Virtual AMT for Unified Management of Physical and Virtual Desktops

Kenichi Kourai
Department of Creative Informatics
Kyushu Institute of Technology
Fukuoka, Japan
kourai@ci.kyutech.ac.jp

Kouki Oozono
Department of Creative Informatics
Kyushu Institute of Technology
Fukuoka, Japan
oozono@ksl.ci.kyutech.ac.jp

***Abstract*—To reduce the burden of administrators in enterprises, current PCs are equipped with Intel Active Management Technology (AMT). AMT enables administrators to perform hardware-level remote management of desktops even on system failures. Recently, however, virtual desktops are emerging with virtual machines (VMs), e.g., in Desktop as a Service (DaaS). Since physical and virtual desktops are mixed in current enterprises, administrators have to manage them using two different tools: that for AMT and that for VMs. In this paper, we propose *vAMT*, which is virtual AMT for VMs. vAMT provides the same interfaces as AMT: WS-Management, SOAP, and KVM interfaces. Using AMT and vAMT, administrators can perform unified management of physical and virtual desktops without being aware of the differences in most of the operations. We have implemented vAMT and confirmed that the existing management tools for AMT could also manage virtual desktops.**

I. Introduction

The number of desktops such as PCs used in enterprises becomes enormous. Administrators in the IT department have to manage all the desktops in all the departments. To manage desktops located in remote departments, many management tools install agent software in target desktops [1], [2]. Then they can install and update software, scan viruses, and perform backups by communicating with the agents. However, when a desktop is turned off, such software-level management tools cannot access the desktop. Even when the power is on, they cannot manage a desktop if the agent or the operating system does not work correctly due to system failures or attacks.

To improve the manageablity of desktops, current PCs for enterprises are equipped with Intel Active Management Technology (AMT) [3]. AMT is one of the core technologies in Intel vPro and is implemented in an embedded processor separated from main CPUs. It enables remote management even if target desktops are turned off or even on system failures. Using AMT, administrators can perform hardware-level management of desktops. Management tools for AMT can monitor and control the systems on the desktops without depending on agents although they also use agents to provide high-level management functions.

Recently, however, virtual desktops are emerging, e.g., in Desktop-as-a-Service (DaaS) clouds. A virtual desktop runs as a virtual machine (VM) in a server and only its screen is displayed in user's device. Since all the desktops cannot be replaced with virtual desktops, physical and virtual desktops are mixed in the current enterprises. In such an environment, administrators have to manage these different desktops using different management tools because management using AMT is applicable only to physical desktops. As a result, the burden of desktop management increases.

In this paper, we propose virtual AMT named *vAMT* for managing virtual desktops. vAMT provides the same interfaces as those provided by AMT. The interfaces include WS-Management [4], Simple Object Access Protocol (SOAP) [5], and Keyboard/Video/Mouse (KVM). Remote management tools can monitor and control virtual desktops through the WS-Management interface with Common Information Model (CIM) [6] or the SOAP interface for Web services. They can also provide out-of-band remote GUI control of virtual desktops by Virtual Network Computing (VNC) [7] with the RFB protocol [8].

Using both AMT and vAMT, administrators can perform unified management of not only physical but also virtual desktops. However, this cannot be achieved by simply providing the same interfaces. There are several differences between physical and virtual desktops. One of the differences is that VMs are destroyed when virtual desktops are turned off. Another is that VMs do not have sensors such as temperature. To bridge the gap between physical and virtual desktops, vAMT allows transparent management of turned-off virtual desktops like turned-off physical ones. It also emulates the functionalities of physical desktops as much as possible.

We have implemented vAMT using OpenPegasus [9], Axis2 [10], and Kernel-based Virtual Machine (KVM) [1] [11]. For the WS-Management interface, we have developed CIM providers for vAMT using CIMPLE [12]. For the SOAP interface, we have developed Web services for vAMT using WSDL2Java included in Axis2. These tools automatically generate templates of CIM providers and Web services from the definition files provided by Intel, respectively. Then we modified the templates so that

[1]Do not confuse the KVM interface and the KVM virtualization software.

vAMT monitors and controls a VM using libvirt [13]. We confirmed that Intel System Defense Utility worked well for both vAMT and AMT.

The rest of this paper is organized as follows. Section II describes desktop management using AMT and emerging virtual desktops. Section III proposes vAMT and Section IV describes the interfaces provided by vAMT. Section V explains the implementation details of vAMT and Section VI shows experimental results. Section VII discusses related work and Section VIII concludes this paper.

II. Desktop Management

A. AMT

AMT is a core technology in Intel vPro and enables the remote management of physical desktops such as PCs at a hardware level. AMT is implemented in a chip separated from CPU and can control hardware such as physical KVM and network interface cards (NICs). The basic functions are detection, failure recovery, and protection. For detection, AMT enables remote management tools to obtain information on a target desktop even if the power of the desktop is turned off. At boot time, AMT obtains information on hardware and software of a desktop from system management BIOS (SMBIOS) and stores it into its flash memory. Then it returns the information anytime.

For failure recovery, AMT enables remote management tools to control hardware of a target desktop. AMT can reboot a desktop when the operating system does not respond. Even when the operating system cannot boot, AMT can boot a desktop using a remote disk image including hardware diagnostic tools and identify the problem. Furthermore, AMT provides out-of-band remote GUI control of a desktop and directly accesses a video card, a keyboard, and a mouse. Even if the network is unreachable due to configuration errors in the operating system, remote management tools can still access the desktop with the network of AMT.

For protection, AMT can monitor the behavior of agents running on top of the operating system in a target desktop. Agents for AMT can send heartbeats to AMT periodically. If the heartbeats are interrupted for some reason, e.g., by intruders, AMT can send alerts to administrators. In addition, AMT can restrict network accesses from the desktop by controlling NICs. For example, it can deny all the communication until the compromised system is recovered.

