Chapter 3

Manageability Standards

The nice thing about standards is that there are so many of them to choose from.

—Andrew S. Tanenbaum

Fortunately, bringing some relief to IT managers and system administrators, the industry is starting to converge on a common set of standards for management of applications, systems, and platforms. Common Information Model (CIM) for representation of various manageable entities and Web-based access to this model is fast becoming a prevalent standard. This chapter discusses these current and upcoming standards.

Common Information Model (CIM)

Common Information Model is a standard that started in DMTF in 1997. During the past 10 years, the CIM standards have grown to cover a wide variety of manageable entities, including systems, network, storage, applications, and services.

CIM is an abstraction and representation of the entities in a managed environment, their properties, attributes, operations, and the way that they relate to each other. It is independent of any specific repository, software usage, protocol, or platform.

Chapter 2 provided a high level overview of CIM. In this chapter we explore in more detail how classes are represented and defined in CIM.

UML Diagram

Figure 3.1 shows an example of CIM classes. The classes are represented in a UML (Unified Modeling Language) diagram.

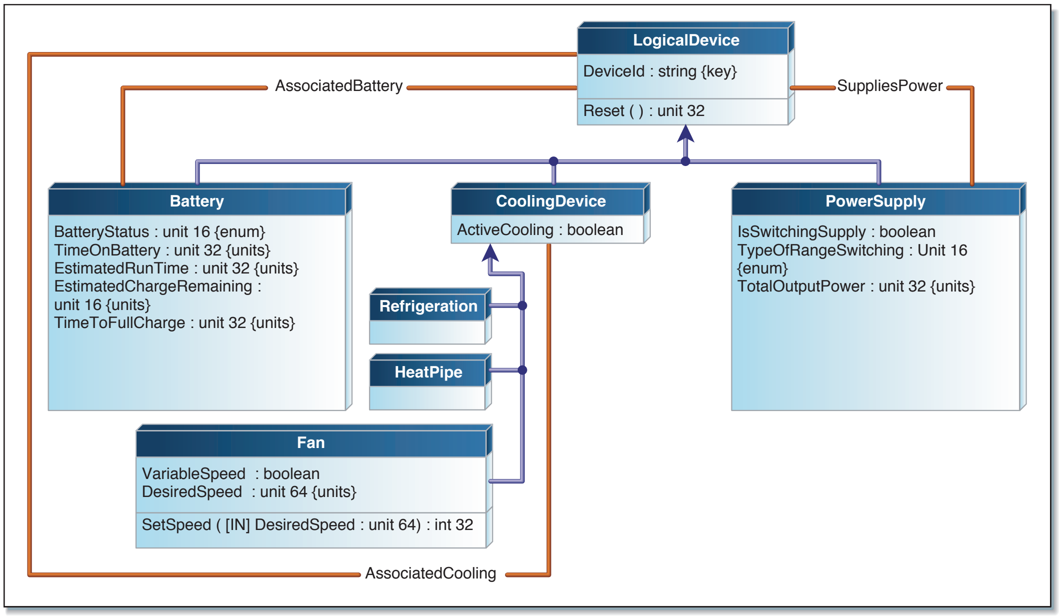

LogicalDevice
DeviceId : string {key}
Reset ( ) : unit 32
AssociatedBattery
SuppliesPower
Battery
BatteryStatus : unit 16 {enum}
TimeOnBattery : unit 32 {units}
EstimatedRunTime : unit 32 {units}
EstimatedChargeRemaining : unit 16 {units}
TimeToFullCharge : unit 32 {units}
CoolingDevice
ActiveCooling : boolean
PowerSupply
IsSwitchingSupply : boolean
TypeOfRangeSwitching : Unit 16 {enum}
TotalOutputPower : unit 32 {units}
Refrigeration
HeatPipe
Fan
VariableSpeed : boolean
DesiredSpeed : unit 64 {units}
SetSpeed ([IN] DesiredSpeed : unit 64) : int 32
AssociatedCooling

Figure 3.1 Example of CIM Classes

The example shows a LogicalDevice, which is an abstract class that represents a wide variety of devices that may be present in a platform. Battery, CoolingDevice, and PowerSuply are further specialization of the LogicalDevice. These devices are inherited (derived) from LogicalDevice as shown by the blue arrows. Additional specific properties are added to these classes. When these classes are instantiated as objects, they represent the respective specialized devices. At the same time, they inherit the properties of the parent class, and hence are LogicalDevice objects as well. All of the

LogicalDevice behavior is common across all these objects. For example, all the logical devices can be reset by invoking the Reset() method of the corresponding object representation.

