Chapter 2

History of Manageability

Civilizations in decline are consistently characterized by a tendency towards standardization and uniformity

— Arnold Toynbee (1889–1975)

Manageability, as a unique discipline, has historically evolved from the growing need to configure and maintain the computer systems, applications, and networks. As more and more of these entities provided capabilities that could be adapted, changed, and optimized for a particular use or preference, the need for manageability grew. With a low number of systems, it was possible to have system administrators individually log into the system locally and manage each system independently. However, as these systems grew in number and became more and more complex, administrators needed to manage a large number of resources from a central management site. The use of remote management tools became important. This led to development of several protocols for remote management. In the beginning, most of these were proprietary in nature, which meant that a system from one manufacturer could only be managed with a management console from the same manufacturer. This led the industry to work on interoperable standards that allow the systems from multiple manufactures to be managed with common tools.

Protocol and Data Model

Remote management interfaces can be logically viewed as a combination of a communication protocol and the payload that is exchanged via that communication protocol.

Historically, this separation has not always been very clear or emphasized, and some standards have treated them with a very tight binding. However, making this logical separation has clear advantages as illustrated in Figure 2.1.

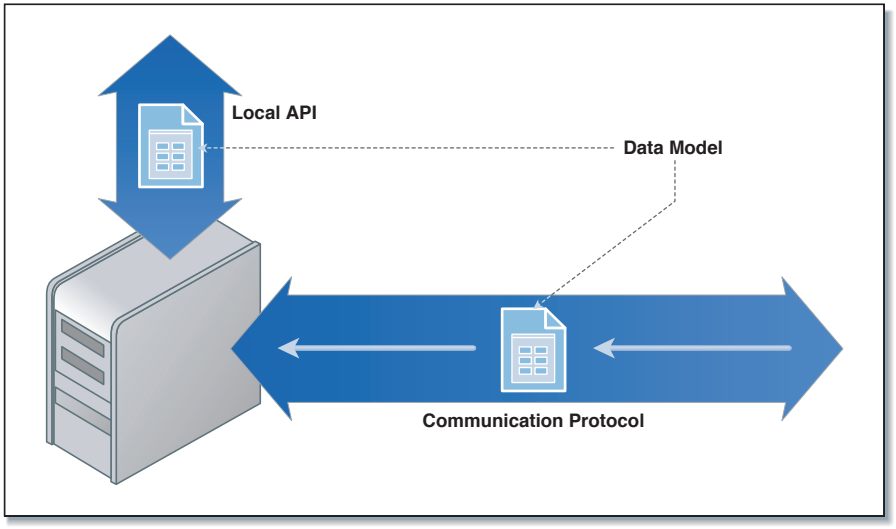

Figure 2.1 Separation of Communication Protocol and Data Model

The communication protocol defines how the messages from one system are to be encoded and sent to another system. It does not depend on the contents of the messages. In much the same way that TCP/IP as a protocol allows a reliable transmission of the packets from one network node to another and does not concern itself with the content of the packets, a good management protocol is agnostic of the management payload or the data model.

The data model defines the actual content that allows the request for specific changes to be made to the managed entity and communicated via the communication protocol to the managed system.

The separation of the two also allows the designer of the system to provide different access mechanisms while keeping the same management data model. For example, a local application programming interface (API) can provide an alternate mechanism for management while keeping the same semantics for the data model.

Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is a management standard that came into existence in late 1980s and achieved widespread acceptance over the next decade. A majority of network devices, routers, switches, and gateways have been using SNMP as the standard management protocol.

The name SNMP suggests that it is only a network protocol. However, this standard defines more than a protocol. SNMP is based on the manager/agent model, consisting of the manager, agent, a database of information, managed objects, and the network protocol.

The manager and the agent communicate use a Management Information Base (MIB) and a set of well defined commands to exchange the information. MIBs are organized in a tree structure with each MIB given its unique place in the tree. Within an MIB, eventually the actual management information is defined through MIB variables. Each individual MIB variable is identified via a unique identifier, called an Object Identifier (OID).

