card approval

Figure 3 JMX event scenario



servlet is not responding. It then uses the MBean server to have the "csa" MBean send a notification indicating that there is a problem.

XML interfaces for JMX have not been defined as a standard. However, an XML service MBean or other utilities can be used to parse the XML data into JMX objects. This XML fragment is an example of how to use the DMTF CIM XML DTD (document type definition) to define a management interface. In this example, the CIM method tag would contain JMX operations and the CIM qualifier tags would contain the JMX *MBeanInfo* and *ModelMBeanInfo* meta-data. We can see in this XML fragment that the catalog application, CatApp, has an attribute *CCAPort* of class CCA.intPort with a value of 8090, no methods, and a notification called CatApp.CCA.LostContact.

The sample program fragment in Figure 6 illustrates how an adapter might be organized. During initialization, it connects with the local SNMP master agent as a subagent. Then, just like the application, it must find the JMX agent's MBean server. It registers itself with the server and adds itself as a listener for notifications from the "csa" object. The *handleNotification* method of the NotificationListener interface will be invoked by the MBean server when it receives a notification from csa. The handleNotification method then transforms the notification into a trap and sends it to the SNMP master agent to be broadcast to all trap receivers. The Tivoli adapter would have similar functionality and flow.

Statistical data monitoring scenario. BigCo's OpenView operator is receiving too many traps indicating that the CCA connection is down. This unavailability affects business revenues, but before BigCo can complain about the lack of service, it needs to gather some statistics about how many requests are made during the day and map these, in a graph, with the connection events. Then the company can assess the effect on revenues in the same time frames. Using the SNMP manager and JMX, BigCo can gather the data from the application with little additional effort.

In this scenario, we assume the application is already initialized and registered with the JMX agent. The same diagram, program fragments, and XML frag-

Figure 4 Application code fragments for detecting and sending the notification

```
import javax.management.*;
...
Integer CCARequests=0, CCAErrors=0;
MBeanServer jmx = MBeanServerFactory.FindMBeanServer();
ObjectName csa = new ObjectName("MyAppServer:name=csa");
jmx.createMBean("RequiredModelMBean", csa));
jmx.invoke(csa, "setModelMBeanInfo",
                    new Object[] {ParseXMLFileIntoInfo("csl.xml")});
...
private ModelMBeanInfo ParseXMLFileInfoInfo(String filename) {
      /* returns a ModelMBeanInfo from the data in an XML formatted file */
      ...
}
...
public void runOrder(cardnum,order) {
      jmx.setAttribute(csa,new Attribute("CCARequests",CCARequests+1));
      try {
            if (CCApproval.getApproval(cardnum))
                  submitOrder(cardnum,order);
      } catch (CCNotAvailableException cce) {
            jmx.setAttribute(csa,new Attribute("CCAErrors",CCAErrors+1));
            jmx.invoke(csa,"sendNotification", new Object[] {
                              new Notification(
                              "CatApp.CCA.LostContact", csa, 001,
                              (new Date()).getTime()),
                              "Lost contact with credit card approval service"));
      }
}
...
```

ments from the previous scenario apply to this one. When the BigCo catalog application was installed, a "MIBGenerator" tool asked the JMX agent for all the registered applications and all the attribute and notification information for each of them. From these data, it constructed a generic application MIB and loaded the resulting file into the SNMP manager's compiled MIB. The SNMP manager operator has customized the SNMP manager to monitor the MIB values that indicate the CCARequestCount and CCAErrorCount.

Note that by returning cached data, the JMX model MBean minimizes the run-time impact on the application. If the attribute value was not in the cache or was stale, then the model MBean would invoke a *getCCARequest* method on the application. The application would receive the request, service it, and return the results to the model MBean.

Operation scenario. After monitoring the request and error rate of calls to CCA, BigCo found that the error rate was greatest during peak shopping hours. When CCA was approached for a solution, the company offered to upgrade BigCo to a higher-capacity, more reliable system on a different port. However, each transaction would cost twice as much. BigCo decided to use the high-speed port during peak shopping times and the lower-speed port the rest of the day. The TME operator was responsible for watching usage rates and changing the CCA port as appropriate.

BigCo also decided to add new items to its catalog and created a content management system for the catalog application. The purchasing agents will update this as they find new items for BigCo to sell. Updating the catalog and optimizing external connections for the application fall into different views

Figure 5 XML fragment defining the management interfaces for the catalog application's model MBean

```
...

<VALUE.OBJECT>
  <Class name="RequiredModelMBean">
    <Qualifier name="name" Type="java.lang.String">
        <value>"MyAppServer:name=csa"</value>
    </Qualifier>...
    <Property name=CCAPort Type="java.lang.Integer">
      <Qualifier name= IsIs type= "Boolean"><Value>FALSE</Value></Qualifier>
      <Qualifier name= Readable type= "Boolean"><Value>T</Value></Qualifier>
      <Qualifier name= Writeable type= "Boolean"><Value>T</Value></Qualifier>
      <Value>8090</Value>
    </Property>
    <Property name=CCARequests Type="java.lang.Integer">
      <Qualifier name= IsIs type= "Boolean"><Value>FALSE</Value></Qualifier>
      <Qualifier name= Readable type= "Boolean"><Value>T</Value></Qualifier>
      <Qualifier name= Writeable type= "Boolean"><Value>T</Value></Qualifier>

    </Property>
    <Property name=CCAErrors Type="java.lang.Integer">
      <Qualifier name= IsIs type= "Boolean"><Value>FALSE</Value></Qualifier>
      <Qualifier name= Readable type= "Boolean"><Value>T</Value></Qualifier>
      <Qualifier name= Writeable type= "Boolean"><Value>T</Value></Qualifier>
    </Property>

  </class>
...

<Notification name="CatApp.CCA.LostContact">
  <Qualifier name="description" type="java.lang.String">
    <Value>"Catalog Application has received a HTTP:404 from Credit Card
            Approval service."</Value>
  </Qualifier>
  <Qualifier name="notifytype" type="java.lang.String">
    <Value>"CatApp.CCA.LostContact"</Value>
  </Qualifier>
  <Qualifier name="log" type="java.lang.String">
    <Value>"T"</Value>
  </Qualifier>
  <Qualifier name="logFile" type="java.lang.String">
    <Value>"jmx.log"</Value>
  </Qualifier>
  <Qualifier name="messageId" type="java.lang.String">
    <Value>"CCA001"</Value>
  </Qualifier>
  <Qualifier name="severity" type="integer">
    <value>1</value>
  </Qualifier>
  <Value>"Lost contact with credit card approval service"</Value>
</Notification>
...

</VALUE.OBJECT>
```