The red lines in the diagram represent Association objects. These Associations connect two unrelated object together with certain behavior. In the example shown, the AssociatedCooling association helps us figure out which cooling device is cooling a particular LogicalDevice. This can be used to differentiate a system fan from a processor fan. If one of these fans fails, a management console can quickly isolate the component impacted because of this failure.

Managed Object Format (MOF)

Although UML diagrams are nice to get an overview of the overall class hierarchy and associations, it is not a formal machine-readable representation as defined by DMTF.

Common Information Model (CIM) Infrastructure Specification defines a formal language to describe the CIM classes and objects. This is called Managed Object Format (MOF). Complete MOF syntax and grammar rules are defined in the CIM specification.

The following example shows an excerpt from the definition of a CIM class (CIM_Sensor) in MOF syntax.

```
//==================================================
// Sensor
//==================================================
   [Abstract, Version ( “2.6.0” ), Description (
       “A Sensor is a hardware device capable of measuring the “
       “characteristics of some physical property - for example,
the “
       “System.”)]
class CIM_Sensor : CIM_LogicalDevice {
      [Description (
          “The Type of the Sensor, e.g. Voltage Sensor. “
          “description of the different Sensor types is as fol-
lows:  “
          “………………………deleted   text… “.”),
       ValueMap { “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”,
“9”,
          “10”, “11”, “12” },
       Values { “Unknown”, “Other”, “Temperature”, “Voltage”,
          "Current", "Tachometer", "Counter", "Switch", "Lock",
          "Humidity", "Smoke Detection", "Presence", "Air Flow"
},
       ModelCorrespondence { "CIM_Sensor.
OtherSensorTypeDescription" }]
   uint16 SensorType;

      [Description (
          "PossibleStates enumerates the outputs of the Sensor.
"
          "For example, a \"Switch\" Sensor may output the
states "
          "\"On\", or \"Off\". Another implementation of the
Switch "
          "may output the states \"Open\", and \"Close\".
Another "
          "example is a NumericSensor supporting thresholds.
This "
          "Sensor can report the states like \"Normal\", \"Upper
"
          "Fatal\", \"Non-Critical\", etc. A NumericSensor that
"
          "does not publish readings and thresholds, but stores
this "
          "data internally, can still report its states."),
       MaxLen ( 128 )]
   string PossibleStates[];

      [Description (
          "The current state indicated by the Sensor. This is
always "
          "one of the \"PossibleStates\"."),
       MaxLen ( 128 )]
   string CurrentState;

};
```

Let us discuss a few key components of this definition. CIM_Sensor is the main *class* defined in this example. This class inherits from a parent class called CIM_LogicalDevice. Class level *qualifiers* provide more information about the class. For example, the qualifier Abstract defines that the class cannot be directly instantiated into an object, but an implementation must further derive a Concrete class to instantiate an object. The CIM_sensor class shown in the example MOF segment lists the properties SensorType, PossibleStates[] and CurrentState. The actual class definition has more properties, but these are omitted from this illustration. SensorType is an unsigned integer, PossibleStates[] is an array of strings, and CurrentState is of the type string. Property level *qualifiers* provide additional information about the properties. For example, *ValueMap* and *Values* arrays provide enumeration values to the SensorType property. This allows applications to understand that if, for example, SensorType = 3, then the sensor is a "Voltage" sensor. Another property level qualifier *MaxLen(128)* restricts the length of the string property CurrentState to 128 bytes.

CIM Object Manager (CIMOM)

Let's delve a little bit into an implementation of the CIM infrastructure. Figure 3.2 shows a conceptual architectural diagram of a CIM-based infrastructure.