The OIDs are assigned based on where the MIB appears in the MIB tree, and how the variables are defined within the MIB. For example, a standard MIB called mib-2 is defined by `iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1)`. Based on this structure, all variables in mib-2 will have a prefix of 1.3.6.1.2.1. Within this mib-2, the complete OID of the variable `sysDescr` (as shown in the following excerpt of mib-2) is 1.3.6.1.2.1.1.3.

```
mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }
system     OBJECT IDENTIFIER ::= { mib-2 1 }
sysDescr OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "A textual description of the entity.  This value
            should include the full name and version
            identification of the system's hardware type,
            software operating-system, and networking
            software.  It is mandatory that this only contain
            printable ASCII characters."
    ::= { system 1 }
```

SNMP uses five basic messages (GET, GET-NEXT, GET-RESPONSE, SET and TRAP) to communicate between the manager and the agent, as illustrated in Figure 2.2. GET and GET-NEXT messages allow the manager to request information for a specific variable. The agent receiving these messages then uses a GET-RESPONSE message to provide the information requested, or an error indication if the request cannot be processed. A SET message allows the manager to change the value of the variable, and thus change configuration data or control a particular object, such as disabling a network interface. The agent uses the GET-RESPONSE message to indicate the changed value or the error condition. There is no specific SET-RESPONSE message, since the content of the messages is exactly same as in case of GET. The TRAP message allows the agent to spontaneously inform the manager of a critical event.

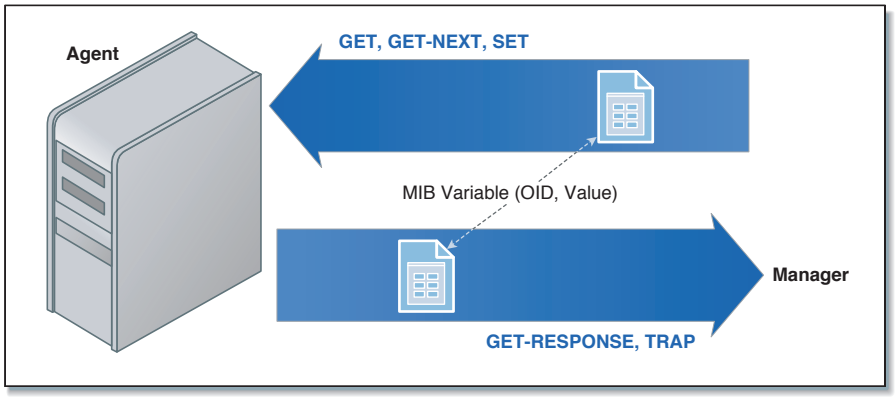

Figure 2.2 SNMP Protocol Messages

The SNMP protocol is built on top of UDP/IP, and uses a simple connectionless mechanism. The protocol is a basic query/response protocol. TRAPs are the only packets that are sent by the agent without a request from the manager. There is no acknowledgement defined for the TRAP messages, so TRAPs are not necessarily guaranteed to be received by the manager. Managers often periodically poll the agent to receive information, in case the TRAP message is missed.

The small number of commands, simplicity, and therefore ease of use led to the widespread usage of the protocol. However, the same simplicity is holding back further progress of SNMP into the Internet and Web era. The first version of SNMP had next to no security (a simple plain text community string as a password). SNMPv2 and SNMPv3 enhanced the security but lost the simplicity, and hence were not readily adopted. Even with what is available in these versions, it is no match for the security and reliability demands of today.

Newer web-based standards, as described later in this chapter, are slowly replacing SNMP deployments. No new standards work is going on for enhancement of SNMP.

Desktop Management Interface

SNMP was widely adopted in the networking segment. However, as people started to look into managing computer systems, components within a system, operating system components, and applications parameters, it became clear that SNMP was not designed to do this. A local API standard was needed. A network-based protocol was not ideal for local management by application, and the OID structure defined by SNMP was very rigid and cumbersome to manage thousands of variables in the system. In 1994, several system vendors including Intel, Dell, HP, IBM, and Compaq formed a standards consortium under the name of Desktop Management Task Force (which was later renamed to Distributed Management Task Force) to embark on the task of defining such a standard. The Desktop Management Interface (DMI) was a result of that effort.

