仮想デスクトップとPCの一元管理を可能にする仮想AMT

大薗 弘記　光来 健一

PC があらゆる部署で使われるようになり，組織の管理者が管理しなければならない PC の数は膨大になってきている．管理者は障害が発生すると PC の設置場所まで行って修復作業を行わなければならない場合もある．このような管理者の負担を軽減するために，最近の PC には Active Management Technology（AMT）が搭載されるようになってきている．AMT を用いることで，管理者は PC をリモートから一元的に管理することができるようになる．しかし，近年，仮想デスクトップの普及により，組織内には PC と仮想デスクトップが混在している．AMT では PC の管理を行うことしかできないため，仮想デスクトップの管理は別に行う必要があった．本論文では，仮想デスクトップと PC を一元的に管理できるようにするために仮想マシン（VM）に対して仮想的な AMT を提供する仮想 AMT（vAMT）を提案する．vAMT は物理マシンを管理する AMT と同様のインタフェースで VM を管理することを可能にする．

1　はじめに

企業など組織の IT 化の進展に伴い，組織で使用される PC の数は膨大になってきている．組織の IT 部門の管理者はこれらすべての PC を管理しなければならない．管理の内容は，ヘルプデスクサポートやパッチ配布，ソフトウェアのアップグレード，ウィルス対策など多岐にわたる．例えば，離れた部署のユーザからの障害報告への対応を行う場合，問題になっているシステムが管理者の目の前にないために対応が難しいことがある．その場合，管理者は障害の発生した PC の設置場所まで赴き，問題箇所を特定してから，障害の原因によっては交換の必要な部品を取りに戻ることになる．OS が動作していて，リモートコントロール用のエージェントがその上で正常に動作している場合は，リモートでの障害対応もかなり容易ではある．しかし，OS そのものが起動しない場合や BIOS 設定を変更する必要がある場合，あるいはハードウェアに問題がある場合などは，ソフトウェアベースの管理だけでは対応しきれない．

そのような問題から，最近の PC には AMT が搭載されるようになってきている．AMT を用いることで，管理者はリモートから PC を一元的に管理することができるようになる．AMT により PC をハードウェアレベルで管理することができるため，リモートで PC の OS 起動前の画面表示を確認したり，BIOS 設定画面をリモートで操作したりすることができ，管理者の負担は大きく軽減される．

しかし，近年，サーバの仮想マシン（VM）上でシステムを動作させ，その画面だけを PC 上で表示する仮想デスクトップが普及してきている．これにより，組織内は物理マシンである PC と VM が混在する環境になっている．このような環境において，AMT では物理マシンの管理を行うことしかできないため，VM の管理は別に行う必要がある．

そこで，仮想デスクトップと PC を一元的に管理できるようにするために，VM に対して仮想的な AMT を提供する仮想 AMT（vAMT）を提案する．vAMT

Kouki Oozono, Kenichi Kourai, 九州工業大学, Kyushu Institute of Technology

は物理マシンを管理する AMT と同様のインタフェースで VM を管理することを可能にする．AMT と vAMT を用いることで，PC と仮想デスクトップの違いを意識することなく，既存の AMT 用管理ツールを用いて一元的な管理を行うことができるようになる．

2 章では AMT の基本的な機能と仮想デスクトップとの混在環境における問題点について述べる．3 章では vAMT を提案し，vAMT のインタフェースである CIM について説明する．4 章で vAMT の機能を実現するために作成した CIM プロバイダについて述べる．5 章で vAMT の動作に関する実験について述べる．6 章で関連研究について触れ，7 章で本論文のまとめと今後の課題を述べる．

2 従来の PC 管理

2.1 AMT によるリモート管理

AMT は，インテル社が提供する vPro の管理機能の核となる技術であり，PC をハードウェアレベルで管理することができる．AMT の基本的な機能としては，検出，障害回復，保護があり，これらは以下の各機能によって実現される．