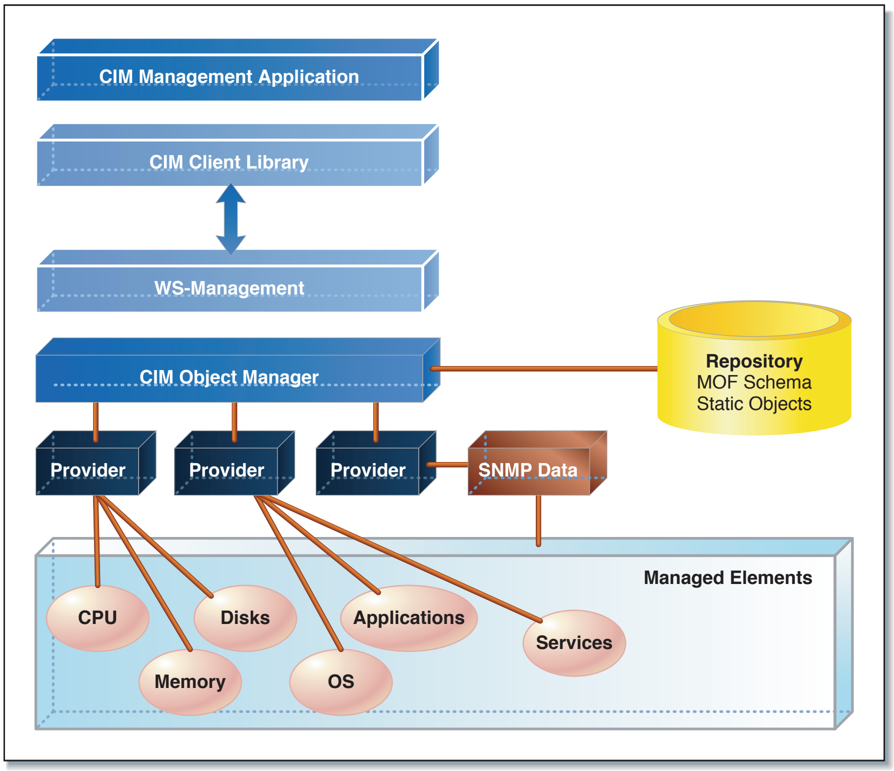


Figure 3.2 A CIM-based Infrastructure

In a typical implementation of CIM infrastructure, a CIM Object Manager (CIMOM) provides the most of the object management functions. The CIMOM provides capability to compile the MOF or XML Schema files and store them into a database. It allows providers to register to provide object instantiations of the CIM classes. The providers are typically controlling a managed resource, such as a disk drive, a networking stack, an application, or a service. These providers register with CIMOM to provide a CIM representation of the managed resource. When an application wants to read specific information, such as "Disk Capacity," the CIMOM routes the call to the appropriate Disk provider. The Provider translates this call to a resource specific interface, and in this case, sends the requests to the disk driver.

Requests to change an object property are handled in the same way. The infrastructure also allows for a resource to send asynchronous event to the provider, which in turn allows the CIMOM to send it to the applications that are interested in receiving this event.

The CIMOM implementations typically provide a network protocol interface for the management applications to communicate with the CIMOM. Web Based Enterprise Management (WBEM) defines this layer of communication. We will discuss this in more detail later in this chapter.

CIM Server

Not all the designs require a full CIMOM implementation. An external management console communicates to a CIMOM via the network protocol interface. The management console does not know (or care) if the object manager it is talking to provides underlying pluggable architecture with provider interfaces. In fact, a simpler CIM server may be able to satisfy the requests from the management consoles just as well as a full CIMOM. There is no complete definition of a CIM server that can be as highly optimized as the need dictates. The only requirement is that it must support a network protocol and retrieval of CIM object instances. Internally, an implementation could just store all data in a few variables, and return values when queried. A CIM server may not be able to parse MOF files, implement object inheritance, or support elaborate queries. Most of the embedded implementations just implement a CIM server, and not a full CIM object manager.

Management Profiles

DMTF has spent a considerable amount of time in providing comprehensive definitions of almost every aspect of platform management. There are CIM definitions for hardware components, disk, network, operating systems, applications, services, and security to name a few. Each of these areas has a number of CIM classes defined. At present, over 2000 CIM classes are defined. Implementations can chose to instrument the classes and properties that are important for the resources they need to manage. Since CIM classes do not mandate that a particular class or a property be implemented, it becomes hard for a management application to manage resources in a consistent way across the network. Furthermore, CIM as a data model does not specify the behavior of the system as a whole when a property value is changed or a method is called. This has led DMTF to further create *profiles* that define this behavior.