B. Virtual Desktops

Virtual desktops are emerging desktop usage. Unlike a physical desktop running as a client PC, a virtual desktop runs as a VM in a server machine. In particular, cloud-based virtual desktops are called Desktop as a Service (DaaS). In a virtual desktop, only the screen of a VM is displayed in user's device such as a PC and a smart device, and keyboard and mouse inputs are sent to the VM via a network. Such desktop virtualization enables administrators to consolidate desktops in a server. As a result, it becomes easier to install and update software and to maintain the system. From the users' point of view, their desktops can be used everywhere.

Currently, administrators have to manage both physical and virtual desktops remotely because these desktops are mixed in many enterprises. The transition from physical to virtual desktops is in progress. Even if the transition is complete, physical desktops such as PCs may be still necessary for running virtual desktops. In addition, it is difficult to use virtual desktops for laptop PCs because they can be used in an environment where network is unreachable. Recent virtual desktops are being usable in network-unreachable environments, but the functionality is restricted.

In such an environment, administrators have to use at least two different management tools to manage two different types of desktops. Since AMT is only for physical desktops, management tools for AMT cannot manage virtual desktops. In contrast, those for virtual desktops cannot manage physical desktops with AMT. This makes the burden of desktop management increase. Administrators have to first distinguish physical and virtual desktops and then use an appropriate tool. Since most of management tools have different user interfaces, administrators can be confused when they switch tools. Another approach is to add the support for virtual desktops to management tools for AMT, or to add the support for AMT to management tools for virtual desktops. However, the burden of the developers of management tools is large in any case.

On the other hand, physical and virtual desktops have many similarities in terms of remote management. Since both desktops can run the same software, administrators manage it in the same way, for example, using remote GUI control. On system failures, both desktops can be recovered by reboots. Therefore, it is desirable that administrators can deal with both desktops using the same tools for common management tasks.

III. Virtual AMT

This paper proposes *vAMT*, which is virtual AMT for VMs. Like AMT for managing physical desktops, vAMT enables administrators to manage virtual desktops using the WS-Management, SOAP, and KVM interfaces. Remote management tools can access vAMT by sending WS-Management requests or SOAP requests. They can also access vAMT for out-of-band remote GUI control of virtual desktops by VNC with the RFB protocol. Using both AMT and vAMT, administrators can perform unified management of physical and virtual desktops using the existing management tools for AMT, as shown in Fig. 1. They do not need to be aware of the differences between physical and virtual desktops in most of the operations.

The architecture of vAMT is illustrated in Fig. 2. The primary components of vAMT are a WS-Management server,

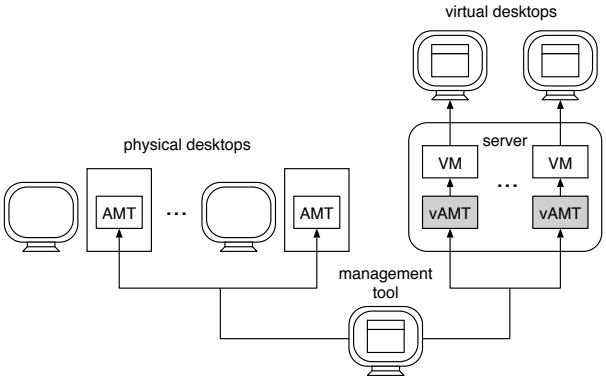

Figure 1. The system overview using both AMT and vAMT.

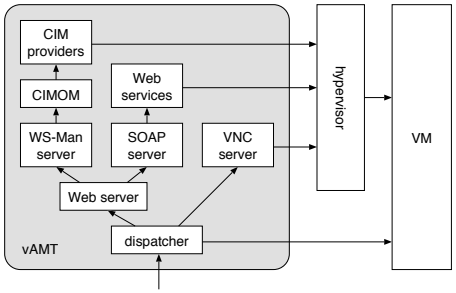

Figure 2. The system architecture of vAMT.

a SOAP server, and a VNC server. When a request is sent in WS-Management, the WS-Management server converts it to a corresponding CIM request and sends the CIM request to CIM object manager (CIMOM). CIMOM delivers the CIM request to an appropriate CIM provider, which is prepared for each function of vAMT. It monitors and controls a VM by accessing virtual hardware of the VM. On the other hand, when a request is sent in SOAP, the SOAP server invokes an appropriate Web service, which is prepared for each category of vAMT functions. When a VNC request is sent, the VNC server handles it by accessing virtual hardware of a VM.

vAMT can be accessed by specifying the same IP address as a target VM. This is the same as AMT, which can be accessed using the IP address of the PC equipped with AMT. The dispatcher of vAMT dispatches various accesses for one IP address by port numbers. For example, it delivers packets to port 16992 to the Web server, which further dispatches requests to either the WS-Management server or the SOAP server by URLs. It delivers packets to port 5900 to the VNC server. The other packets are delivered to a VM.

What to consider for vAMT are the differences between physical and virtual desktops. One of the biggest differences is the state when the power of a desktop is turned off. For physical desktops, PCs always exist because they are concrete hardware. For virtual desktops, however, VMs are destroyed after they stop. To absorb this difference, vAMT allows transparent management of turned-off virtual desktops like turned-off physical ones. Using vAMT, management tools for AMT are not aware that target virtual desktops are turned on or off.

Another difference is the functionalities between PCs and VMs. For example, PCs have physical sensors such as temperature and management tools should monitor the status of the sensors. In contrast, VMs do not have such sensors. As specific features, they can be newly created and be migrated to other hosts by management tools. Therefore vAMT emulates the functionalities of PCs as much as possible, whereas it does not support VM-specific operations because those are not included in the AMT specification. Such operations are still done using VM-specific tools.