コンピュータ資産の検出 AMT はシステムの起動時に System Management BIOS（SMBIOS）[1] テーブルからハードウェアとソフトウェアに関する情報を取得し，それを独自のフラッシュメモリに格納している．これにより，PC の電源がオフの状態でもリモートから情報の取得を可能にする．ネットワーク上に存在する AMT 対応のシステムを検出してリストアップすることもできる．

リモート電源制御 AMT を用いてリモートから PC の電源を操作することができる．ソフトウェアベースの管理ツールの場合は電源をオフにすることはできるが，オンにすることはできない．これに対して AMT の場合はハードウェアレベルで管理を行えるため，管理エージェントが起動していない状態でも自由に電源を操作できる．

リモートブート AMT には IDE Redirect（IDE-R）というブートデバイスをネットワーク上の他のストレージにリダイレクトできる機能がある．これにより，管理者側のディスクをリモートからマウントすることができる．OS が起動しない場合に，管理者側にあるハードウェア診断ツールの入った起動ディスクイメージを使って対象の PC を起動し，問題のある箇所を特定することができる．

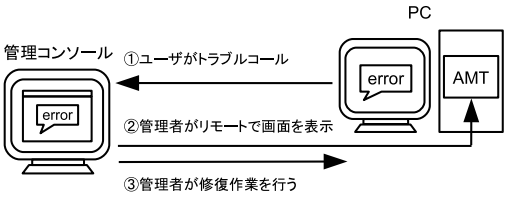

図 1　AMT の障害回復機能

シリアルコンソール Serial Over LAN（SOL）機能によって，BIOS や OS のシリアルコンソールを LAN コントローラにリダイレクトできる．これによって，リモートにある PC の OS 起動前の画面表示を確認したり，BIOS 設定画面をリモートで操作したりすることができる．

KVM リモートコントロール PC に接続されているキーボードやマウスをリモートから操作したり，画面を確認したりすることができる．図 1 のように，PC に問題が発生した時にそのエラー画面を管理者側から見て，設置場所まで行くことなくリモートで修復作業を行える．

パケットフィルタリング AMT が NIC の制御を行うことで，特定の IP アドレスへの送信・受信パケットをすべて遮断したり，特定の通信ポートを閉じたり，逆に，特定のアドレスとの通信のみを許可したりといった設定を行える．これにより，ウィルスに感染した PC をネットワークから遮断し，管理用ポートを経由して修復作業を行った後に再度ネットワークに接続するまでを，すべてリモート処理で行うことが可能になる．

管理エージェントの監視 AMT は OS 上で動く管理エージェントの動作を監視することができる．管理エージェントは定期的に AMT に対してハートビートを送っているので，図 2 のようにハートビートが停止した場合に AMT が管理者にアラートを送信したり，パケットフィルタリング機

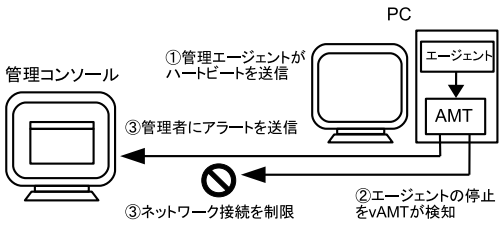

図 2　AMT の保護機能

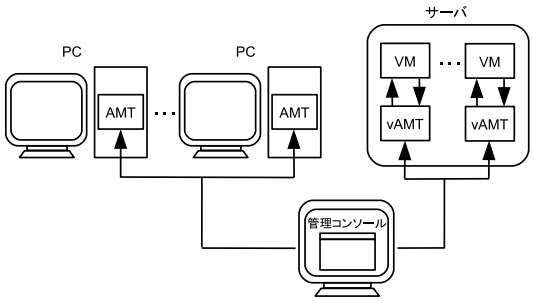

図 3　システムの全体構成

能を用いてネットワークへの接続を制限したりすることができる．

2.2　仮想デスクトップの普及とその管理

近年，PC 上でシステムやアプリケーションを動作させる形態から，仮想デスクトップと呼ばれる形態への移行が進んでいる．仮想デスクトップはサーバ上で VM を動作させて，ネットワークを通じて VM に接続してその画面だけを PC に表示するシステムである．仮想デスクトップを使用することで，システムをサーバで集中管理することができ，ソフトウェアの追加や更新，修正などのメンテナンスが容易となる．仮想デスクトップの普及はまだ過渡期であることや，ネットワークが繋がらない環境で使用されるマシンは仮想デスクトップに置き換えることができないなどの問題により，現在，組織内では PC と仮想デスクトップが混在している．