A *profile* is a specification that defines the CIM model and associated behavior for a management domain. The management domain is a set of related management tasks. For example, a server system may have a set of redundant power supplies that work together to provide power to the system. These power supplies are organized in a power domain, and a management console can query if the power domain is healthy and all the power supplies are active, and it can register for an event if one of the power supplies fails and the redundancy is lost. Such a behavior is specified in a power supply profile. Similarly, a mobile laptop system may have an instrumented battery. The management console can query all the instrumented laptops for their expected battery life. Such a behavior is documented in a system battery profile.

A *profile* contains the definition of a set of mandatory, as well as recommended classes, properties, methods and events. A *profile* also specifies the behavior of the system when some of these parameters are changed.

DMTF has two initiatives, System Management Architecture for Server Hardware (*SMASH*) and Desktop/Mobile Architecture for System Hardware (*DASH*), that have defined a number of profiles that are applicable for systems hardware management, which is the focus of this book.

DASH and SMASH have many profiles in common, while a few profiles are specific to each domain. Following are some of the examples of the profiles that are defined by DASH and SMASH.

- Power supply profile
- OS status profile
- Media redirection profile
- Platform watchdog profile
- Sensor profile
- PCI device profile
- LED profile
- KVM redirection profile
- BIOS management profile
- Alarm device profile
- Battery profile

More details on these profiles can be obtained from the DMTF Web site.

Web-Based Enterprise Management (WBEM)

Web-Based Enterprise Management (WBEM) provides the ability to exchange CIM information in an interoperable and efficient manner. WBEM includes protocols, query languages, discovery mechanisms, mappings, and anything else needed to exchange CIM information.

Today, WBEM defines three protocols for the communication between a management console and the CIM infrastructure.

CIM-XML over HTTP was the very first protocol defined by DMTF as a way to transport CIM objects over the network. Some implementations, particularly in storage industry, use this protocol heavily.

Windows Management Instrumentation (WMI), Microsoft's CIM implementation, had used a DCOM based remote network interface to communicate with the network console. Although not defined by DMTF, this has been commonly used.

DMTF has now published the new WS-Management protocol, which is composed using the latest Web services–based standards. We feel this protocol is likely to become the dominant standard for the network communications with a CIM infrastructure.

In addition to the above programmatic web-based protocols, DMTF also has defined an interactive Command Line Protocol (SMASH-CLP). This allows the network operators to use a simple text based interface to communicate with a backend CIM infrastructure.

WS-Management

The WS-Management architecture is based on a suite of specifications that define rich functions from which designs may be composed to meet varied service requirements. It is based on general SOAP-based Web services protocols, as illustrated in Figure 3.3.

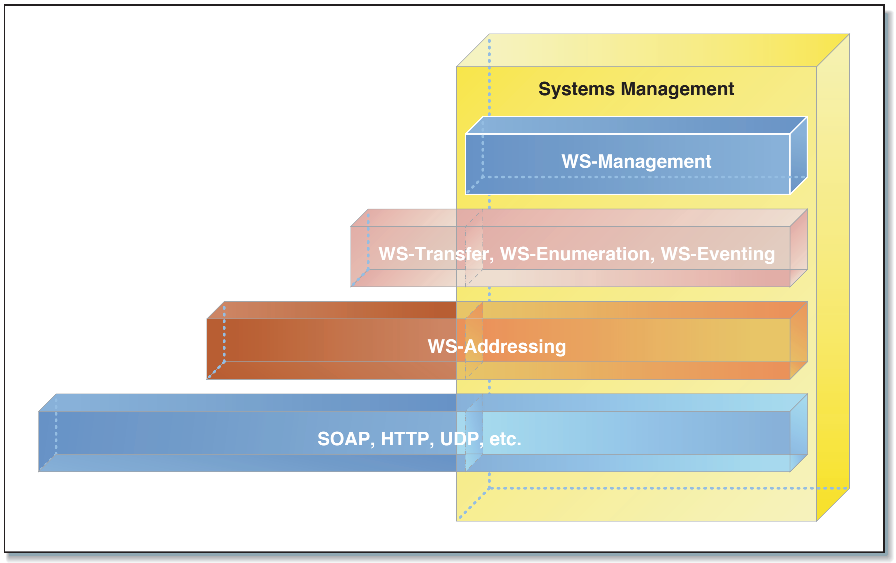


Figure 3.3 Systems Management with the WS-Management Protocol

To promote interoperability between management applications and managed resources, this specification identifies a core set of web service specifications and usage requirements to expose a common set of operations that are central to all systems management. This comprises the abilities to

- Discover the presence of management resources and navigate between them.
- Get, Put, Create, and Delete individual management resources, such as settings and dynamic values.
- Enumerate the contents of containers and collections, such as large tables and logs.
- Subscribe to events emitted by managed resources.
- Execute specific management methods with strongly typed input and output parameters.