IV. vAMT Interfaces

vAMT provides three interfaces to remote management tools. These interfaces are used for obtaining information from vAMT and a target VM and for controlling the VM.

A. WS-Management Interface

To allow remote management using the CIM extended for AMT, vAMT provides the WS-Management interface. WS-Management became a standard interface from AMT version 6.0. CIM is used through this interface. It adopts an object-oriented hierarchical architecture to make it easy to express target components and track mutual dependencies among them.

CIM is composed of classes, properties, methods, modifiers, references, and associations. A *class* defines properties and methods that its instances commonly have. A *property* has a name, a data type, and a value. The value expresses the characteristics of its class. A *method* is invoked for operating an instance of the class defining the method. A *modifier* provides additional information on a class, a method, a method parameter, a property, a reference, and an association. A *reference* is a special data type indicating a pointer to another instance. An *association* is a type of class including more than two references and expresses the relationship between instances of different classes.

To define CIM classes, the managed object format (MOF) language is used. MOF is an extension of BNF notation for syntax and is defined in the CIM specification [6]. Fig. 3 illustrates an example of a CIM class described in MOF. This example defines three classes, CIM_Processor, CIM_Chip, and CIM_Realizes. For brevity, these definitions are a bit different from actual classes defined in AMT.

CIM_Processor is a class inherited the CIM_LogicalDevice class, which handles logical information on a device. It has the Number property for a CPU number and the Enable method for enabling and disabling CPUs. The Key modifier in the Number property indicates that the property is used for identifying an instance. Such a property is called a *key property*. The IN

```
class CIM_Processor : CIM_LogicalDevice {
  [Key] uint32 Number;
  uint32 Enable([IN] boolean Enabled);
};

class CIM_Chip : CIM_PhysicalElement {
  [Key] string Tag;
};

class CIM_Realizes {
  [Key] CIM_PhysicalElement REF Antecedent;
  [Key] CIM_LogicalDevice REF Dependent;
};
```

Figure 3. CIM classes described in MOF.

modifier in the parameter of the Enable method indicates that the parameter is used for an input.

CIM_Realizes is an association class whose properties are references to CIM_PhysicalElement and CIM_LogicalDevice. REF after each data type indicates that the property is a reference. This class associates logical and physical information on a device. For example, an instance of CIM_Processor is associated with that of CIM_Chip, which inherits the CIM_PhysicalElement class.

To operate CIM classes and instances, CIM operations are used. For example, remote management tools can obtain all the instances of the specified CIM class by the EnumerateInstances operation. To obtain one specific instance, they specify one or several key properties using the GetInstance operation. They can also invoke the specified method of a CIM class with parameters and obtain a return value. These operations are bound to WS-Management operations [14].

B. SOAP Interface

To allow remote management using Web services for AMT, vAMT provides the SOAP interface. This interface was deprecated from AMT version 6.0, but many tools continue to use this interface. The support was removed from AMT version 9.0. To define Web services, Web services description language (WSDL) is used. WSDL is based on XML and is used for describing the functionality offered by Web services.

A WSDL document is mainly composed of port types, operations, messages, and types. A *port type* defines related operations collectively. An *operation* defines a method with input messages (method parameters) and output parameters (return values). A *message* defines data with a type. A *type* defines a data type.

Fig. 4 illustrates an example of a Web service described in WSDL. Note that this example omits several definitions. This example defines RemoteControl operation in the RemoteControlSoapPortType port type. This port type manages the power of a VM and how to boot it. The input message of the operation is RemoteControlIn, whereas the output message is RemoteControlOut. The RemoteControlIn message is composed of the RemoteControl type, which contains the RemoteControlCommand type. In this example, we omit the definition of the RemoteControlOut message.

```
<types>
  <element name="RemoteControl">
    <element name="Command"
             type="RemoteControlCommand"/>
  </element>
</types>

<message name="RemoteControlIn">
  <part name="parameters" element="RemoteControl">
</message>

<portType name="RemoteControlSoapPortType">
  <operation name="RemoteControl">
    <input message="RemoteControlIn"/>
    <output message="RemoteControlOut"/>
  </operation>
</portType>
```

Figure 4. A Web service described in WSDL.

C. KVM Interface

To enable out-of-band remote GUI control of a VM, vAMT provides the KVM interface. Using VNC, remote management tools can monitor the screen of a VM and control the keyboard and the mouse of a VM. Unlike in-band remote management, they connect to vAMT using VNC and vAMT directly accesses virtual hardware of a VM. This out-of-band management does not depend on the system state inside a VM. Even if network failure occurs inside a VM, remote tools can access the VM. vAMT provides two methods for this out-of-band remote management. One method uses a default port and allows remote tools to connect to vAMT in the standard VNC protocol. The other method uses a redirection port and enables remote tools to securely connect to vAMT using a proprietary protocol. Currently, vAMT does not support the redirection port.

D. Other Interfaces

AMT also provides other interfaces: Serial over LAN (SOL) for serial console, IDE Redirection (IDER) for mounting a remote image, Remote Management and Control Protocol (RMCP) in Alert Standard Format (ASF) [15] for pings. SOL and IDER use the redirection port with proprietary protocols. vAMT does not support these interfaces currently.