そのため，組織内の全てのマシンの管理を行う場合，物理マシンである PC と仮想デスクトップである VM の両方を管理する必要がある．しかし，AMT は物理マシンの管理を行うための技術であるため，VM の管理には別の管理ツールを用いる必要がある．そのため，管理者は少なくとも 2 種類の管理ツールを使い分けなければならず，管理が煩雑になる．

3　vAMT

本研究では，仮想的な AMT である vAMT を VM に対して提供する．vAMT は物理マシンを管理する AMT と同様のインタフェースで VM を管理することを可能にする．リモートの管理ツールは Web Services for Management（WS-Management） [2] というプロトコルを用いて vAMT にアクセスを行う．vAMT から情報を取得したり，管理を実行したりするために，システム管理の標準となっている Common Information Model (CIM） [3] を AMT 用に拡張したインタフェースを用いる．AMT と vAMT を用いることで，図 3 のように物理マシンと VM の違いを意識することなく，既存の管理ツールを用いて一元的な管理を行うことができるようになる．

3.1　CIM

管理モデルとして CIM を利用することで管理対象をメーカや種類に関わらず一貫して管理することができる．CIM は異なる管理対象オブジェクトの複雑な相互依存や関連性をたどったり，表現したりすることを容易にするためにオブジェクト指向の階層的アーキテクチャになっている．CIM を定義するための要素として，クラス，プロパティ，メソッド，修飾子，参照，関連がある．

クラス　クラスには特定の種類のオブジェクトに共通するプロパティやメソッドを定義する．

プロパティ　クラスの特徴を表現するための値で，名前，データ型，値を持つ．

メソッド　メソッドが定義されているクラスのインスタンスに対して操作を行うために呼び出される．

修飾子　クラス，メソッド，メソッドの引数，プロパティ，参照，関連に関する追加情報を提供する．

参照　他のインスタンスへのポインタであることを示す特別なデータ型である．

関連　2つ以上の参照を含むクラスの一種であり，異なるクラスのインスタンスの関係を表す．

CIM のクラスを実際に定義するためには Managed Object Format （MOF）が用いられる．MOF は構文仕様のために BNF 記法を拡張したもので，CIM の仕様で定められている．CIM クラスを MOF で書いた例を図 4 に示す．この例では CIM_Processor，CIM_Chip，CIM_Realizes という 3 つのクラスが定義されている．ただし，説明の都合上，図 4 は実際の定義とは少し異なる．

CIM_Processor はデバイスの論理的な情報を扱うための CIM クラス CIM_LogicalDevice を継承したクラスで，CPU 番号を示す Number というプロパティと，CPU の有効化・無効化を行うための Enable というメソッドを持っている．Number に付いている [Key] という修飾子はそのプロパティがインスタンスを識別するのに用いられることを表しており，キープロパティと呼ばれる．また，Enable メソッドの引数である Enabled に付いている [IN] という修飾子はその引数が入力引数であることを示す．

CIM_Chip はデバイスを物理要素として見た時の情報を扱うための CIM クラス CIM_PhysicalElement を継承したクラスで，Tag というキープロパティと CPU の製造メーカを示す Manufacturer というプロパティを持っている．CIM_Realizes は CIM_PhysicalElement と CIM_LogicalDevice の参照型をプロパティに持つ関連クラスである．データ型の後の REF は参照型であることを表している．これは各デバイスの論理情報と物理情報を対応づけるためのクラスである．

CIM のクラスやインスタンスを操作するために，CIM オペレーションが用いられる．表 1 に主な CIM オペレーションを示す．例えば，インスタンスの取得方法には EnumerateInstances と GetInstance の 2 種類がある．EnumerateInstances では CIM クラスのすべてのインスタンスを取得するのに対して，GetInstance ではプロパティの値を指定することで 1 つのインスタンスのみを取得する．