In each of these areas of scope, the WS-Management specification defines minimal implementation requirements for conformant web service implementations. An implementation is free to extend beyond this set of operations, and may also choose not to support one or more areas of functionality listed above if that functionality is not appropriate to the target device or system.

As shown in Figure 3.3, the WS-Management specification uses underlying Web Services specifications, namely WS-Transfer, WS-Enumeration, and WS-Eventing. These are also sometimes referred to as TEEN (Transfer, Eventing, and Enumeration) specifications. These specifications are originally defined by World Wide Web Consortium (W3C, www.w3.org), and are evolving. On the other hand, the WS-Management specification is owned by DMTF, and is under a different release cycle. To minimize ever-changing dependencies, DMTF picked a version and included it in the WS-Management specification. Console vendors must be careful if using the W3C version of the TEEN specifications. Some of the changes in W3C TEEN specifications are not backward compatible with WS-Management.

The following example illustrates the WS-Management messages in a more detail. In this example scenario, the management console starts with querying the WS-Management server and enumerating all of the management objects it supports. Once it enumerates the objects, it makes further query to get the object and see the property values.

Figure 3.4 will help in understanding the protocol flow.

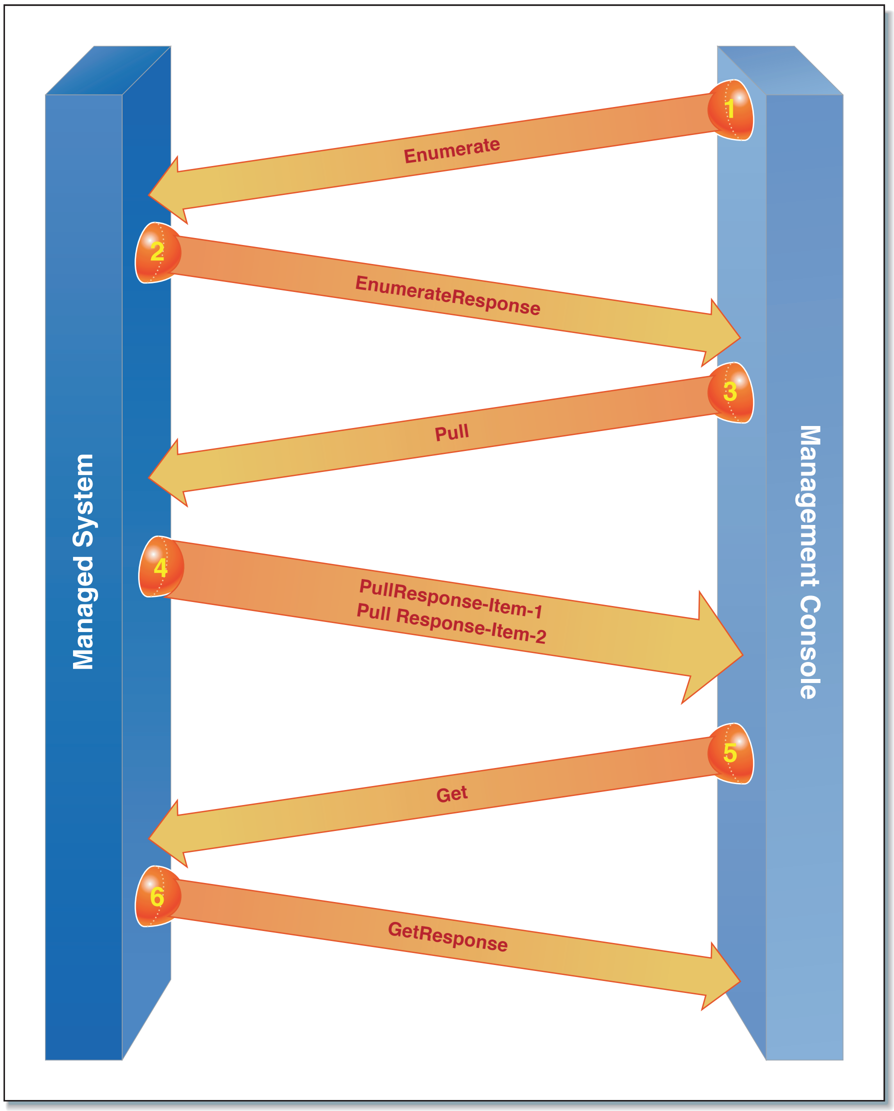


Figure 3.4 WS-Management Protocol Flow

Let's examine these messages in more detail.