V. Implementation

A. System Architecture

Fig. 5 depicts the detailed system architecture of vAMT. For the WS-Management server and the CIMOM, we used

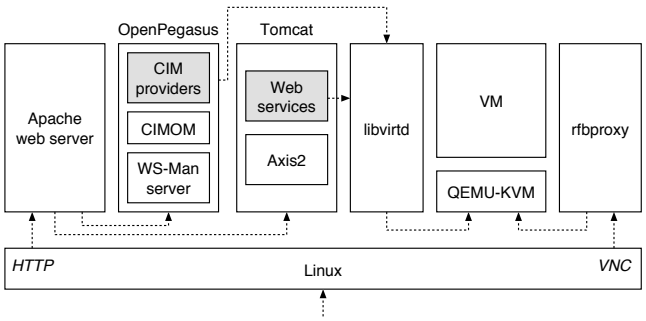

Figure 5. The detailed system architecture of vAMT.

OpenPegasus 2.11.1 [9]. To handle requests from management tools for AMT, we have modified OpenPegasus. One modification is using the default namespace (root/cimv2) when any namespace is not specified. OpenPegasus requires specifying a namespace in a selector, whereas several management tools we tried do not specify any namespace. Another modification is using the IP address and port number of OpenPegasus when an end-point reference (EPR) is "default." OpenPegasus requires specifying an EPR explicitly, whereas several management tools for AMT do not. In addition, we modified OpenPegasus so that it accepts WS-Management requests in which CIM classes do not start with "CIM_". AMT uses not only CIM classes starting with "CIM_" but also those starting with "AMT_" and "IPS_" as an AMT extension. Also, we added the digest access authentication [16] in HTTP to OpenPegasus, which supported only the basic access authentication. The digest access authentication is required by many management tools.

For the SOAP server, we used Apache Axis2/Java 1.6.2 [10] with the Apache Tomcat 6.0.35 [17]. Axis2 contains various tools for generating and deploying Web services. CIM providers in OpenPegasus and Web services in Tomcat access the libvirt daemon 1.1.4 [13] to monitor and control a VM. The Apache web server [18] is used for dispatching requests to the WS-Management server and the SOAP server.

For the VNC server, we used QEMU-KVM 0.12.5 [19]. QEMU-KVM is a user space tool of the KVM virtualization software and provides a VNC server for connecting to its VM. Rfbproxy [20] is used for supporting a turned-off VM, as described in Section V-E.

B. CIM Providers for vAMT

To create CIM providers for managing VMs, we used CIMPLE 2.0.24 [12]. CIMPLE generates templates of CIM providers from MOF in which CIM classes are described. Specifically, it generates two C++ classes from one CIM class. One is a class directly corresponding to a CIM class and defines members corresponding to the properties in the CIM class. The other is a class corresponding to a CIM provider for the CIM class and defines member functions such as enum_instances and get_instance. These functions are invoked when CIM operations such as EnumerateInstances and GetInstance are executed.

We used the MOF files included in SDK provided by Intel. We modified the syntax check of CIMPLE slightly so that it could parse MOF files for AMT. The original CIMPLE does not permit that the same key properties appear in both a super class and a subclass, but the MOF files for AMT violate this constraint.

To monitor and control VMs from vAMT, we used the libvirt virtualization library 1.1.4 [13]. Libvirt enables vAMT to handle VMs running on top of various virtualization software in a unified manner. For example, vAMT can obtain a domain object using the virDomainLookupByName function with a VM name. Note that vAMT specifies virtualization software in the virConnectOpen function invoked at first. The library communicates with the libvirt daemon and the daemon accesses QEMU-KVM.

Fig. 6 shows the function prototypes of the CIM provider for the CIM_Processor class in Fig. 3. The enum_instances function is executed for enumerating all the instances of the class. CIMPLE generates only a function header and a return statement in the body. This function creates the same number of instances of CIM_Processor as virtual CPUs. Then it sets a different CPU number to the Number property of each instance. Finally, it registers the instances to the parameter handler to return them to a remote tool.

The get_instance function is executed for obtaining one specific instance of the class. The parameter model stores only the value of the key property Number requested by a remote tool. This function searches the instance whose property Number has the same value as model. If only one instance is found, the function sets all the properties of the instance to the parameter instance. Otherwise, it returns an error. In this example, the function returns information on the virtual CPU with the specified CPU number.

The Enable function is executed when a remote tool invokes the Enable method using a CIM operation. The target instance is specified by the parameter self and the method parameter is passed by the parameter Enabled of this function. In this example, the status of the virtual CPUs is maintained in the array of the cpuinfo objects. If the value of Enabled is true and the target virtual CPU is not enabled, the function enables it. If the value of Enabled is false and the target virtual CPU is enabled, the function disables it. Then it re-calculates the number of enabled virtual CPUs and changes the actual CPU assignment to the specified VM by using the virDomainSetVcpus function in libvirt. Finally, it sets zero to the parameter return_value if the request is successfully executed.

For association classes, CIM providers are implemented similarly. Let us consider the enum_instances function of the CIM_Realizes class in Fig. 3. First, the function

```
Enum_Instances_Status enum_instances(const CIM_Processor* model, Enum_Handler<CIM_Processor>* handler);
Get_Instance_Status get_instance(const CIM_Processor* model, CIM_Processor*& instance);
Invoke_Method_Status Enable(const CIM_Processor* self, const Property<boolean>& Enabled,
                            Property<uint32>& return_value);
```

Figure 6. The function prototypes of the CIM provider for CIM_Processor.

