Mario Francois Jauvin. “JMAPI - Java Management Application Programming Interface”
Handbook of Emerging Communications Technologies: The Next Decade.
Ed. Saba Zamir
Boca Raton: CRC Press LLC, 2000

4 JMAPI - Java Management Application Programming Interface

Mario Francois Jauvin

CONTENTS

4.1 WHAT IT IS

The Java Management Application Programming Interface (JMAPI) is a collection of Java classes, provided by Sun Microsystems, that allow network management software developers to write management applications using a standardized platform-independent application programming interface. The initial API was based on the Java Developer Kit (JDK) 1.0 API specification. JMAPI was subsequently modified to support the 1.1 JDK specification in order to make use of its security and the Remote Method Invocation (RMI).

JMAPI is not a network or system management product and by itself cannot provide any management functionality. Rather, it is designed to be used by developers of network management systems and also by network element hardware vendors. A look at the JMAPI architecture will clarify these concepts.

4.2 ARCHITECTURE

The architecture of JMAPI consists of the components at the highest-level as shown in Figure 4.1.

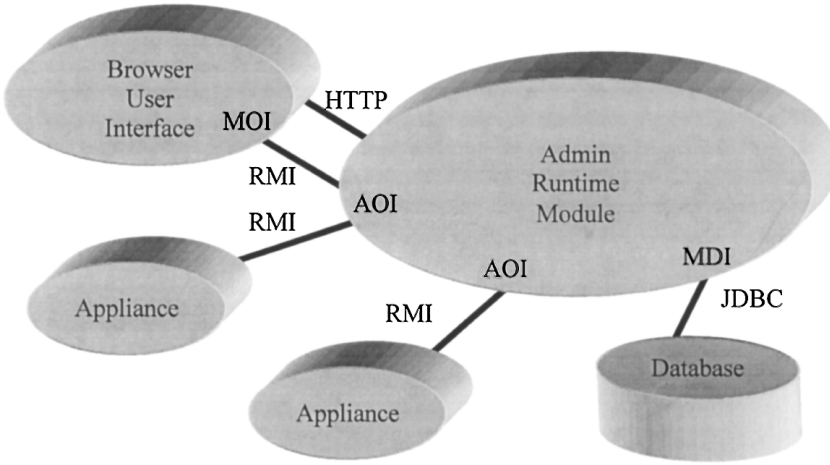

Figure 4.1 JMAPI Architecture

Everything except for the database portion runs in a Java environment.

4.2.1 BROWSER USER INTERFACE

The Browser User Interface (BUI) is somewhat a misnomer. The BUI actually is the JMAPI component from which an administrator can issue management operations or get management status from the objects being managed. The BUI can run either as an applet in a browser or as a standalone Java application with or without a graphical user interface. The BUI interacts entirely with one or more Admin Runtime Module and a HyperText Transfer Protocol (HTTP) server.

The BUI is comprised of the following modules:

- Admin View Module
- Managed Object Interfaces
- Java-enabled browser

4.2.1.1 Admin View Module

The Admin View Module (AVM) contains the key client-side classes for developers of JMAPI-applets whose primary role is to provide user interface and application-level functionality. The user interface is built completely on top of JDK Abstract Window Toolkit (AWT). The AVM classes are broken down into three packages: AVM Help to provide a general purpose help environment, AVM Base to provide AWT extensions for integrated management solution building and AVM Integration to provide integration between AVM Base classes and the Managed Object Interface.

4.2.1.2 Managed Object Interface

The Managed Object Interface (MOI) uses RMI to perform remote management of objects. A managed object is an abstraction of a resource (network element such as a hub or router, a service such as a DNS server, or anything else one may wish to manage) in the enterprise. Although the managed object is an abstraction of the resource, the actual resource can be managed from a JMAPI point of view only if it also implements a JMAPI appliance.

4.2.1.3 Java-enabled Browser

Any JDK 1.1 compliant browser, such as Netscape Navigator, HotJava, or Internet Explorer, can be used to run the AVM and MOI. If a Java application is used the browser will not be necessary.

4.2.2 Admin Runtime Module

The Admin Runtime Module (ARM) is the component that provides active instantiated managed objects for a number of appliances. The ARM is the component that will carry out the management operations requested by the BUI through the Managed Object Interface. The ARM and all of its associated functionality is referred to in JMAPI terminology as a long-running process called the JMAPI server or the managed object server. Please note that a BUI can communicate with more than one ARM, but ARMs do not communicate amongst them. Also note that one ARM can communicate with multiple appliances, and one appliance can communicate with multiple ARMs.

4.2.2.1 HTTP Server

The HTTP server is only used to load the applet, the AVM classes, and the network management application built on top of them into the browser. The benefits of this approach is that a new version of the network management application or JMAPI can automatically be acquired and software distribution and versioning are no longer an issue.

4.2.2.2 Managed Object Factory

The Managed Object Factory (MOF) is the JMAPI component responsible for creating instances of managed objects. Managed objects are RMI remote objects. The creation of a managed object is a two-step process. First, the new object is created by calling MOFactory.newObj(). Then the new object is made persistent by calling its addObject() method. This component is what actually constitutes the managed object server.

The MOF is also the component responsible for actually implementing the management operations requested by the BUI through interactions with Agent Objects Interfaces and Managed Data Interfaces.

4.2.2.2.1 Managed Data Interfaces

The Managed Data Interfaces (MDI) is responsible for keeping persistent information about managed objects using any commercial JDBC (Java Database Connectivity) compliant relational database management system.