Step 1: Management console sends the Enumerate request to the managed system. The managed system is hosting the WS-Management service. The message directed to a specific ResourceURI (`http:……/PhysicalElements`) as shown in the `<To>` tag of Figure 3.5. The `<Action>` specifies that this is an Enumerate request.

```
<Envelope>
  <Header>
    <To>
      http://192.168.0.100/agent?ResourceURI=
       http://schemas.dmtf.org/wbem/wscim/1/
          cim-schema/2+/CIM_PhysicalElement
    </To>
    <Action>
      http://schemas.xmlsoap.org/ws/2004/09/
        enumeration/Enumerate
    </Action>
    <MessageId>
      uuid:1778973d-10e9-477D-ae07-34e424e6577a
    </MessageId>
    <ReplyTo>
      http://schemas.xmlsoap.org/ws/2004/08/
        addressing/role/anonymous
    </ReplyTo>
  </Header>
  <Body>
    <Enumerate>
      <Expires>expiry time</Expires>
    </Enumerate>
  </Body>
</Envelope>
```

Figure 3.5 WS-Management Enumerate Request

Step 2: The response from the WS-Management service to the above Enumerate request is shown in Figure 3.6. This response starts the enumeration of all PhysicalElement objects provided by the WS-Management service for this targeted URI. This response primarily contains an EnumerationContext, which then can be used to walk through the elements in the list. Think of this as a file handle that is used in remaining calls to read the contents of the file.

```
<Envelope>
  <Header>
    <To>
      http://schemas.xmlsoap.org/ws/2004/08/
        addressing/role/anonymous
    </To>
    <Action>
      http://schemas.xmlsoap.org/ws/2004/09/
        enumeration/EnumerateResponse
    </Action>
    <RelatesTo>
      uuid:1778973d-10e9-477D-ae07-34e424e6577a
    </RelatesTo>
    <MessageID>
      uuid:dc0eeb8f-d025-4A45-a859-2b4ca640a1ff
    </MessageID>
  </Header>
  <Body>
    <EnumerateResponse>
      <EmumerationContext>0</EmumerationContext>
    </EnumerateResponse>
  </Body>
</Envelope>
```

Figure 3.6 WS-Management Enumerate Response

Step 3: Now that the EnumerationContext is known, the management console can issue the request to start the walkthrough of the list as shown in Figure 3.7. It starts by sending the request Enumeration/Pull with the EnuerationContext provided in the previous response. Management console can further specify how many maximum elements it wants to receive in the reply.

```
<Envelope>
  <Header>
    <To>
      http://134.134.201.169/agent?ResourceURI=
        http://schemas.dmtf.org/wbem/wscim/1/
          cim-schema/2+/CIM_PhysicalElement
    </To>
    <Action>
      http://schemas.xmlsoap.org/ws/2004/09/
        enumeration/Pull
    </Action>
    <MessageId>
      uuid:9180bb55-9f9d-4808-93ca-72a6e922105a
    </MessageId>
    <ReplyTo>
      http://schemas.xmlsoap.org/ws/2004/08/
        addressing/role/anonymous
    </ReplyTo>
  </Header>
  <Body>
    <Pull>
      <EmumerationContext>0</EmumerationContext>
      <MaxElements>5</MaxElements>
    </Pull>
  </Body>
</Envelope>
```

Figure 3.7 WS-Management Enumeration Pull

Step 4: The response to the Pull request is shown in Figure 3.8. Note that the response returned the two elements Data1 and Data2. The EnumerationContext is also changed, so that the subsequent requests can be made to return the rest of the elements following the ones that are already returned.

```
<Envelope>
  <Header>
    <To>http://schemas.xmlsoap.org/ws/2004/08/
      addressing/role/anonymous</To>
    <Action>
      http://schemas.xmlsoap.org/ws/2004/09/
        enumeration/PullResponse
    </Action>
    <RelatesTo>
      uuid:9180bb55-9f9d-4808-93ca-72a6e922105a
    </RelatesTo>
    <MessageID>
      uuid:6500bf62-72e6-4468-8c2d-cd969ed0bd56
    </MessageID>
  </Header>
  <Body>
    <PullResponse>
      <EmumerationContext>
        2
      </EmumerationContext>
      <Items>
        <CIM_PhysicalElement>
          DATA1
        </CIM_PhysicalElement>
        <CIM_PhysicalElement>
          DATA2
        </CIM_PhysicalElement>
      </Items>
    </PullResponse>
  </Body>
</Envelope>
```