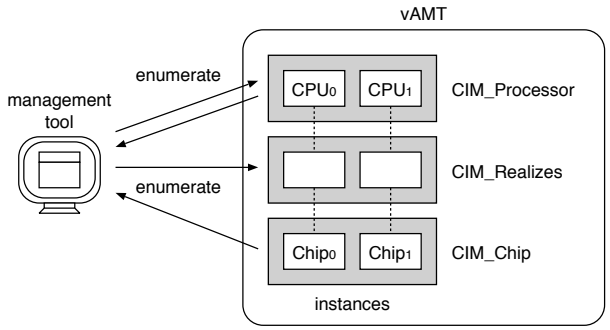

Figure 7. Obtaining CPU information.

creates two instances of CIM_Chip and CIM_Processor, respectively, and sets the values to only their key properties. Next, it creates an instance of CIM_Realizes and sets the above two instances to its members as references. At this time, these are casted to their super classes. Finally, it registers the instance to handler. From all the registered instances, CIMPLE finds all the instances associated with a requested instance and returns them to a remote tool.

A remote tool combines these CIM providers and obtains CPU information as shown in Fig. 7. It first obtains all the instances of CIM_Processor. For each instance, it obtains associated instances of CIM_Chip by requesting EnumerateInstances for CIM_Realizes. Similarly, it obtains instances of CIM_Location, which are associated with each instance of CIM_Chip. CIM_Location is a CIM class for the physical position and address of a device. For the association class, CIM_PhysicalElementLocation is used. Finally, the remote tool can obtain necessary information from the properties of all the obtained instances.

As another example, the power management of a VM requires three CIM providers when only the WS-Management interface is used. First, a remote tool obtains the value of the PowerState property of the CIM_AssociatedPowerManagementService class to examine the power status of a VM. Next, it obtains an instance of the CIM_ComputerSystem class for the target system of the power management. Finally, it invokes the RequestPowerStateChange method of the CIM_PowerManagementService class to execute the power management using libvirt.

There are 264 CIM classes for vAMT. Since all CIM classes are not used in real management tools, we prioritized CIM classes by examining the real uses in several management tools. Then we have implemented 39 providers.

C. Web Services for vAMT

To create Web services for managing VMs, we used a tool named WSDL2Java included in Apache Axis2/Java. WSDL2Java generates templates of Web services from WSDL. Specifically, it generates multiple Java classes for all the port types and the types from one WSDL file. A class for a port type has methods corresponding to the operations defined in the port type. We used the WSDL files included in SDK provided by Intel.

Let us explain the implementation of the RemoteControl operation for the RemoteControlSoapPortType port type in Fig. 4. The remoteControl method is executed when a remote tool requests this operation. The parameter types and the return type correspond to the types of input and output messages, respectively. In this example, the method obtains the requested command type from the RemoteControl object in the input parameter and manages the power of a VM according to the value. After the command execution, the method returns the status.

To monitor and control VMs, we used the libvirt-java binding tool 0.5.1 [13]. This tool provides a Java application programming interface (API) so that Java programs can use the functionality of libvirt. In the example of the RemoteControl operation, the method invokes the shutdown method for the Domain object, which corresponds to a target VM.

In general, many operations in web services for AMT return not only status but also various information with complex data structure to a remote tool. For example, the CbFilterEnumerate operation in the CircuitBreaker web service returns information on packet filtering with the data structure as in Fig. 8. In addition to the status, CbFilterEnumerateResponse consists of a filter name such as "NoPing," an IP address, a network mask, etc. For such operations, we created many Java objects and assembled the data structure by using the accessors defined for properties.

There are 522 operations in 23 Web services for vAMT. We prioritized these operations on the basis of the real uses in several management tools and have implemented 20 operations.

D. Request Redirection

To redirect requests to appropriate servers as in Fig. 2, vAMT configures static network address and port translation

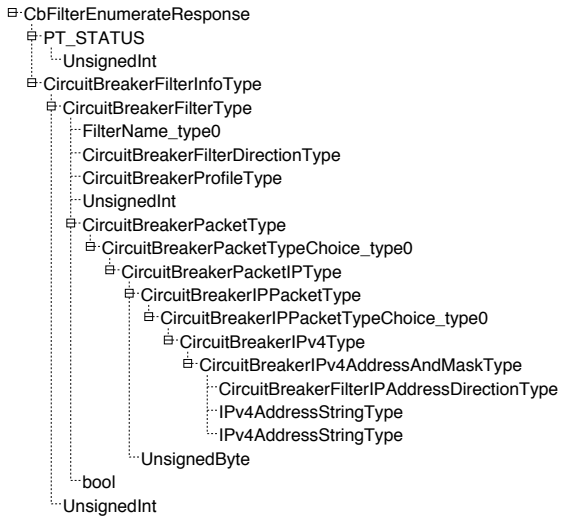

Figure 8. The data structure of a response to the `CbFilterEnumerate` operation.

(NAPT) using iptables in the host operating system. For WS-Management and SOAP requests, a pair of the IP address of a VM and port 16992 is translated into that of the IP address of the host and the port of the web server in vAMT. For VNC requests, the IP address of a VM and port 5900 are translated into the IP address of the host and the VNC server in vAMT, respectively. The redirected port numbers are different for each vAMT to coexist multiple vAMTs. For example, requests to VM_1 are redirected to ports 6992 and 5901, whereas those to VM_2 are to ports 6993 and 5902.

vAMT further redirects requests to either the WS-Management server or the SOAP server using a reverse proxy in the web server. If the URL specified in an HTTP request starts with `/wsman`, the request is redirected to the WS-Management server. If the URL starts with `/*Service` such as `/RemoteControlSoapService`, the request is redirected to the SOAP server.

E. Management of Turned-off VMs

To monitor a turned-off VM, vAMT obtains information on it from its configuration. When a VM is running, the runtime of the virtualized system manages the latest information such as the number of virtual CPUs, the memory size of the VM. However, it does not manage such information for turned-off VMs. Fortunately, initial values of such information are stored in VM configuration when a VM is created. vAMT can obtain most appropriate information from either the runtime or the configuration by using libvirt.