Figure 6 Code fragment for the SNMP adapter to forward the notification to the SNMP management system

```
import javax.management.*;
import bigco.snmptoolkit.*;
...
ConnectToSNMPMasterAgent();
...
MBeanServer jmx = MBeanServerFactory.FindMBeanServer();
ObjectInstance regTrapper = jmx.registerMBean(this,
                                    new ObjectName("myAppServer:snmp.trapper"));
jmx.addNotificationListener(new ObjectName("myAppServer:name=csa"),
                                    this,null,null);
...
public void handleNotification(Notification ntfy, Object handback) {
        if (ntfy.getType( ).equals("CatApp.CCA.LostContact")) {
            SNMPTrap trapGuy = transformToTrap(ntfy);
            SNMPBroadcast(trapGuy);
        }
}
...
```

of responsibility and management systems. In this case, we do not want the TME operator to be aware that the addItems operation of the catalog customizer exists. Likewise, the catalog administrator should not be aware that the catalog has external connections. BigCo can use JMX to feed and handle both of the views and their support by different management systems.

Two methods are added to the catalog application: *reconfigureCCALink* for the TME operator and *addNewItems* for the BigCo buyers. We must also tie the MBean to the application instance that runs the methods. (See Figure 7.)

As we see in the program fragment in Figure 8, we have added to our previous example. We have tied this object to the RequiredModelMBean object to execute the operations in the *setManagedResource* invocation. The operation methods *reconfigure-NewLink* and *addNewItems* have been added. The XML fragment in Figure 9, describing the new methods, would be added to the XML file.

The two new methods can now be invoked by the TME and catalog customization adapters.

JMX components and interfaces

This section gives a detailed review of the JMX components and their interfaces.

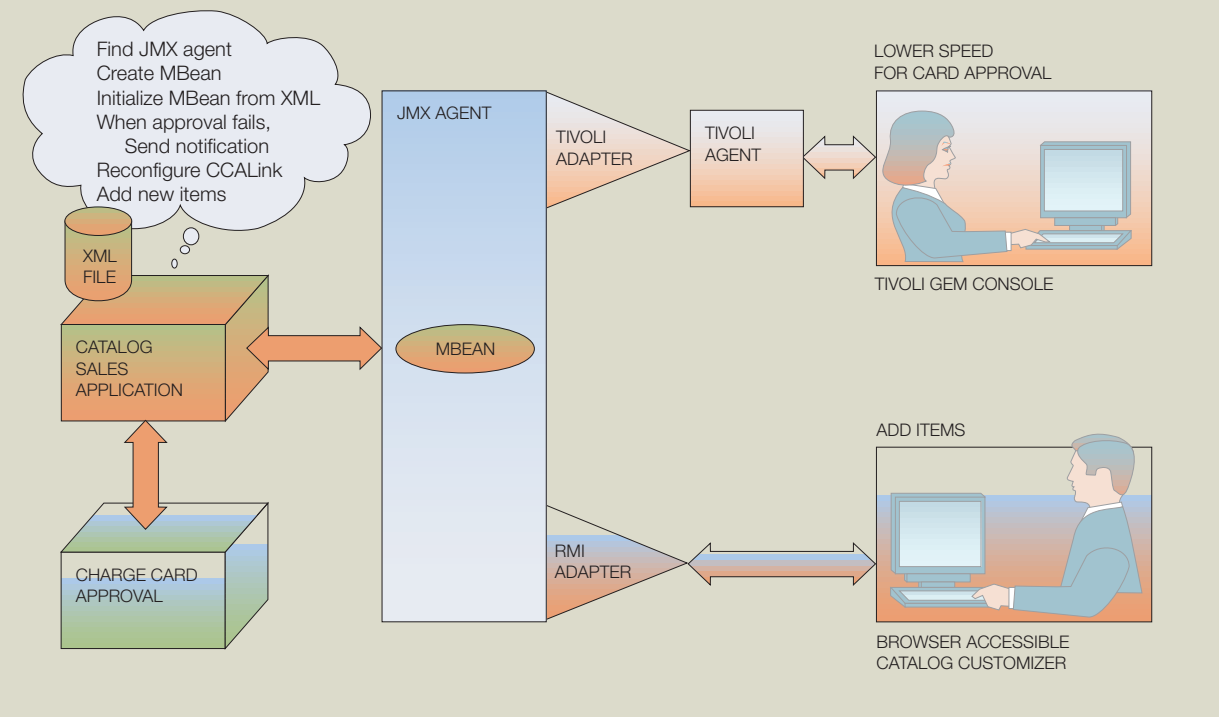
Instrumentation MBeans. Managed resources provide or use MBeans to expose their management interfaces. The management interface of an application consists of the attributes for configuration and metric data, operations, and events that the application exposes for use by a management system. The management interface is maintained in a meta-data object: MBeanInfo. MBeans are instantiated or registered with the JMX agent. This may be done by the MBean itself, another program, or an adapter. The JMX agent makes the MBean locatable by adapters and applications through query support. The JMX agent does not give out the reference to the MBean, only the name. This provides a controlled access point to institute security and distributability. The JMX agent will invoke the application's management interface methods on the MBean and return the responses to the request originator. All MBeans must implement notifications, transactions, and distributability in the same way.

Notifications. Any MBean can send JMX notifications by implementing the JMX NotificationBroadcaster interface. Any MBean can receive JMX notifications by implementing the JMX NotificationListener interface and registering for notifications.

Notification Interfaces:

NotificationBroadcaster Interface:
addNotificationListener

Figure 7 JMX operation scenario




```
      (NotificationListener listener,
      NotificationFilter filter,
      Object hb)
  removeNotificationListener
      (NotificationListener listener)
  MBeanNotificationInfo[]
      getNotificationInfo( )

NotificationListener Interface:
  handleNotification
      (Notification ntfy,
      Object, hb)

NotificationFilter Interface:
  boolean isNotificationEnabled
      (Notification ntfy)

Notification(String type,
      Object source,
      long sequenceNum,
      long timeStamp,
      String messageText)
```

Transactions. If the model MBean is executing in an environment where management operations are transactional, this should be shielded from the application. If the application must be aware of the transaction, then the application will depend on a certain version of the JMX agent and model MBean to be available.