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

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

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

図 4　MOF で記述された CIM クラスの例

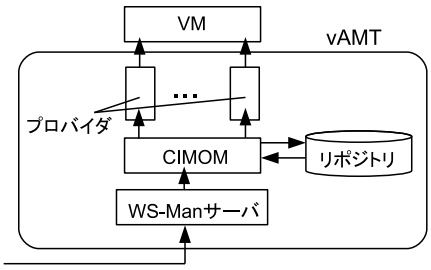

図 5　vAMT の構成

3.2　システム構成

vAMT は図 5 のように，WS-Management サーバ，CIM オブジェクトマネージャ（CIMOM），リポジトリ，CIM プロバイダによって構成される．WS-Management サーバは送られてきたリクエストを CIM オペレーションのリクエストに変換して CIMOM に渡す．CIMOM は CIM プロバイダの登録情報が格納されているリポジトリを参照して，リクエストを適切な CIM プロバイダに送る．CIM プロバイダでは，VM の仮想ハードウェアにアクセスすることで VM の情報を取得したり，管理を実行したりする．

この実装には OpenPegasus [4] を用いた．OpenPegasus は CIM サーバを提供するオープンソース実装で，管理アプリケーションとコンピュータの各資

表 1　主な CIM オペレーション

名称	機能
EnumerateInstances	指定したクラスのすべてのインスタンスを取得する.
EnumerateInstanceNames	指定したクラスのすべてのインスタンス名を取得する.
GetInstance	指定したインスタンスを１つ取得する.
DeleteInstance	指定したインスタンスを１つ削除する.
CreateInstance	指定したインスタンスを新たに１つ作成する.
ModifyInstance	指定したインスタンスのプロパティを修正する.
Associators	指定したオブジェクトと関連のあるオブジェクトをすべて取得する.
AssociatorNames	指定したオブジェクトと関連のあるオブジェクト名をすべて取得する.
References	指定したオブジェクトを参照する関連づけをすべて取得する.

源間の通信を管理する．管理インタフェースとして XML/HTTP 通信間のマッピング，管理情報を保持するリポジトリへのインタフェース，実際にコンピュータの資源にアクセスを行う CIM プロバイダへのインタフェースなど，様々なインタフェースを提供する．

4　CIM プロバイダの作成

OpenPegasus によって CIMOM，リポジトリ，WS-Management サーバは提供されるが，vAMT を実現するには VM に対してアクセスを行う CIM プロバイダを作成する必要がある．CIM プロバイダの作成には，CIMPLE [5] というツールを利用した．CIMPLE を用いることで，CIM クラスが定義されている MOF から CIM プロバイダの雛形を生成することができる．

4.1　CIMPLE

CIMPLE は CIM クラスに対応する C++のクラスと CIM プロバイダの雛形となるクラスを生成する．前者のクラスは CIM クラスのプロパティに対応するメンバを持つ．後者のクラスにはメンバ関数として enum_instances や get_instance などが定義されている．例えば，enum_instances 関数は CIM オペレーションの EnumerateInstances が実行される時に呼び出される．図 4 の CIM_Processor のための CIM プロバイダに enum_instances 関数を記述する例を図 6 に示す．この例では CIM_Processor の create メソッドを用いて CPU の数だけインスタンスを作成し，各インスタンスに CPU 番号を設定している．初期化を終えたインスタンスはハンドラに登録することで要求元に送られる．

```
Enum_Instances_Status
CIM_Processor_Provider::enum_instances(
  const CIM_Processor* model,
  Enum_Handler<CIM_Processor>* handler)
{
  for (i = 0; i < nCPUs; i++) {
    CIM_Processor *cpu =
      CIM_Processor::create();
    cpu->Number.set(i);
    handler->handle(cpu);
  }
  return ENUM_INSTANCES_OK;
}
```