4.2.2.2.2 Agent Object Interfaces

The Agent Object Interfaces (AOI) are the interface to agent objects residing on the appliances. These interfaces provide the MOF the ability to control and do the actual management of appliances or network resources.

4.2.2.2.3 Notification Dispatcher

The Notification Dispatcher (ND) is the JMAP component which filters and forwards events from the appliances managed by the ARM to the interested parties (BUI) that have registered for notification with the ARM.

4.2.3 Appliances

4.2.3.1 Agent Object Factory

As you may recall from earlier, a managed object instance in the ARM has no remote management capabilities unless it can communicate with a remote agent object running on the appliance. In order to support this remote management capability the appliance must implement the Agent Object Factory which creates and maintains Agent Object Instances.

4.2.3.2 Agent Object Instance

Each Agent Object Instance (which corresponds to RMI objects) will call Java classes or Native Methods (using Java Native Interface, or JNI) to implement the management operation. If necessary, the Class Loader will download the Java classes and the Library Loader will download the Native Methods.

4.3 SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) AND JMAPI

The importance of the SNMP protocol as the most prevalent management protocol was recognized and is integrated with JMAPI. Although it is possible for the BUI interface to directly interface with SNMP entities, the most useful application of the SNMP support provided by JMAPI is at the ARM or MOF level. In that context, SNMP is being used (instead of RMI) to communicate to SNMP agent entities (instead of JMAPI appliances).

4.4 JMAPI AND THE INDUSTRY

The potential for JMAPI lies in the fact that this technology can be used to manage a heterogeneous environment, which is more the norm than the exception in today's enterprises. Its proposal for a unified management API is a key contributor to being capable of managing systems of different operating systems, different hardware platforms, and different protocols. Last but not least, the portability inherent in Java imposes few restrictions on where the JMAPI management platform can run.

Before JMAPI can realize its full potential, vendors of hardware and software must embrace it and provide the necessary interfaces to implement appliances. In the case of network hardware vendors, this might not be a trivial task or feasible in the restricted hardware footprint where such implementation must reside. For example, a Java Virtual Machine would have to be implemented as well as management software interfaces to query or modify the operation of the device (to be made accessible to JMAPI via JNI). This is, of course, in addition to the necessary SNMP agent entity implementation.

The all-Java aspect of JMAPI is both a blessing and a curse. The fact that a single portable language is used throughout the environment means that development time and skills required are reduced. However, in the case of the ARM, which is meant to be a server type component, unacceptable performance is most likely to appear in medium-sized environments and almost guaranteed in large environments.

The JMAPI issue that has caused the most difficulties for people trying to evaluate or investigate the technology is its requirements for a commercial relational database management system (RDBMS, such as Sybase, Oracle, etc.). Some people have incorrectly interpreted this issue to mean that JMAPI requires a method of providing persistent storage for the states and attributes of managed objects. This is actually one of JMAPI's strong points. The real issue is that this requirement could not be achieved using a less costly and less administration-intensive solution than a commercial RDBMS. The change to support any JDBC-compliant RDBMS is one step in the right direction, but the fact that no recommendation using a non commercial JDBC-compliant solution exists from Sun makes this still a very resource-intensive approach.

4.5 WHAT'S AHEAD FOR JMAPI

Given that Sun has not modified JMAPI in the last six months it is difficult to assess where the technology will go in the future. According to Bruce Boardman[1] JMAPI "has a significant and growing list of supporters — including Bay, Cisco, IBM, Novell, PLATINUM Technologies, Sun, and 3Com." Paula Musich,[2] on the other hand, indicated in December of the same year that although Cisco and Bay had pledged their allegiance to JMAPI in May 1996, in December 1997 they had no products shipping based on it. To this date I am not aware of any such products either. The fact that the specification is not final probably has a lot to do with this. I think that a potential customer interested in using JMAPI is probably even more puzzled by the fact that Sun's own Solstice Enterprise Manager product comes with Java add-on technology called Java Dynamic Management Kit (JDMK) with no reference altogether in its product literature to JMAPI.

Bill Gates' (Microsoft CEO) statement[3] in April 1998 to the effect that he had not heard of the JMAPI acronym and Microsoft's direction to push for Web Based Enterprise Management (WBEM, a competing technology to JMAPI) are certainly not good news. The industry would not be served by two new management standards when it is looking for a unifying platform-independent solution.

REFERENCES

1. Bruce Boardman, "The Dawning Of The Age Of Java Management," *Network Computing*, May 15, 1997.
2. Paula Musich, "Sun Rises — Finally," *PC Week*, December 5, 1997.
3. Steven Burke, "Gates Takes Jab At Java Network Management," *Computer News Resellers*, April 27, 1998.
4. Luca Deri, "Rapid Network Management Application Development," IBM Zurich Research Laboratory, University of Berne.
5. Cameron Sturdevant, "Room for growth," *PC Week Labs*, July 14, 1997.
6. Paula Musich, "Web management specs inch forward," *PC Week*, July 11, 1997.
7. Paula Musich, "Next Java jolt: Management," *PC Week*, July 11, 1997.
8. Paula Musich, "CA announces Solaris version of Unicenter TNG," *PC Week Online*, July 16, 1997.
9. Paula Musich, "The Golden Age of Enterprise Systems," *PC Week Online*, October 6, 1997.
10. *Java Management API Architecture*, SunSoft, September 1996.
11. *Java Management Programmer's Guide*, SunSoft, September 1997.
12. *Java Management API, Help Developers' Guide*, SunSoft, May 1997.
13. *Java Management API User Interface Visual Design Style Guide*, SunSoft, September 1997.
14. *Java Management API User Interface Style Guide*, SunSoft, May 1997.