Distributability. If the JMX agent is distributable then the application or adapters may be accessing MBeans that are not residing in the same Java virtual machine. The model MBean and the JMX agent must be implemented so that location transparency for the application and adapter is achieved. The JMX agent does not have to provide distributability for both the applications and the adapter; it may support only one of these being remote.

Standard MBeans. A standard MBean can be any JavaBeans program that has been registered with the MBean server. Implementation is performed by the application developer or application instrumentor. A Java interface must be defined for the program named *classname*MBean. This interface defines the management interfaces for the MBean. This may include all or just a subset of all the methods in the

Figure 8 Application code fragments for operations

```
import javax.management.*;
...
Integer CCARequests=0, CCAErrors=0;
MBeanServer jmx = MBeanServerFactory.FindMBeanServer( ) ;
ObjectName csa = new ObjectName("MyAppServer:name=csa");
jmx.createMBean("RequiredModelMBean", csa));
jmx.invoke(csa, "setModelMBeanInfo", new Object[ ] {ParseXMLFileIntoInfo("csI.xml")});
jmx.invoke(csa, "setManagedResource",
     (new Object[] {this, "objectReference"}));

...
private ModelMBeanInfo ParseXMLFileInfoInfo(String filename) {
       /* returns a ModelMBeanInfo from the data in an XML formatted file */
       ...
}
...
public void runOrder(cardnum,order) {
        jmx.setAttribute(csa,new Attribute("CCARequests",CCARequests+1));
        try {
              if (CCApproval.getApproval(cardnum))
                    submitOrder(cardnum,order);
        } catch (CCNotAvailableException cce) {
              jmx.setAttribute(csa,new Attribute("CCAErrors",CCAErrors+1));
              jmx.invoke(csa,"sendNotification", new Object[ ] {
                                    new Notification(
                                    "CatApp.CCA.LostContact", csa, 001,
                                    (new Date( ) ).getTime( ) ),
                                    "Lost contact with credit card approval service"));
        }
}
...
public void reconfigureCCALink(int level) {
        if (level == CCA.LOW)
             setCCAPort(8010);
        else
             setCCAPort(8090);
}
public void addNewItems(ItemList items) {...}
...
```

class. When the MBean is registered with the MBean server, the server will create an MBeanInfo metadata object by introspection on the *classname*MBean interface file. All get*AttributeName* and set*AttributeName* method pairs will create attributes for the bean. All other methods will create operations for the bean. The JMX agent will validate that requests are part of the management interface and then perform invocation directly on the MBean.

Applications using standard MBeans must implement the entire MBean and its interface. This can be useful if an application already has management interface classes to support its own manager. These

Figure 9 XML fragment defining the management operations to be added to the management interface in Figure 5

```
...
<VALUE.OBJECT>
  <Class name="RequiredModelMBean">
        ...
    <Method name="reconfigureCCALink" Type="void">
      <Qualifier name="description" type="java.lang.String">
        <value>"Set speed for Credit Card Approval"</value></Qualifier>
      <Qualifier name="impact" type="int"><value>1</value></Qualifier>
      <Parameter name="portSpeed" Type="int">
        <Qualifier name="description" type="java.lang.String">
          <value>"New speed of port: CCA.HIGH or CCA.LOW"</value></Qualifier>
      </Parameter>
    </Method>
    <Method name=addNewItems>
      <Qualifier name=="description" type==="java.lang.String">
          <value>"Add new products to web catalog application"</value>
      </Qualifier>
      <Qualifier name="impact" type="int"><value>1</value></Qualifier>
      <Parameter name="newItems" Type="ItemList">
        <Qualifier name="description" type="java.lang.String">
          <value>"List of Item objects to be added to catalog"</value></Qualifier>
      </Parameter>
    </Method>
...
  </class
...
</value.object>
...
```

class instances can simply be registered with the MBean server. These management interfaces are defined at development time.

Dynamic MBeans. A dynamic MBean can be an instance of any Java class that implements the DynamicMBean interface. Dynamic MBeans define their management interfaces at run time by maintaining their own MBeanInfo objects—a *classname*MBean interface is not necessary, as it is for standard MBeans. Dynamic MBeans are used to delegate to or wrap existing managed or management resources. These may be resources that do not follow JavaBeans design patterns, or that are not implemented using Java technology. Dynamic MBean implementations may be generated, developed by the application developer, or developed by the application instrumentor. Typically, an instrumentor will develop an MBean service that will instantiate or generate the dynamic MBeans. The dynamic MBean is responsible for maintaining its own MBeanInfo meta-data and providing it, on request, to the JMX agent during run time. The dynamic MBean is also responsible for implementing and validating correct invocation of the interfaces it defines in the MBeanInfo object. The MBean server delegates invocations of getAttribute, setAttribute, and invoke to the dynamic MBean.

```
DynamicMBean Interface:
  Object getAttribute
      (String attributeName)
  AttributeList getAttributes
      (String[] attributeNames)
  void setAttribute
      (Attribute newAttribute)
  AttributeList setAttributes
      (AttributeList newAttributes)
  Invoke
      (String actionName,
       Object[] parms,
       String[] signature)
  MBeanInfo getMBeanInfo( )
```

Open MBeans. Open MBeans are dynamic MBeans that are restricted to accepting and returning a limited number of data types. By using open MBeans and these basic data types, the need for class loading is eliminated. This can make it easier to deploy the MBean and support a highly distributed system. However, this does not remove the need to understand the semantics of the data to be passed or returned. The open MBean data types can be:

- Basic data types: int, boolean, float, double, etc.
- Class wrappers for basic data types: Integer, Boolean, Float, Double, String, etc.
- Tables: array of rows of the same basic data type
- Composites: objects that can be decomposed into other open data types

Open MBeans must return an OpenMBeanInfo object from the getMBeanInfo method. OpenMBeanInfo extends MBeanInfo and adds some additional meta-data, legal values, default values, etc. The OpenMBean classes are not available in the current JMX reference implementation.