DMI defines the data model in a much simpler format using Management Information Format (MIF). The MIF uses a more text-based definition that can be parsed by a machine as well as easily read by humans. The information is organized into groups of attributes (equivalent to SNMP variables). The groups are uniquely identified based on their Group ID (a string). Any number of groups can be combined together and organized in a MIF file. The flexibility of DMI makes it very easy to organize MIF groups in any arbitrary collection based on a specific implementation. So, if an implementation only needs one group, no other groups need be implemented. Group attributes have a name, description, type, read/write properties, and other information that gives enough guidance to a management application to interpret the attribute value.

A sample MIF segment containing a single group is shown here.

```
Start Group
      Name = "ComponentID"
      Class = "DMTF|ComponentID|001"
      ID = 1
      Description = "This group defines the attributes "
        "common to all components.  This group is required."
      Pragma = "SNMP:1.3.6.1.4.1.412.2.1.1 ;"

      Start Attribute
            Name = "Serial Number"
            ID = 4
            Description = "Serial number for this system."
            Access = Read-Only
            Storage = Specific
            Type = DisplayString(64)
            Value = ""
      End Attribute
      // Additional Attributes Deleted from illustration //
End Group
```

DMI operations are modeled along the same lines as SNMP. However, instead of defining a protocol and on-the-wire messages, DMI defines an API-style interface, which is much simpler and readily usable by software programs.

The DMI API defines calls to enumerate (list) DMI Components, Groups, Attributes, and Class names. This provides a discovery mechanism to find out what exists. Once the management application has discovered that a specific group is instrumented, then it can call DmiGetAttribute or DmiSetAttribute to operate on the values of the attributes in the group. Some groups can have multiple instances, like rows in a table. For such groups, the API provides add and delete row operations to allow manipulations of the group instances. A mechanism to register for a callback in case of an event is also specified.

In addition to the management API, DMI also defines a pluggable manager and provider architecture. As shown in Figure 2.3, this centers around a DMI Service Provider (SP) as the manager of the information, a Component Interface (CI) for registering one or more components (also called instrumentation) providing information, and a Management Interface (MI) for management applications. This provides a very modular architecture, where multiple vendors can plug in their instrumentation and multiple management applications can use that information.