Figure 3.8 The Response to the enumerationPull Request

The two data items returned are the XML representations of the CIM objects, as shown in Figures 3.9 and 3.10.

```
<CIM_PhysicalElement>
  <Tag> 406ACME-08K8198</Tag>
  <Description>
    Physical media (SATA disk)
  </Description>
  <ElementName>SATA Disk</ElementName>
  <Manufacturer>ACME Inc.</Manufacturer>
  <Model>SATA Disk, The Big Cahuna 2005</Model>
  <SKU>AABB8900</SKU>
  <SerialNumber>78999999999999</SerialNumber>
  <Version>1.0</Version>
  <PoweredOn>True</PoweredOn>
  <ManufacturerDate>Jan 30, 2005</ManufacturerDate>
  <VendorEquipmentType>
    SATA Cahuna
  </VendorEquipmentType>
  <CanBeFRUed>True</CanBeFRUed>
</CIM_PhysicalElement>
```

Figure 3.9 XML Representation of the Data Object (Item 1)

```
<CIM_PhysicalElement>
  <Tag> 406BigBlobMem-08K8198</Tag>
  <Description>
    Physical Memory (DDR memory)
  </Description>
  <ElementName>DDR memory</ElementName>
  <Manufacturer>BigBlobMem Inc.</Manufacturer>
  <Model>DDR memory, The Big Blob</Model>
  <SKU>99000ababab</SKU>
  <SerialNumber>756568432</SerialNumber>
  <Version>1.0</Version>
  <PoweredOn>True</PoweredOn>
  <ManufacturerDate>Jan 30, 2005</ManufacturerDate>
  <VendorEquipmentType>
    DDR, Blob
  </VendorEquipmentType>
  <CanBeFRUed>True</CanBeFRUed>
</CIM_PhysicalElement>
```

Figure 3.10 XML Representation of Data Object (Item 2)

Step 5: If the management console has knowledge (as a result of prior enumeration) that a target object exists on the managed system, then it can make a direct GetRequest call, as shown in Figure 3.11. In this example, a specific call to get the Processor object is shown. The Selector of CPU0 selects the specific instance of the Processor object. The transfer/Get specifies the action, indicating that this is a Get operation as per WS-transfer protocol semantics.

```
<Envelope>
  <Header>
    <To>
      http://192.168.0.100/
        wsman?ResourceURI= http://schemas.dmtf.org/
           wbem/wscim/1/cim-schema/2+/CIM_PhysicalElement
    </To>
    <Action>http://schemas.xmlsoap.org/ws/
      2004/09/transfer/Get
    </Action>
    <SelectorSet>
      <Selector=“DeviceID”>CPU0</Selector>
    </SelectorSet>
    <ReplyTo>
      http://schemas.xmlsoap.org/ws/2004/08/
        addressing/role/anonymous
    </ReplyTo>
    <MessageID>
      uuid:da649f50-3368-4646-bc90-9b294ea058fb
    </MessageID>
  </Header>
  <Body/>
</Envelope>
```

Figure 3.11 WS-Management Get Request

Step 6: The response to the Get request comes in the form of GetResponse, shown in Figure 3.12.

```
<Envelope>
  <Header>
    <To>
      http ://schemas.xmlsoap.org/ws/2004/08/
        addressing/role/anonymous
    </To>
    <Action>
      http://schemas.xmlsoap.org/ws/2004/09/
        transfer/GetResponse
    </Action>
    <RelatesTo>
      uuid:da649f50-3368-4646-bc90-9b294ea058fb
    </RelatesTo>
  </Header>
  <Body> DATA </Body>
</Envelope>
```

Figure 3.12 WS-Management Get Response

The Data element in the message shown in Figure 3.12 is expanded in Figure 3.13. Again, this is an XML representation of the CIM_Processor object.