図 6　enum_instances の記述例

get_instance 関数にはその CIM クラス中の 1 つのインスタンスを返すように記述する．get_instance の記述例を図 7 に示す．要求されたインスタンスの持つプロパティの値が引数 model に格納されており，一致するインスタンスが存在する場合にはそのインスタンスを返す．この例では，要求された CPU 番号を持つ CPU が存在すれば対応するインスタンスを返している．CIMPLE では get_instance 関数が定義されていない場合，enum_instances 関数を用いて get_instance 関数の機能をエミュレートすることができる．

```
Get_Instance_Status
CIM_Processor_Provider::get_instance(
  const CIM_Processor* model,
  CIM_Processor*& instance)
{
  int val = model->Number.value;
  if (val >= 0 && val < nCPUs){
    instance->Number.set(val);
    return GET_INSTANCE_OK;
  }
  return GET_INSTANCE_NOT_FOUND;
}
```

図 7　get_instance の記述例

```
Invoke_Method_Status
CIM_Processor_Provider::Enable(
  const CIM_Processor* self,
  const Property<boolean>& Enabled,
  Property<uint32>& return_value)
{
  //ここに処理を記述
  return INVOKE_METHOD_OK;
}
```

図 8　Invoke Method の記述例

図 8 は CIM_Processor の Enable メソッドの記述例である．引数 self で対象となるインスタンスが指定され，Enabled で引数が渡される．この関数の中に必要な処理を記述し，引数 return_value に返り値を設定することで CIM クラスのメソッドを実現できる．

図 4 の CIM_Realizes の CIM プロバイダに enum_instances 関数を記述する例を図 9 に示す．まず，関連づけを行う CIM_Chip と CIM_Processor のインスタンスをそれぞれ作成し，キープロパティに値を代入して初期化を行う．次に，CIM_Realizes のインスタンスを作成し，CIM_Chip と CIM_Processor のインスタンスをメンバの Antecedent と Dependent に代入する．この時，それぞれを CIM_PhysicalElement と CIM_LogicalDevice にキャストしている．

```
Enum_Instances_Status
CIM_Realizes_Provider::enum_instances(
  const CIM_Realizes* model,
  Enum_Instances_Handler<CIM_Realizes>*
  handler)
{
  CIM_Chip* chip =
    CIM_Chip::create(true);
  chip->Tag.set("CPU 0");

  CIM_Processor* cpu =
    CIM_Processor::create(true);
  cpu->Number.set(0);

  CIM_Realizes* link =
    CIM_Realizes::create(true);
  link->Antecedent =
    cast<CIM_PhysicalElement*>(chip);
  link->Dependent =
    cast<CIM_LogicalDevice*>(cpu);
  handler->handle(link);
  return ENUM_INSTANCES_OK;
}
```

図 9　関連づけの記述例

4.2　作成した CIM プロバイダ

これまでに vAMT 用に CIM プロバイダを作成した CIM クラスを表 2 にまとめた．

4.2.1　バージョン情報の取得

管理ツールは CIM_SoftwareIdentity クラスを用いて AMT のバージョンを取得している．この CIM クラスには様々な情報を返すインスタンスが存在するが，その中の 1 つのインスタンスが AMT の情報を保持している．そこで，CIM プロバイダの get_instance メソッドにおいて，リクエストされた InstanceID プロパティの値が “AMT” の場合に vAMT のバージョンを返すようにした．AMT はバージョンによって機能が異なるため，最新バージョンを返すようにした．

表 2　vAMT に対応した CIM クラス

クラス名	説明
CIM_SoftwareIdentity	インストールの依存性管理用のソフトウェア情報を持つ.
CIM_AssociatedPowerManagementService	システム要素と電源管理情報の対応づけを行う.
CIM_ComputerSystem	システム情報を持つ.
CIM_PowerManagementService	電源管理のための操作を行う.
CIM_BIOSElement	BIOS 情報を持つ.
CIM_Chassis	マシンのメーカやモデル名などの製品情報を持つ.
CIM_ComputerSystemPackage	各デバイスとシステム情報の対応づけを行う.