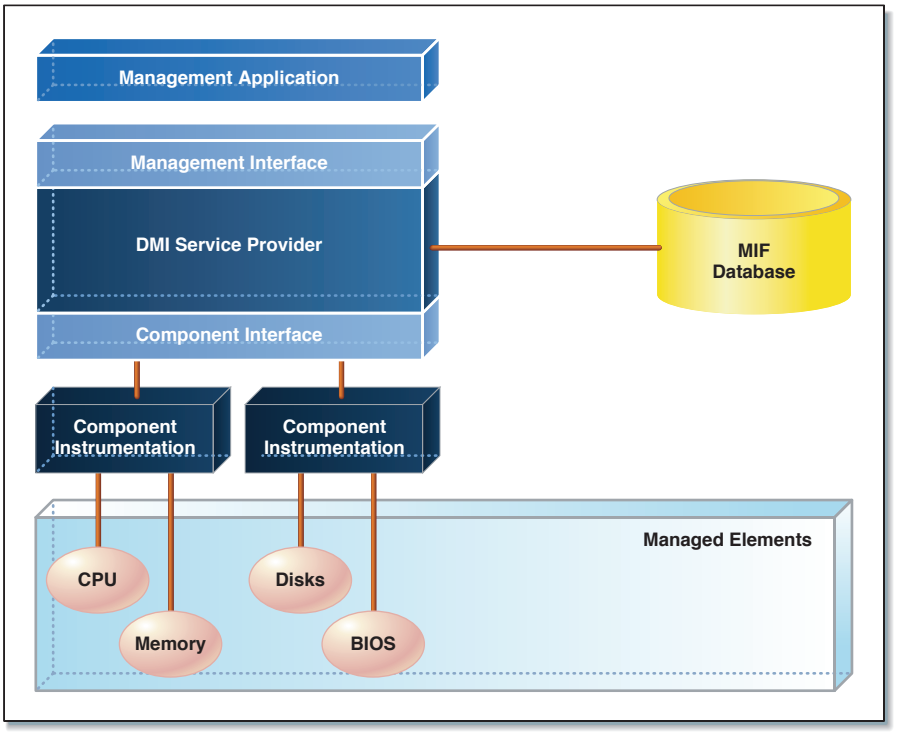

Figure 2.3 DMI Service Provider Infrastructure

This structured format came as a benefit to system vendors, who could take components and instrumentations from their component suppliers and integrate all of it into a cohesive solution. Several hardware and computer system vendors developed instrumentations and built solutions based on DMI infrastructure.

However, DMI's strong definition of infrastructure and interfaces was not warmly welcomed by Microsoft, as it was not aligned with the rest of the Windows management infrastructure. Microsoft started working on an alternate architecture, Windows Management Instrumentation (WMI), which then started the foundation of next generation of DMTF standard, as described later in this chapter.

Wired for Management

As DMTF was defining the DMI standards on standardizing the management information format and the infrastructure to allow consistent management of multiple components in the platform, Intel started working with the system vendors and component vendors to define a baseline of management information that a system must present. Wired for Management (WFM) defined such a baseline. The baseline was defined in terms of the DMI groups and attributes that must be instrumented to meet WFM compliance.

In addition, mechanisms and structures were also defined (with the effort led by BIOS vendors) for the BIOS to collect the information from the system and put it in BIOS structures. This was defined by the SMBIOS (System Management BIOS) specification, later standardized by DMTF. SMBIOS structures also follow the data defined by DMI groups and attributes definitions. However, since BIOS is under a space constraint, the structure uses lower level bits and bytes definitions.

Another technology that came out of the WFM effort was the Preboot eXecution Environment (PXE) specifications. The PXE environment allowed the remote boot of a computer system from a network image. This is often used today in a variety of enterprise environments to do a "bare-metal" provisioning. Bare metal here refers to a system that is fresh out of the box and has not yet been provisioned with an operating system. PXE allows the system to be booted from a remote image, which can then in turn install the operating system. PXE can also be used for diskless systems to always boot from a network image.

PXE is being used today in enterprise environments, but is not scalable and lacks security. So, it is unlikely that PXE usages will grow much unless the scalability and security issues are resolved. Later in the book, we discuss some alternate mechanisms that address these issues.

Intelligent Platform Management Interface

As standards were being developed by DMTF and others to define mechanisms for management applications to manage systems (that is, external view), the server system vendors were facing another challenge. This was to look inside the system and define efficient ways of combining hardware

components from multiple vendors and provide a way of collecting the management information from these hardware components inside the platform (system) via some standard management bus. Intel, Dell, HP, and NEC led the creation of the Intelligent Platform Management Interface (IPMI) to address this need. IPMI defines standardized, abstracted interfaces to the platform management subsystem. IPMI includes the definition of interfaces for extending platform management between boards within the main chassis, and between multiple chassis.

The term *platform management* is used to refer to the monitoring and control functions that are built in to the platform hardware and primarily used for the purpose of monitoring the health of the system's hardware. This typically includes monitoring elements such as system temperatures, voltages, fans, power supplies, bus errors, system physical security, and so on. It includes automatic and manual recovery capabilities such as local or remote system resets and power on/off operations. It includes the logging of abnormal or out-of-range conditions for later examination and alerting where the platform issues the alert without aid of runtime software. It also includes inventory information that can help identify a failed hardware unit.

IPMI defines an Intelligent Platform Management Bus (IPMB), which is an I^2C-based bus that provides a standardized interconnection between different boards within a chassis. The IPMB can also serve as a standardized interface for auxiliary or emergency management add-in cards.

IPMI also specifies an Intelligent Chassis Management Bus (ICMB), which provides a standardized interface for platform management information and control between chassis.