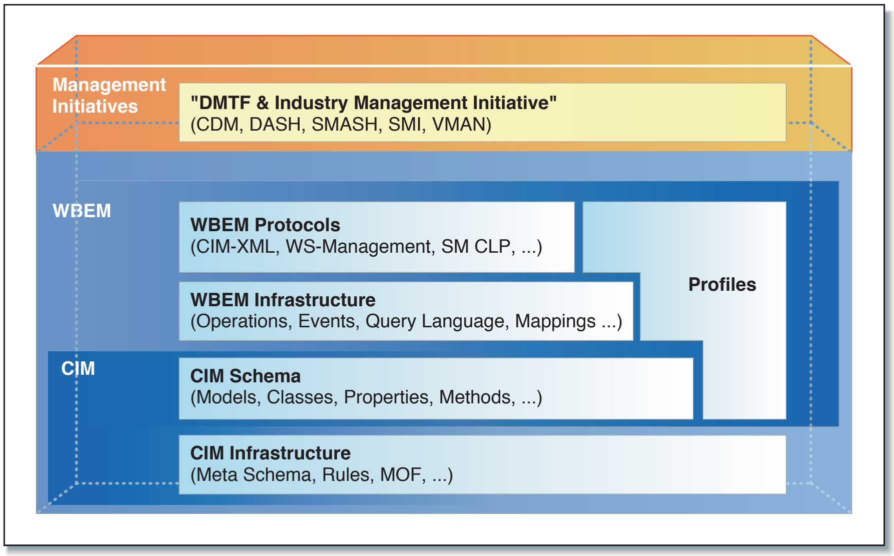
```
<CIM_Processor>
  <Family>165</Family>
  <DeviceID>CPU0</DeviceID>
  <OtherFamilyDescription>
    “Intel (R) Xeon (TM)“
  </OtherFamilyDescription>
  <MaxClockSpeed>3000</MaxClockSpeed>
  <CurrentClockSpeed>3000</CurrentClockSpeed>
  <DataWidth>32</DataWidth>
  <AddressWidth>32</AddressWidth>
  <LoadPercentage>2</LoadPercentage>
  <Stepping>5</Stepping>
</CIM_Processor>
```

Figure 3.13 XML Representation of Data Object Embedded in Get Response

Other operations supported by WS-Management are WS-Transfer:Put and WS-Transfer:Delete. WS-Management also supports events, which is based on WS-Eventing specification. WS-Eventing allows a management console to create, delete, and renew event subscriptions. Once subscribed to events, a management console receives the events in a similar way as most of the responses discussed above.

Putting It All Together

Having looked at the different components of the standards-based management infrastructure, let's look at how these pieces fit together. Figure 3.14 provides a bird's eye view of the different technologies discussed in this chapter.


Management Initiatives
"DMTF & Industry Management Initiative"
(CDM, DASH, SMASH, SMI, VMAN)
WBEM
WBEM Protocols
(CIM-XML, WS-Management, SM CLP, ...)
WBEM Infrastructure
(Operations, Events, Query Language, Mappings ...)
Profiles
CIM
CIM Schema
(Models, Classes, Properties, Methods, ...)
CIM Infrastructure
(Meta Schema, Rules, MOF, ...)

Courtesy of dmtf.org

Figure 3.14 Manageability Standards and Initiatives

CIM Infrastructure defines the language, MOF, UML, and the rules that provide a foundation of Common Information Model.

CIM Schema comprises the over 2000 CIM classes that define various objects, their properties, and methods.

WBEM defines the infrastructure and protocols that specify how management applications interact with the Common Information Model.

Profiles define the system behavior and implementation requirements for classes and properties. This allows for interoperability among different implementations.

To provide an overall interoperability guarantee of how these components play together, initiatives such as SMASH and DASH put the processes in place such as interoperability events and compliance requirements.

Summary

In Chapter 1, we discussed the general concepts of systems and platform management. We then, in Chapter 2, looked at the manageability from a historical perspective. In this chapter, we reviewed the prevalent management standards. These standards are used by Intel® Active Management Technology (Intel AMT), which is a major feature of Intel® vProTM technology. In the next chapter, we start looking at the capability of Intel vPro platforms, and then dive into a discussion of Intel AMT for a majority of the book. In Chapter 19, we will circle back and discuss the standard profiles supported by Intel AMT.