Model MBeans. The RequiredModelMBean class is an implementation of model MBean interfaces that must be provided in conjunction with a JMX agent. The JMX agent functions as a factory for model MBeans—model MBean instances are created and maintained by the JMX agent. This allows the RequiredModelMBean class implementation to vary depending upon the needs of the environment in which the JMX agent is installed. The application requesting the instantiation of the RequiredModelMBean object does not have to be aware of the specifics of the implementation of the RequiredModelMBean class. Implementation differences between JMX and Java virtual machine environments may include persistence, transactional behavior, caching, performance requirements, location transparency, distributability, etc. The RequiredModelMBean object is responsible for implementing and managing these differences internally. The JMX agent that instantiates the RequiredModelMBean object may be specifically designed to support a particular RequiredModelMBean class.

Since the model MBean implementation is provided by the JMX agent, the application does not have to implement the model MBean, just customize and use it. The instrumentation code is consistent and minimal. The application gains the benefit of default policy and support for logging events, data persistence, data caching, and notification handling. The application initializes its model MBean and passes its identity, management interface, and any policy overrides. The application can add custom attributes to the model MBean during execution. The application-specific information can be modified without interruption during run time. The model MBean then sets its behavior interface and does any set-up necessary for event logging and handling, data persistence and currency, and monitoring. The model MBean default behavior and simple APIs will satisfy the management needs of most applications, but also allow complex application management scenarios.

```
ModelMBean Interface:
  Implements DynamicMBean,
      PersistentMBean, and
      ModelMBeanNotificationBroadcaster)
  Supports ModelMBeanInfo
  setModelMBeanInfo
      (ModelMBeanInfo mbi)
  setManagedResource
      (Object managedObjRef,
       String ref_type)
```

The model MBean must implement the ModelMBean interface, DynamicMBean interface, PersistentMBean interface, and ModelMBeanNotificationBroadcaster interface. The ModelMBeanBroadcaster interface supports generic text notifications and attribute change notifications. Most importantly, a model MBean has a ModelMBeanInfo meta-data object. ModelMBeanInfo extends MBeanInfo and adds descriptors to the meta-data for all attributes, operations, and notifications in MBeanInfo. Model MBeans use the descriptors to support policy for activities such as caching, persistence, and logging.

Descriptors. The model MBean provides MBeanInfo meta-data as well as descriptor meta-data. The MBeanInfo interface publishes meta-data about the attributes, operations, and notifications in the management interface. The model MBean descriptors contain behavioral information about the same management interface. A descriptor is a set of keyword/value pairs that can be accessed with the descriptor interface.

```
Descriptor Interface:
  Object clone( )
  String[] getFieldNames( )
  String[] getFields( )
  Object getFieldValue
```

```
    (String fieldName)
  Object[] getFieldValues
    (String[] fieldNames)
  boolean isValid( )
  void removeField
    (String fieldName)
  void setField
    (String fieldName,
     Object fieldValue)
  void setFields
    (String[] fieldNames,
     Object[] fieldValues)
```

A set of keywords has been defined by the JMX specification for standard and uniform functionality and treatment. New keywords can be added by applications or adapters at any time. These are the current, standard descriptor keywords:

```
In MBean Descriptor:
  name, version, visibility, export, persistPolicy,
  persistPeriod, persistLocation, log, logFile,
  CurrencyTimeLimit

In AttributeDescriptors:
  name, value, default, legalValues,
  displayName, getMethod, setMethod,
  protocolMap, persistPolicy, persistPeriod,
  currencyTimeLimit, lastUpdatedTimeStamp,
  iterable, visibility, presentationString

In OperationDescriptors:
  name, displayName, role, impact,
  targetObject, targetType, lastReturnedValue,
  currencyTimeLimit, lastReturnedTimeStamp,
  visibility, presentationString

In NotificationDescriptors:
  name, severity, messageId, log, logFile,
  visibility, presentationString
```

Descriptors at the MBean level define default policies for the attributes, operations, and notifications for persistence, caching, and logging. The application can pass and use an XML or properties file to create and initialize the ModelMBeanInfo object and descriptors instead of customizing them programmatically. This permits management interfaces to be defined in a language-independent and portable manner. The DescriptorAccess interface is used to retrieve a copy of or fully replace a management interface element's descriptor.

```
DescriptorAccess interface:
  Descriptor getDescriptor( )
  void setDescriptor(Descriptor inDescriptor)
```

Attribute handling. The model MBean attribute descriptor includes policy for managing the object's persistence, caching, protocol mapping, and how get and set requests are handled. When the model MBean is created by the managed resource, the resource defines the operations that will be executed by the MBean that will satisfy get and set requests. By defining operations, the actual methods called to satisfy get and set requests are allowed to vary and to be delegated to a wide range of objects at run time. This allows management of distributed, dynamic applications. If an attribute has no operation associated with it, then the values are maintained in the MBean. This can also minimize managed resource interruption for static resource information.

Attribute value caching is supported. In general, if the data requested by the adapter are current, then the managed resource is not interrupted with a data retrieval request. Therefore, direct interaction with the managed resource is not required for each interaction with the management system. This helps minimize the impact of management activity on runtime application resources and performance.

Operation handling. A model MBean can invoke all defined operations on the same target object instance (defined with the setManagedResource method on the bean), or it can define different target object instances for different operations (defined in the TargetObject field of the operation descriptor). This allows distributed, component-based applications to be supported by the bean. It also supports management of applications that do not already have a management facade. As with attributes, the model MBean supports caching the last returned value of the operation. Caching can reduce the interruptions to the managed application.

Notification handling. Model MBeans support a generic, text-only notification for use by applications. They also send attribute change notifications whenever an attribute's value is changed. Application-specific notifications are supported as for any other MBean. The model MBean notification descriptor defines whether it should be logged, as well as a severity value, visibility value, and message identifier for national language support.

```
ModelMBeanNotificationBroadcaster Interface:
  extends NotificationBroadcaster
  addAttributeChangeNotificationListener
      (NotificationListener inlistener,
       String inAttributeName,
       Object inHandback)
  removeAttributeChangeNotificationListener
      (NotificationListener inlistener,
       String inAttributeName)
  sendAttributeChangeNotification
      (Attribute inOldVal,
       Attribute inNewVal)
  sendAttributeChangeNotification
      (AttributeChangeNotification ntfyObj)
  sendNotification(Notification ntfyObj)
  sendNotification(String ntfyText)
```

Persistence. The model MBean is responsible for its own persistence. This does not mean that it *must* persist. It is foreseeable that some implementations of the JMX agent will be completely transient in nature. In a simple implementation the model MBean may be saved in a flat file. In a more complex environment, persistence may be handled by the JMX agent in which the MBean has been instantiated. If the MBean persists, it should support the persistence policy at both the attribute level and the MBean level. The persistence policy may switch persistence off, force persistence on checkpoint intervals, allow persistence to occur whenever the MBean is updated, or "throttle" the update of persistent data so that information is not written out any more frequently than a certain interval. Since persistence policy can be set at the attribute level, all or some of its attributes can be saved by the model MBean. In all cases, the saved MBean may then be used to prime the next instantiation of the application or its model MBean.

```
PersistentMBean Interface:
  void load( )
  void store( )
```

JMX agents can be independent and ignorant of data locale. The data location can vary from one installation to another depending on how the JMX agent and managed resource are installed and configured. Application configuration data can be defined within the directory service for use by multiple application instances or JMX agent instances. Data locale has no effect on the interaction between the application, its model MBean, the JMX agent, the adapter, or the management system. As with all data persistence issues, the platform data services capabilities may affect performance and security.