4.2.2　電源の制御

vAMT を用いて VM の電源制御を行うには，3 つの CIM クラスが用いられる．まず，`CIM_AssociatedPowerManagementService` クラスのプロパティである `PowerState` の値を取得することで，VM の電源状態を調べる．次に，電源操作を行う VM のシステム情報を `CIM_ComputerSystem` クラスのインスタンスから取得する．最後に，取得したシステム情報を持つ VM の電源操作を行うために，`CIM_PowerManagementService` クラスの `RequestPowerStateChange` メソッドを呼び出す．

VM の情報を取得したり，電源を操作したりするのには libvirt [6] というライブラリを用いた．libvirt は様々な仮想化ソフトウェア上で動作している VM を統一的に扱うことを可能にしている．libvirt を用いて VM にアクセスすることで，異なる仮想化ソフトウェアに対して 1 つの CIM プロバイダで対応できる．VM の電源状態を調べるには `virDomainIsActive` 関数を使用した．この関数は VM を指定して呼び出すことで，電源がオンの時に 1，オフの時に 0 を返す．また，VM の電源を入れる時に `virDomainCreate` 関数，電源を切る時に `virDomainShutdown` 関数を使用した．

4.2.3　AssetDisplay

既存の管理ツールを用いて管理を実行するために，ツールが使用している CIM クラスについて調べた．AMT 対応の管理ツールは様々だが，高度なものほど管理を開始する前に AMT を持つ PC を検出する段階で数多くの CIM クラスを使用して情報を取得している．そこで比較的簡素な AMT の SDK に含まれる管理コマンドへの対応を行った．

管理コマンドの 1 つである AssetDisplay はシステムのハードウェア情報を取得するコマンドであり，オプションで取得する情報を切り替えることができる．例えば，VM のシステム情報を取得するには `CIM_BIOSElement`，`CIM_Chassis`，`CIM_ComputerSystemPackage` という 3 つの CIM クラスが用いられる．これらはそれぞれ BIOS 情報，製造メーカやモデル名の情報，GUID を取得するためのものである．VM の場合は物理的なハードウェアがないため，製造メーカやモデル名は仮想化ソフトウェアの製造元や種類を取得するようにした．

5　実験

作成した CIM プロバイダの動作確認のために，Windows に付属する汎用的な管理ツールである WinRM を用いて AMT のバージョンを取得する get コマンドを vAMT に対して実行した．コマンドの引数として，CIM クラス名を `CIM_SoftwareIdentity`，キープロパティである `InstanceID` の値を “AMT” と指定した．その結果，図 10 のように AMT に対して実行した時とほぼ同様の情報を取得することができた．

また，既存の AMT 対応管理ツールから VM の管理を実行できることを確認するために，SDK の AssetDisplay コマンドを vAMT に対して実行した．AMT の場合は PC 本体のメーカやモデル名が表示されるのに対して，vAMT の場合は図 11 のように VM の製造元や種類が表示されることを確認した．これにより，既存の管理ツールによって vAMT を用いた管

```
PS C:¥Windows¥system32> winrm get cimv2/CIM_SoftwareIdentity?InstanceID=AMT -r:http://192.168.0.71:16992/wsman
                        -a:None -encoding:utf-8
CIM_SoftwareIdentity
    InstanceID = AMT
    IsEntity = true
    IsLargeBuildNumber = false
    VersionString = 7.1.4
```

図 10　バージョン情報取得コマンドの実行結果

```
>AssetDisplay -computersys -host 192.168.0.71 -user oozono -pass qx

-----------------------------------------
Enumerating types = ComputerSystem
-----------------------------------------
BIOS Version: SeaBIOS
-----------------------------------------

 Item #0
------------------------------
Manufacturer: Red Hat,Inc.
Model: KVM
Serial Number: 001
Version: 01
Platform GUID: 23858844180679538053B9C04DB97531
```

図 11　AssetDisplay の実行結果

理が行えることが分かった．

次に，AMT のバージョンを取得するコマンドを vAMT と AMT に対して実行し，その処理時間を比較した．AMT の場合は電源がオンの状態とオフの状態で測定した．図 12 に測定結果を示す．その結果，vAMT の処理時間は AMT よりも短いことが分かった．これは vAMT を搭載している PC 本体の CPU に比べて AMT の性能が低いためと考えられる．一方，AMT は電源がオフの場合，1 回目は非常に長い時間がかかるが，2 回目以降や電源がオンの場合には処理時間は改善されている．