To access a turned-off VM using VNC, vAMT uses a VNC proxy server called rfbproxy. For a turned-off VM, a VNC server for it does not exist because QEMU-KVM including a VNC server runs only while a VM is running. Therefore, when a management tool connects to vAMT for a turned-off VM, rfbproxy returns a dummy black screen and ignores keyboard and mouse inputs. This behavior is the same as that of AMT in a turned-off PC. To return such a dummy screen, we recorded a black screen using the function of rfbproxy in advance. Then rfbproxy continues to play back the recorded events. For a turned-on VM, rfbproxy simply redirects requests to the VNC server in QEMU-KVM.

F. Support for VM Migration

VMs can be migrated to another host in various reasons. Since vAMT runs in the outside of a migrated VM, it could not monitor or control the VM after the migration. In the current implementation, vAMT is restarted at the destination host of the VM migration.

To seamlessly support the migration of a target VM, there are two possible ways. One is that vAMT remotely manages the VM migrated to another host. Requests to vAMT can be redirected from iptables in the destination host. Then vAMT can monitor and control the remote VM by communicating with the libvirt daemon in the destination host. One drawback is inefficiency in the communication for the request redirection and libvirt between the source and destination hosts. Another drawback is that administrators cannot shut down the source host still running vAMT after the target VM has been migrated. The other drawback is that vAMT may lose several requests while vAMT switches its target hosts.

The other way is running vAMT in another VM and migrating together with the target VM using the technique proposed in VMCoupler [21]. After co-migration, vAMT can manage the target VM as before migration. The communication for the request redirection and libvirt is performed efficiently inside the destination host. Administrators can shut down the source host because vAMT is not running. In addition, it is not necessary that vAMT is aware of the migration of the target VM.

VI. Experiments

We conducted experiments to confirm that management tools for AMT could be used for vAMT. For a physical desktop with AMT 7.1.4, we used a PC with one Intel Core i7 3.4 GHz processor and 2 GB of memory and ran Windows 7. For a virtual desktop with vAMT, we used a VM with one virtual CPU and 1 GB of memory and ran Linux 3.2.0 as the guest operating system. For running the VM, we used a PC with one Intel Core i7 2.93 GHz processor and 4 GB of memory and ran Linux 2.6.32 as the host operating system and QEMU-KVM 0.12.5 as the virtualization software. To run management tools remotely, we used a PC with one Intel Xeon W3550 3.06 GHz processor and 6 GB of memory and ran Windows 7. All the PCs and the VM are connected with Gigabit Ethernet.

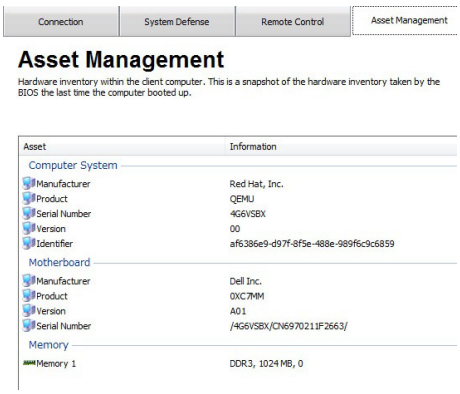

Figure 9. The asset management screen in System Defense Utility.

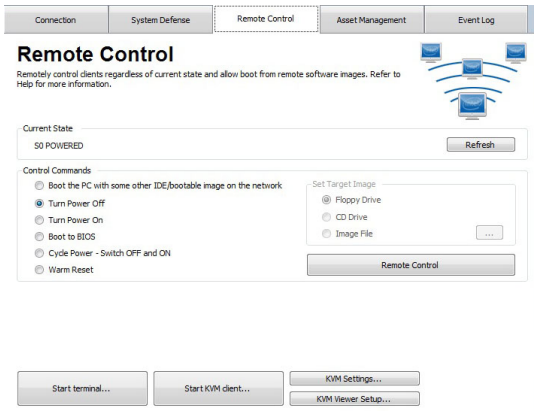

Figure 10. The remote control screen in System Defense Utility.

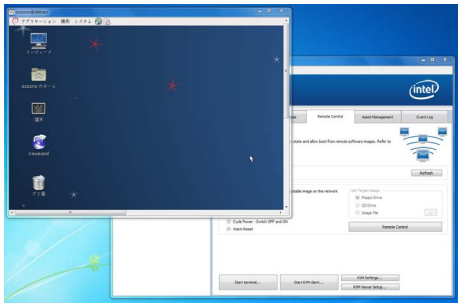

Figure 11. The VNC screen in System Defense Utility.

A. Comparison of Behaviors

To confirm the behavior of vAMT, we used Intel System Defense Utility as a management tool for AMT.

1) Connection: When we connected to AMT, the tool used 40 CIM classes and five Web services and sent 189 requests to AMT in total. Until it established the connection, it obtained the capabilities of AMT, hardware information of a desktop, etc. When we connected to vAMT using the same tool, the tool used 26 CIM classes and five Web services and sent 97 requests to vAMT in total. The reason of the difference between AMT and vAMT is that vAMT does not support several requests yet. However, the tool could establish the connection to vAMT because it ignores functions that are not indispensable.

After the connection was established, the hardware information of the target desktop was shown in the asset management tab, as shown in Fig. 9. When the tool connected to vAMT, information on the VM was shown in the Computer System and Memory areas. For example, the manufacturer is "Red Hat, Inc." and the memory size is 1024 MB. In the Motherboard area, the same information as the PC was shown.

2) Power Management: When we changed the power status of the physical desktop with AMT, the tool sent one WS-Management request and one SOAP request. When we selected the Remote Control tab, the tool obtained the power status with the CIM_AssociatedPowerManagementService class and showed it on the screen. When we executed the power management, the tool changed the power status with the RemoteControlService operation and obtained the power status again.