JMX agent responsibilities. The JMX agent includes the MBean server and all registered management service MBeans. The MBean server maintains a registry or repository of the current set of registered MBean names and references, delegates method calls to, acts as a factory for, and acts as a query service for MBeans.

```
MBeanServerFactory Interface:
  static MBeanServer createMBeanServer( )
  static MBeanServer createMBeanServer
    (String domain)
  static ArrayList findMBeanServer
    (String AgentId)
  static MBeanServer newMBeanServer( )
  static MBeanServer newMBeanServer
    (String domain)
  static void releaseMBeanServer
    (MBeanServer mbeanServer)

MBeanServer Interface:
  ObjectInstance createMBean
    (String className,
     ObjectName name,
     ObjectName loader,
     Object parms[], String[] sig)
  Object instantiate
    (String className,
     ObjectName loader,
     Object [] parms, String sig)
  ObjectInstance registerMBean
    (Object mbean, ObjectName name)
  unregisterMBean(ObjectName name)
  Object getAttribute
    (ObjectName name, String AttrName)
  setAttribute
    (ObjectName name, Attribute newAttr)
  AttributeList getAttributes
    (ObjectName name,
     String[] attributeNames)
  AttributeList setAttributes
    (ObjectName name,
     AttributeList newAttrs)
  Object invoke
    (ObjectName name,
     String actionName,
     Object[] parms, String[] sig)
  MBeanInfo getMBeanInfo
    (ObjectName name)
  boolean isRegistered
```

```
    (ObjectName name)
  ObjectInstance getObjectInstance
    (ObjectName name)
  String getDefaultDomain()
  Integer getMBeanCount()
  Set queryMBeans
    (ObjectName name,
     QueryExp query)
  Set queryNames
    (ObjectName name,
     QueryExp query)
  ObjectInputStream deserialize
    (String className,
     ObjectName loader,
     byte[] data)
  addNotificationListener
    (ObjectName name,
     NotificationListener listener,
     NotificationFilter filter,
     Object hbo)
  removeNotificationListener
    (ObjectName name,
     NotificationListener ntfy)
```

Instantiation. The MBeanServer class includes interfaces for instantiation of any class by the MBean server, allowing the MBean to be tied to the life or implementation of the MBean server rather than the resource. It also allows the implementation of the MBean to vary between instances of the application and the server without affecting the application instrumentation code interacting with the MBean. This feature is used heavily by model MBeans.

Registration. When an MBean registers with the MBean server, the server will add it to its current set of MBeans. If the MBean has implemented the MBeanRegistration interface, the server will call it during registration processing.

The management adapters will receive an MBeanServerNotification.Registration notification from the JMX agent whenever a new instance of the MBean is created and registered. In this way, the management adapters become aware of each system-managed application and can map its event behavior appropriately. This allows management system customization for the new applications or updates to event-driven resource inventory and discovery services.

Query of MBeans and MBeanInfo objects. The JMX agent maintains access to the current set of MBeans and adapters. Adapters can collect data from the agent to respond to discovery or inventory probes from the management system. The JMX agent returns names of relevant MBeans to applications and adapters that need to operate on them. Applications and adapters can query a JMX agent for sets of MBeans based on pattern matching on the MBeans' names. The MBeans are only accessible via the MBean server by name—the server will not return a reference to any MBean. Whenever the MBeanInfo meta-data are requested for a standard MBean, the meta-data are returned by the MBean server. For dynamic and model MBeans, meta-data are requested from the MBean itself—the JMX agent is not responsible for maintaining or policing these meta-data.

Delegation. When an MBeanServer interface is called for setAttribute, getAttribute, or invoke methods, the request is delegated to the MBean. The targeted MBean's name is the first parameter of the method. If it is a standard MBean, the actual get, set, or *operationName* method will be called directly; no JMX-specific interface needs to be implemented. If it is a dynamic MBean, the JMX agent will delegate the request via the DynamicMBean interface. This consists of setAttribute, getAttribute, and invoke methods mirroring the MBean server interface. The dynamic MBean is responsible for either delegating or responding to the request directly. This allows the interface to the managed application to be set at run time. Model and open MBeans implement the DynamicMBean interface and are treated in the same way. Responses from MBeans to the requests from the JMX agent are returned to the request originator, which is usually an adapter.

Management service MBeans. The JMX agent provides a relationship service. This service allows creation of an independent relationship MBean to represent any relationship between two or more MBeans. Some typical relationships are collection, containment, dependency, and parent-child.

The JMX agent can perform local application data and threshold monitoring with the JMX monitor service and monitor MBeans. Monitoring intervals and threshold rules are configurable by the application or the management system via the JMX agent interfaces. The value of a monitored attribute is retrieved at monitoring intervals. The monitor evaluates the value and publishes notifications as needed.

The JMX agent also provides a timer service that allows operations to be defined at a specific time or

after a specific period of time has elapsed. This is an optional service.

Managed application responsibilities. Each application uses MBeans to expose its management data, operations, and events for use by a management system. At initialization the application obtains access to the JMX agent through the static FindMBeanServer method. This will return a list of references to MBean servers in the same Java virtual machine. The application will then create or find, and then use, one or more instances of its MBean. The MBean name consists of the agent's domain identifier and a list of *keyword* and *value* pairs, called attributes. The predefined attributes that are part of the MBean name are used to establish a unique application identity.

Applications using standard or dynamic MBeans must implement these beans. Dynamic MBeans may require less programming, because they can wrap or delegate method calls to existing application resources. Since the model MBean implementation is provided by the JMX agent, the applications can simply customize and use it.