IPMI was developed as a complementary technology to provide low-level management information to broader frameworks based on SNMP, DMI, and CIM (discussed later in this chapter). However, since it provided information about hardware, and the information needed to be made available to remote management consoles in OS-absent scenarios, it did define a simple UDP based Remote Management Control Protocol (RMCP) to send IPMI messages to a remote system. It also based its Platform Event Trap Format definition on SNMP traps, which provided a mechanism to send asynchronous alerts to management consoles.

Alert Standard Format

While Server vendors were busy solving the platform management problem and building a modular and scalable framework with IPMI, client vendors, led by Intel and IBM, had been working on Alert on LAN (AOL) technology, which initially focused on providing OS-independent alerting mechanisms, for events like OS failures, from the LAN devices directly to the management consoles. This technology was later standardized as Alert Standard Format (ASF), and submitted to DMTF in due course. IPMI (for servers) and ASF (for clients) continued to evolve together, sometimes sharing technologies, and at other times developing parallel technologies. ASF adopted Platform Event Trap Format defined by IPMI, and IPMI adopted Remote Management and Control Protocol (RMCP) defined by ASF. However, the actual messages under the covers of these common protocols (RMCP and PET) are quite different between ASF and IPMI.

For inside the platform interfaces, ASF also defined System Management Bus (SMBus) for connecting a small number of sensors. It is also an I^2C-based bus, but is not as extensible as IPMB. In fact, IPMB comprehends connecting to SMBus based sensors, and is thus a superset.

Common Information Model

SNMP was focused on managing the network devices. DMI was created to manage components in a platform from host OS-based applications. IPMI and ASF were more focused on inside the platform as well as out-of-band and OS-absent management. Software, applications, and services didn't really have any widespread standard for management. All this, coupled with the need to have an end-to-end management of all infrastructure, led to the concepts of a Common Information Model (CIM), unifying all the previous management models.

The need for end-to-end management, across multiple components, in a distributed environment is a reality and is now a requirement. It is no longer sufficient to manage personal computers, servers, subnets, the network core, storage, and software in isolation. These components all interoperate to provide connectivity and services. Information passes between these boundaries. Management must pass across these boundaries as well.

These are the problems addressed by the Common Information Model. The goals of CIM are to address both FCAPS management (fault, configuration, accounting, performance, and security management) and to support the abstraction and decomposition of services and functionality. The information model defines and organizes common and consistent semantics for computing and networking equipment and services. The model's organization is based on an object-oriented paradigm, promoting the use of inheritance, relationships, abstraction, and encapsulation to improve the quality and consistency of management data.

The value of CIM stems from its object orientation. Object-oriented design provides support for the following capabilities that other "flat" data formats do not allow.

Abstraction and Classification

To reduce the complexity of the problem domain, high level and fundamental concepts (the "objects" of the management domain) are defined. These objects are then grouped into types ("classes") by identifying common characteristics and features ("properties"), relationships ("associations") and behavior ("methods").

Object Inheritance

By creating subclasses from the high level and fundamental objects, additional detail can be provided. When created, a subclass "inherits" all the information (properties, methods, and associations) defined for its higher level objects. Subclasses are created to put the right level of detail and complexity at the appropriate level in the model. This can be visualized as a triangle, where the top of the triangle is a "fundamental" object, and more detail and more classes are defined as you move closer to the base.

Ability to Depict Dependencies, Component and Connection Associations

Relationships between objects are extremely powerful concepts. Before CIM, management standards captured relationships in multidimensional arrays or cross-referenced data tables. The object paradigm offers a more elegant approach in that the relationships and associations are directly modeled. In addition, the way that relationships are named and defined describes the semantics of the object associations. Further semantics and information can be provided in properties (specifying common characteristics and features) of the associations.

Standard, Inheritable Methods

The ability to define standard object behavior (methods) is another form of abstraction. Bundling standard methods with an object's data is called *encapsulation*. Imagine the flexibility and possibilities of a standard able to invoke a Reset method against a hung device, regardless of the hardware, operating system, or device.

Summary

In this chapter we reviewed various manageability standards and technologies from a historical perspective. The next chapter provides details on the current state-of-the-art management technologies, such as CIM and Web Services–based management, which are gaining widespread acceptance.