Using the screen shown in Fig. 10, we confirmed that the tool could perform the power management for the virtual desktop with vAMT in the same way. To examine the actual power status, we obtained the power status using the virsh tool [13] in the PC running the VM. As a result, the power status of the VM was changed as requested.

3) VNC: When the tool accessed to AMT with VNC, it sent only VNC requests, neither WS-Management nor SOAP requests. We confirmed that the tool could access to vAMT and showed the VNC screen of the virtual desktop as in Fig. 11.

B. Performance

To compare the performance of vAMT with that of AMT, we used Windows Remote Management (WinRM) and command-line tools included in the SDK provided by Intel. We measured the time needed to execute each command for AMT and vAMT 10 times.

1) Obtaining the AMT Version: To obtain version information of AMT, we used WinRM, which can send WS-Management requests by specifying a CIM class and its properties. In this experiment, WinRM executed the GetInstance operation for the CIM_SoftwareIdentity class with the InstanceID property. Then AMT returned an instance in which the value of the property was "AMT."

For AMT and vAMT, we measured the time needed for obtaining the AMT version. The results are shown in Fig. 12. When the PC was turned off, it took more than 2 seconds to obtain the AMT version. This is because the AMT chip is in the sleep mode. The successive execution took 392 ms to complete. When the PC was turned on, it took only 112 ms to obtain the version. For vAMT, the execution time was

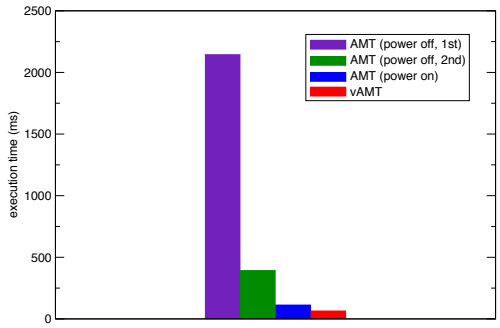

Figure 12. The time for obtaining the AMT version.

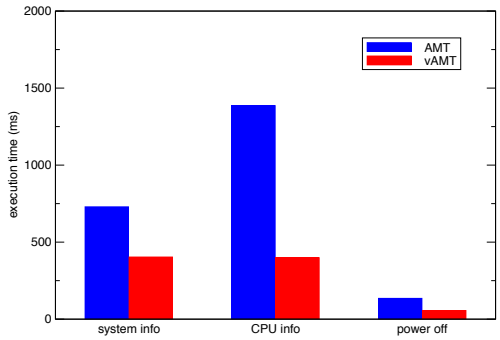

Figure 13. The execution time of commands for AMT and vAMT.

64 ms. This shows that vAMT could handle requests in a shorter time than AMT. The reason is because the hardware performance of the AMT chip is lower than that of the CPU running vAMT. In the following experiments, we turned on the PC and measured the performance of AMT.

2) Obtaining Hardware Information: To obtain hardware information of a physical desktop with AMT, we used the AssetDisplay command. When the command obtained system information, it used three CIM classes: CIM_BIOSElement for BIOS information, CIM_Chassis for a manufacturer and a model name, and CIM_ComputerSystemPackage for a global unique identifier. It sent 12 WS-Management requests for both AMT and vAMT. The left two bars in Fig. 13 show the average execution time. As in the above experiment, vAMT was faster then AMT.

When the command obtained CPU information, it used five CIM classes, as described in Section V-B: CIM_Processor, CIM_Chip, CIM_Location, CIM_Realizes, CIM_PhysicalElementLocation. The former three are for obtaining instances and the latter two are for association. The middle two bars in Fig. 13 show the average execution time. It is shown that AMT is much slower than vAMT. Compared this execution time with the above time for obtaining system information, the execution time for vAMT was almost the same, but that for AMT was 1.9 times slower. This may be caused by using association classes, which have to search association information for a requested instance.

3) Power Management: To change the power status of a physical desktop with AMT, we used the RemoteControlUntyped command. When the command turned the power off, it used one CIM class, CIM_PowerManagementService. The right two bars in Fig. 13 show the average execution time. Similar to the other experiments, vAMT changed the power status faster than AMT, but the difference was smaller. This is because this command sent only one request.

VII. Related Work

There are several hardware-level management interfaces. Wake on LAN (WOL) enables only turning on a PC by a special network message. Intelligent Platform Management Interface (IPMI) [22] is similar to AMT and enables remote management without depending on CPUs and the operating system. It can monitor temperatures, power status, fan status, etc. even when the operating system is down. HP's implementation is called iLO and Dell's is called iDRAC. However, IPMI cannot be used for desktop management because it is for server platforms and is not equipped with PCs.

For IPMI, several IPMI simulators have been developed such as the MIMIC IPMI simulator [23] and the OpenIPMI lanserv simulator [24]. In particular, the lanserv simulator can be used with a virtual IPMI device of QEMU-KVM, which is provided as patches [25]. The lanserv simulator can receive requests from remote management tools and communicate with the virtual IPMI device to manage a VM. In addition, agents inside a VM can access the virtual IPMI device and use the functionalities of IPMI through the lanserv simulator. These simulators are mainly used for testing management tools without server hardware equipped with IPMI. Our vAMT can be also used for such a purpose.

Until Intel AMT SDK 2.x, the AMT emulator was included in the SDK. It is mainly for developing management tools before AMT hardware was released. However, the emulator was removed from SDK 3.0. It may have similarities to vAMT, but it supported only the SOAP interface.