The application exposes values for its management data by updating its MBean with a single setAttributes method call for publication to or consumption by all management systems. The application sets and updates any type of data as an attribute in the MBean when it is convenient for the application. Since the MBean can be persistent and is locatable, critical but transient applications can retain any required counters or state information within the JMX agent. Likewise, with persistent MBeans, the application's data survive recycling of the JMX agent.

An application sends event notifications to all interested management systems with one sendNotification method call on its MBean. Predefined or unique notifications can be sent for any application- or management system-defined significant event. Notifications are normally sent when operator intervention is required or the application's state is significantly changed (examples of significant state transition events might be an unavailable resource, an exception, or other nonrecoverable error causing failure of a component of the application). Applications that need to publish notifications to JMX notification listeners must implement the JMX NotificationBroadcaster interface in their MBeans. The application publishes its notifications (also known as events) via the sendNotification method. Adapters and applications wishing to receive these notifications must subscribe to them with the addNotificationListener method and implement the JMX NotificationListener interface handleNotification method. The model MBean always implements the NotificationBroadcaster interface.

MBeans can be created that support attribute change notifications for any or all attributes. The MBean sends an AttributeChangeNotification event to interested adapters and applications whenever a value change for the attribute occurs. The interested application (adapter, managed resource, or JMX monitor MBean) registers for notification of changes in attribute values using the AttributeChangeNotificationFilter method. The setAttribute operation on an MBean will initiate an AttributeChangeNotification event to notification listeners. By default, no AttributeChangeNotification events will be sent unless a listener is explicitly registered for them. Normally, the setAttribute method on the MBean invokes the corresponding *set* method defined for the attribute on the application directly. Alternatively, managed resources can use the AttributeChangeNotification event to trigger internal actions to implement the intended effect of changing the attribute. The model MBean supports issuing AttributeChangeNotification events whenever attribute values are changed.

The application can represent itself with a set of MBeans and create relationships among them. These relationships can represent containment, dependency, path, or any other relationship role the application may choose to define. JMX relationships can be one-to-one, one-to-many, or many-to-many.

Adapter responsibilities. The adapter understands the management system protocols and maps them to the data and capabilities of the MBean and JMX agent. When the adapter receives a command from the management system, it interacts with the JMX agent to satisfy the request. The adapter then generates and sends a response to its management system. The adapter owns the data definition that is appropriate for the management system it supports. This includes mapping to and from the management system data representation and the JMX agent and MBean data representation. For instance, the SNMP JMX adapter owns the MIB definition used to access application data supported by the JMX agent and MBeans. The adapter also provides the interfaces and any supporting technology for communicating with its management system.

Model MBeans provide the description for the mapping of the application's managed attributes to existing management data models, i.e., specific MIBs or CIM objects. The adapter can use these mappings as hints on how to represent the data to the management system. Conversely, the adapter can take advantage of generic mappings to MIBs and CIM objects generated by tools interacting with the JMX agent. For example, a JMX MIB generator can interact with the JMX agent and create a MIB file that is loaded by an SNMP management system. The generated MIB file can represent the resources known by the JMX agent. The applications represented by these resources do not have to be aware of how the management data are mapped to the MIB. This scenario will also work for other definition files required by management systems—AMS, MIF (Management Information Format), MOF, etc.

The adapter registers with the JMX agent during initialization. Then it can register for notifications from any interesting MBeans. The adapter can pass custom notification filters that prevent the adapter from receiving certain notifications based on the identifier, origin data, type, and severity of the notification. The adapter listens for notifications from the JMX agent's MBeans or event service, analyzes them, and transforms them into traps or events and forwards them to the management system.

In order to retrieve application data maintained by the JMX agent or invoke an operation on an application, the adapter asks the JMX agent for the name of the MBean that represents the application or component. The adapter then invokes the MBean's methods through the JMX agent to accomplish its task. The adapter receives the responses to the requests from the specific application's MBean and maps and recasts this into a response to the management system.

Adapters can connect to application-specific management systems as well as enterprise management systems. This means that developers can use one set of management instrumentation to satisfy the needs of both. Developers will discover that application-specific configuration, availability, and operations management user interfaces are easy to develop based on the JMX agent and MBeanInfo meta-data available to them. It is feasible to provide a *manager servlet* that invokes the JMX agent or adapter from HTML or JSP** (JavaServer Pages**) and returns the results of the request to the browser. This provides a consistent and relatively simple means to generate user interfaces and to display information for any application on any host (or several hosts on one form) through a browser. If the application supports operations through the JMX agent, it thereby provides an interface for an operations user interface.

Management system behavior. The management system behavior is not affected by the use of JMX between the management systems' agents and the application. Thus the management system communicates with its agents (proprietary or standard) to exchange application status and data. It processes events from and sends data update requests and commands to its agents. The management system communicates with the management interfaces as agents, or through agents communicating with adapters as subagents. It is not aware of the existence of the JMX agent, other than as an additional internal application that requires some level of management. Enterprise management systems may be developed to interact with JMX agents directly (via the MBean server interface or a general-purpose RMI adapter) rather than through their existing agent infrastructure, but it is not absolutely necessary. Application-specific management systems (designed to manage one particular application or component of the enterprise environment) can use JMX (in an application role) to forward its management information into the existing management system. These application management systems can also communicate with the applications or components using a management adapter as well.

The JMX vision

Customers have been asking for end-to-end, business system, and application-oriented management solutions for years. And yet, even application management has been an elusive and nearly unobtainable goal for enterprise management system vendors. This mismatch of need and solution can be traced to one missing component: the lack of widespread participation in management by applications. Network and systems management systems have been successful largely due to widespread expectation that all devices and systems would support being managed through SNMP "out of the box." When applications are subjected to this same expectation—being manageable as delivered through a *de facto* standard management technology—then application management becomes feasible. End-to-end business system management becomes possible. JMX can be the *de facto* standard management technology catalyst necessary initiate this chain of events.

Once JMX is part of the foundation application-execution environment infrastructure, it can collect operational, statistical, relational, and state management data. At the very least, the aggregated data and relationships could feed more sophisticated management systems that would allow distributed (Web and enterprise) application operations to be more fully managed. Operators and administrators could perform well-informed relative prioritization of resources among applications. They could aggregate components into applications and applications into business systems. With widespread enablement through JMX, automatic (very rapid) correlation of events and reaction to events across the system would be feasible. JMX's data would act as a knowledge base accessed by a management system that would perform distillation of cause and solution and perhaps offer response recommendations to operations centers or automated response systems.