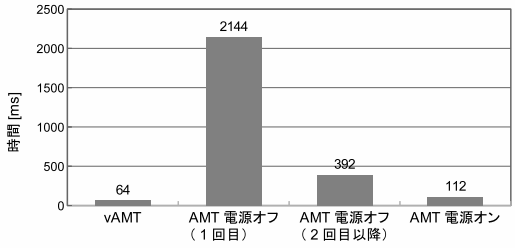

図 12　情報取得にかかる時間

6　関連研究

QND Plus [7] に代表されるソフトウェアベースの管理ツールを用いることで PC と VM を一元的に管理することができる．AMT を搭載した PC の管理にも対応しており，QND Plus を用いて AMT の機能を使用することができる．ただし，QND Plus は管理エージェントの停止時や管理される PC の電源がオフの時には管理を行うことができない．

Virt-manager [8] は libvirt を用いて実装された VM の管理ツールである．そのため，Xen や KVM，QEMU などの異なる仮想化ソフトウェアの VM を一括して管理することができる．Virt-manager は VM を外から管理するため，VM 内で管理エージェントを動作させる必要がない．しかし，Virt-manager では

PC の管理を行うことはできない．

IPMI [9] は CPU や OS に依存することなくハードウェアを管理するためのサーバ向けの管理インタフェースである．AMT と同様に，サーバの温度や電源，ファンの状態などを監視できる．また，システムがダウンしている場合でもサーバシステムの状態をチェックしたり，管理機能を持つ別のサーバを接続してサーバを監視したりする機能を持っている．IPMI はサーバ向けであり，PC には搭載されていないため，PC と仮想デスクトップの一元管理には用いることができない．

VM の管理を行えるようにするために，仮想化に対応した CIM [10] も定義されている．この CIM を用いることで物理サーバとその上で動作する VM を１つの管理ツールで一元的に管理することができる．しかし．仮想化対応の CIM は仮想化 AMT の規格には含まれていない．また，物理サーバと VM を意識せずに扱うこともできない．

7 まとめ

本論文では，VM を管理するための仮想的な AMT である vAMT を提案した．vAMT が AMT と同様のインタフェースを提供することによって，既存の管理ツールを用いて PC と VM を一元的に管理することができる．OpenPegasus を用いて vAMT のシステムを構築し，VM を管理するためのいくつかの CIM プロバイダを作成した．既存の AMT 対応の管理ツールから vAMT に対してコマンドを実行して AMT の場合と同様の結果を得ることができた．これにより，vAMT を用いて VM の管理を行えることを確認した．

今後の課題は，AMT の基本機能の中の障害回復機能を実現できるようにすることである．その第一段階として，VM に対して VNC 接続を行うのに必要な CIM プロバイダを実装して，管理者側からリモートで VM の操作を行えるようにする．

謝辞

本研究の一部は，科学技術振興機構 戦略的創造研究推進事業 (CREST) 研究領域「実用化を目指した組み込みシステム用ディペンダブル・オペレーティングシステム」による．

参考文献

[1] DMTF. System Management BIOS （SMBIOS） Reference Specification Version 2.7.1. 2011.
[2] DMTF. Web Services for Management (WS-Management) Specification Version 1.1.1. 2012.
[3] DMTF. Common Information Model (CIM) Infrastructure Version 2.7.0. 2012.
[4] The Open Group. OpenPegasus, http://www.openpegasus.org/.
[5] K. Schopmeyer and M. Brasher. CIMPLE, http://simplewbem.org/.
[6] Red Hat. libvirt, http://libvirt.org/.
[7] QualitySoft. QND Plus, http://www.quality.co.jp/
[8] Red Hat. Virt-manager, http://virt-manager.org/
[9] Intel, Hewlett-Packard, NEC, and Dell. Intelligent Platform Management Specification Second Generation v2.0. 2004.
[10] DMTF. CIM System Virtualization Model Version 1.0.0. 2007.