To manage not only physical machines but also VMs through CIM, the CIM specification extended for virtualization [26] is defined. Using this extension, management tools can manage both physical and virtual desktops. In addition, this extension supports VM-specific operations such as creation and migration. Xen-CIM [27] is the first implementation of this specification for Xen. Libvirt-CIM [28] manages Linux virtualization platforms using libvirt. However, this extension requires the modification to the existing management tools. Management tools have to differentiate two types of desktops and use different CIM classes for them. Since AMT does not support this extension, management tools for AMT cannot be used for managing VMs without modification.

Several cloud management software such as CloudStack [29] and OpenStack [30] support not only VMs but also physical machines called bare metal. Traditional Infrastructure-as-a-Service (IaaS) clouds provide users with VMs whenever requested. Recently, they also provide bare metal in the same manner. IPMI is used for the power management of bare metal and Preboot Execution Environment (PXE) is used for booting the operating system. Although cloud management software can handle a mix of VMs and bare metal, it needs to be aware of the differences between VMs and bare metal. In addition, managed bare metal has to be server machines equipped with IPMI.

Desktop management tools can achieve unified desktop management by being combined with management tools for virtual desktops. For example, VMware Horizon Mirage [2] can manage not only physical desktops but also virtual desktops with VMware Horizon View [31]. Microsoft System Center Configuration Manager [1] can manage both types of desktops, for example, with XenDesktop [32]. However, only the maintenance of desktop images and agent-based desktop management are mainly supported because such tools provide only software-level management. Several software-level tools also support AMT, but they can manage only physical desktops, not virtual ones, using AMT.

VIII. Conclusion

In this paper, we proposed vAMT for managing virtual desktops. vAMT has the same interfaces with AMT and enables administrators to manage physical and virtual desktops using the existing management tools in a unified manner. We have implemented vAMT and confirmed that we could monitor and control virtual desktops with vAMT.

Our future work is implementing all the CIM providers and Web services for vAMT. In the current implementation, we have implemented only a minimal set of them as a proof of concept. For example, packet filtering is not supported. Another direction is implementing unsupported interfaces such as SOL. In addition, we need to implement a local interface of AMT, which is accessed from agents inside desktops. Through the interface, the agents can register software information to vAMT for detection and send heartbeats to vAMT for protection.

Acknowledgment

This research was supported in part by JSPS KAKENHI Grant Number 25330086.

References

[1] Microsoft, "System Center Configuration Manager: Unified Device Management," http://www.microsoft.com/systemcenter/configurationmanager/.
[2] VMware, "VMware Horizon Mirage Desktop Management," http://www.vmware.com/products/horizon-mirage.
[3] Intel Corp., "Intel Active Management Technology," http://www.intel.com/amt/.
[4] DMTF, "Web Services for Management Specification Version 1.1.1," 2012.
[5] World Wide Web Consortium, "SOAP Version 1.2," 2007.
[6] DMTF, "Common Interface Model Infrastructure Version 2.7.0," 2012.
[7] T. Richardson, Q. S.-F. K. R. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, 1998.
[8] T. Richardson, "The RFB Protocol Version 3.8," http://www.realvnc.com.
[9] The Open Group, "OpenPegasus," http://www.openpegasus.org/.
[10] Apache Software Foundation, "Apache Axis2," http://axis.apache.org/.
[11] Red Hat, Inc., "Kernel Based Virtual Machine," http://www.linux-kvm.org/.
[12] K. Schopmeyer and M. Brasher, "CIMPLE," http://www.simplewbem.org/.
[13] Red Hat, Inc., "libvirt: The Virtualization API," http://libvirt.org/.
[14] DMTF, "WS-Management CIM Binding Specification Version 1.2.0," 2011.
[15] ——, "Alert Standard Format Specification Version 2.0," 2003.
[16] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617, 1999.
[17] Apache Software Foundation, "Apache Tomcat," http://tomcat.apache.org/.
[18] ——, "The Apache HTTP Server Project," http://httpd.apache.org/.
[19] A. Kivity, "QEMU-KVM Fork for x86," http://wiki.qemu.org/KVM.
[20] T. Waugh, "rfbproxy," http://rfbproxy.sourceforge.net/.
[21] K. Kourai and H. Utsunomiya, "Synchronized Co-migration of Virtual Machines for IDS Offloading in Clouds," in *Proc. Int'l Conf. Cloud Computing Technology and Science*, 2013, pp. 120–129.
[22] Intel, Hewlett-Packard, NEC, and Dell, "Intelligent Platform Management Specification Second Generation v2.0," 2004.
[23] Gambit Communications, Inc., "MIMIC IPMI Simulator," http://www.gambitcomm.com/site/products/IPMISimulator.shtml.
[24] C. Minyard, "OpenIPMI," http://sourceforge.net/projects/openipmi/.
[25] ——, "[Qemu-devel] [PATCH 00/20] Add an IPMI device to QEMU," http://lists.gnu.org/archive/html/qemu-devel/2013-05/msg04335.html.
[26] DMTF, "CIM System Virtualization Model Version 1.0.0," 2007.
[27] G. S. Bestor and J. Fehlig, "Open Standard CIM Management for Xen," Xen Summit, 2006.
[28] Red Hat, Inc., "Libvirt-CIM: The Virtualization CIM API," http://libvirt.org/CIM/.
[29] Apache Software Foundation, "Apache CloudStack: Open Source Cloud Computing," http://cloudstack.apache.org/.
[30] OpenStack Project, "OpenStack Open Source Cloud Computing Software," https://www.openstack.org/.
[31] VMware, "VMware Horizon View Virtual Desktop Management," http://www.vmware.com/products/horizon-view.
[32] Citrix Systems, Inc., "Virtual Desktop Delivery, Virtual Apps on Demand," http://www.citrix.com/products/xendesktop/.