Since JMX can provide the foundation for operational standards and control, it can support definition and enforcement of service level agreements. Service level agreements encapsulate the expectation of certain guaranteed levels of performance (particularly response time), automated recovery (fail over), and throughput at an aggregated systems level for a managed infrastructure. The current state of the art is *ad hoc*. That is, the consumer of resources in the distributed environment can, at any time, experience a busy signal, unpredictable results, and delays as a result of transient outages or state migrations within a distributed complex. Moreover, this is an area that is not well understood or addressed in the distributed, Web-centric, enterprise management environment, which begs for solutions that are highly automated and just-in-time in nature.

A simple, everyday example of the current unsatisfactory state of operational standards and control can be experienced by a user at any time by the simple request for an arbitrary URL (uniform resource locator) from a browser. The user has no guarantee that the URL will be resolved in a timely manner, if at all. As a result, the user may wait for unacceptable periods, see time-outs, or experience generally delayed operations for no explicable reason. Unfortunately, the cause cannot be reliably and rapidly addressed by the system administrator (or by the service provider) who receives the outage calls from potentially irate service users.

Today, for these scenarios, the service provider has inadequate or nonexistent instrumentation and tools. Service providers resort to crude rule-of-thumb and after-the-fact solutions that consist primarily of some form of capacity expansion (ordering and installing more capability for the next time). Inevitably, this becomes a never-ending reactive exercise, as the peak load and throughput requirements of the system inexorably grow over time.

Because of the fundamental nature and placement of JMX as a gatekeeper of management data, actions, and events, JMX has the *long-term* potential to provide the instrumentation that will be the key to addressing certain aspects of the service level problem in an automated, preventive, real-time manner. For example, because the management system would have sufficient information and controls, it may be able to modulate certain critical load levels in the system during key periods, analyze the best usage patterns in a given environment at a given time, automatically dampen, in near real time, certain noncritical demands on a given system during peak-load or critical periods in favor of higher-priority requirements, as defined by the administrators responsible for the infrastructure and applications.

Conclusion

Java Management eXtensions is a first-version specification and an emerging technology. It offers the hope of insulating Java application developers from customer management system choices, just as JDBC** (Java Database Connectivity) does for database choices. It also sets the stage for more sophisticated, intelligent, and complete management systems. With the backing of the major enterprise management vendors, including Tivoli, Computer Associates, and BullSoft, there is some assurance for developers that JMX-instrumented applications will be supported by enterprise management vendors.

JMX v 1.0 is a reasonable management infrastructure for J2SE environments; however, it is currently limited by its single Java virtual machine design point. It would be difficult to use JMX technology, in its current form, in a distributed application model like Enterprise JavaBeans in J2EE** (Java 2 Platform, Enterprise Edition). There is hope that this will be corrected in the next iteration of the JMX specification, as this requirement is well known. J2EE management is being developed with its own expert group and JMX is currently on its supported technologies list. The JMX and J2EE expert groups both include BEA Systems, GemStone, Iona, IBM, and iPlanet. The

presence of these vendors should assure that the JMX solution for J2EE is complete and viable.

Other management technologies currently being developed in standards organizations, specifically CIM/WBEM in the DMTF, and ARM and the Manageability Service Broker from The Open Group, should be monitored. These technologies will need to be considered, along with JMX, when management technologies are being chosen for applications under development. As these technologies mature and gain adoption throughout industries, there may be a need to support them in addition to JMX. Fortunately, the JMX architecture provides an API into which these technologies can be adapted as JMX agents, or conversely, where JMX enablement is adapted into these newer management technologies. In the meantime, JMX is a sound technology choice for developing manageable applications.

To summarize, enterprise and e-business applications should be developed with manageability in mind if they are to be part of the business processes. The components of these applications implemented with Java technology can be instrumented for manageability using JMX. This development investment is protected by the involvement of the enterprise management and Java communities.

Acknowledgments

I would like to dedicate this paper to the memory of Keith Duvall in acknowledgment and appreciation of his enormous contribution to its content and quality. I would also like to recognize his support and encouragement during our work with the JMX technology and expert group. I would like to acknowledge the review and support of this paper from Brett Coley, Peter Brittenham, Karl Gottschalk, John Feller, Vera Plechash, Ray Williams, John Sweitzer, and Karl Schopmeyer.

Cited references and notes

1. Java Management Extensions; available at http://java.sun.com/products/JavaManagement.
2. J2SE (Java 2 Platform, Standard Edition) is Sun MicroSystems' Java platform. More information is available at http://java.sun.com/j2se/.
3. Sun Microsystems Jini. Technology information is available at http://www.sun.com/jini/index.html.
4. Universal Plug and Play (UPnP). See http://www.upnp.org/.
5. Hewlett-Packard's e-speak product. See http://www.e-speak.hp.com/.
6. IBM's WebServices Initiative information is available through http://www.alphaworks.ibm.com/tech/webservicestoolkit.
7. IBM System/390 Series is available through http://www.s390.ibm.com/.
8. Amdahl Millennium. Information available through http://www.amdahl.com/ or through Amdahl Corporation, 1250 East Arques Avenue, Sunnyvale, CA 94088-3470.
9. Candle Corporation, 201 N. Douglas St., El Segundo, CA 90245. See http://www.candle.com/.
10. Computer Associates International, Inc., One Computer Associates Plaza, Islandia, NY 11749. See http://www.cai.com/.
11. SNA (Systems Network Architecture) is an IBM proprietary architecture for network computing within an enterprise. More information is available through http://www.ibm.com/.
12. TCP/IP (Transmission Control Protocol/Internet Protocol) is an Internet protocol standard defined by the Internet Engineering Task Force. Additional information is available through http://www.ietf.org/.
13. Tivoli NetView for 390. See http://www.tivoli.com/products/index/netview_390/.
14. Hewlett-Packard's OpenView product. More information is available through http://www.openview.hp.com?qt=OpenView/ or from Hewlett-Packard, 3000 Hanover Street, Palo Alto, CA 94304-1185.
15. Tivoli Systems Inc., 9442 Capital of Texas Highway North, Arboretum Plaza One, Austin, TX 78759. See http://www.tivoli.com/.
16. BMC Software, Inc., 2101 Citywest Blvd., Houston, TX 77042-2827. See http://www.bmc.com/.
17. CMIP (Common Management Information Protocol) is usually referred to in conjunction with CMIS (Common Management Information Services). This management standard was defined by OSI (Open Systems Interconnection) as Standard ISO 9595/2 and 9596/2 (International Standards Organization: see http://www.iso.ch/).
18. SNMP (Simple Network Management Protocol) is an IETF standard. See http://www.ietf.org/.
19. DMTF (Distributed Management Task Force) is a standards body responsible for DMI (Distributed Management Interface), CIM, and WBEM management standards. See http://www.dmtf.org.
20. CIM/WBEM (Common Information Model/Web-Based Enterprise Management) is defined by the DMTF. More information is available through http://www.dmtf.org.
21. Systems Management Server (SMS) is Microsoft's workstation management application. More information is available through http://www.microsoft.com/smsmgmt/default.asp?RLD=263.
22. See http://www.snia.org.
23. DPI (Distributed Program Interface, IETF RFC 1228) is used to dynamically extend the MIB in SNMP agents. Support for DPI is predominant in IBM systems. SMUX (SNMP Multiplexing Protocol, IETF RFC 1227) allows an application to communicate with an SNMP agent to satisfy a portion of the MIB. SMUX is predominant in UNIX systems.
24. AgentX is a standard SNMP agent-to-subagent protocol being defined in the IETF as RFC2741. More information is available at http://www.ietf.org/rfc/rfc2741.txt?number=2741.
25. IETF (Internet Engineering Task Force) defines standards for Internet technologies, including routing, security, transport, and management. See http://www.ietf.org/.

26. IBM WebSphere. Administration console information on WebSphere is available through http://www-4.ibm.com/software/webservers/appserv/.
27. *Understanding the Application Management Model Version 1.0*, The Distributed Management Task Force, Inc. (May 17, 1998). Available at http://www.dmtf.org/spec/whitepapers/CIM_Applications_wp.htm.
28. Marimba, Inc., 440 Clyde Ave., Mountain View, CA 94043. See http://www.marimba.com/.
29. Tivoli's application management specification (AMS) file is used to define the characteristics of a managed application. See http://www.tivoli.com/products/index/module_designer/resources/ams_v20.pdf.
30. MOF (managed object format) file. This format is used to describe CIM information and is defined by the DMTF in the CIM specification. See http://www.dmtf.org/spec/cim_schema23/.
31. TEC (Tivoli Enterprise Console). See http://www.tivoli.com/products/index/tec/. Currently there is no TEC JMX adapter that is publicly available.

General references

W. Bumpus, J. W. Sweitzer, P. Thompson, A. R. Westerinen, and R. C. Williams, *Common Information Model Implementing the Object Model for Enterprise Management*, John Wiley & Sons, Inc., New York (2000).

G. Carpenter and B. Wijnen, *SNMP-DPI Simple Network Management Protocol Distributed Program Interface* (May 1991); available at http://www.ietf.org/rfc/rfc1228.txt?number=1228.

J. Case, M. Fedor, M. Schoffstall, and J. Davin, *Simple Network Management Protocol*, STD 15, RFC 1157 (May 1990); available at http://www.ietf.org/rfc/rfc1157.txt?number=1157.

CIM for XML Document Type Definition Version 2.0, The Distributed Management Task Force, Inc. (July 20, 1999); available at http://www.dmtf.org/download/spec/xmls/CIM_DTD_V20.txt.

Common Information Model Schema: Version 2.3, The Distributed Management Task Force, Inc. (July 14, 1999); available at http://www.dmtf.org/spec/cim_schema_v23.html.

Common Information Model (CIM) Specification Version 2.2, The Distributed Management Task Force, Inc. (July 14, 1999); available at http://www.dmtf.org/spec/cim_spec_v22.

M. Daniele, B. Wijnen, M. Ellison, and D. Francisco, *Agent Extensibility (AgentX) Protocol Version 1*, RFC2741 (January 2000); available at http://www.ietf.org/rfc/rfc2741.txt?number=2741.

J. D. Murray, *Windows NT SNMP*, O'Reilly and Associates, Inc., Sebastopol, CA (1998).

M. Rose, *SNMP Multiplexing Protocol and MIB*, RFC 1227 (May 1991); available at http://www.ietf.org/rfc/rfc1227.txt?number=1227.

M. Rose and K. McCloghrie, *Concise MIB Definitions*, STD 16, RFC 1212 (March 1991), available at http://www.ietf.org/rfc/rfc1212.txt?number=1212.

M. T. Rose and K. McCloghrie, *How to Manage Your Network Using SNMP: The Networking Management Practicum*, Prentice Hall Inc., Englewood Cliffs, NJ (1995).

M. Rose and K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-Based Internets*, STD 16, RFC 1155 (May 1990); available at http://www.ietf.org/rfc/rfc1155.txt?number=1155.

Specification for CIM Operations over HTTP Version 1.0, The Distributed Management Task Force, Inc. (August 11, 1999); available at http://www.dmtf.org/download/spec/xmls/CIM_HTTP_Mapping10.htm.

Specification for the Representation of CIM in XML Version 2.0, The Distributed Management Task Force, Inc. (July 20, 1999), available at http://www.dmtf.org/download/spec/xmls/CIM_XML_Mapping20.htm.

A. Tang and S. Scoggins, *Open Networking with OSI*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1992).

U. Warrier, L. Besaw, L. LaBarre, and B. Handspicker, *The Common Information Services and Protocols for the Internet (CMOT and CMIP)*, RFC1189 (October 1990); available at http://www.ietf.org/rfc/rfc1189.txt?number=1189.

Accepted for publication November 10, 2000.

Heather Kreger *IBM Software Solutions Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709-2195 (electronic mail: kreger@us.ibm.com).* Ms. Kreger represents IBM as an active, contributing member of the Java Management eXtensions (JSR0003) Expert Group. Her years in lead positions on the development teams of Automated Network Operations/MVS and Netview/AIX products combined with her development experience on the Lotus Domino™ Go Webserver and WebSphere Application Server products gives her unique insight into the problems of and solutions for managing e-business applications. Ms. Kreger has contributed to the specification, reference implementation, and compatibility test suites for Sun's JMX Reference Implementation. She is also involved in The Open Group Management Program, the Distributed Management Task Force Application Management and Interoperability work groups, and Sun's WBEM JSR (Java Specification Request) expert group.