仮想 AMT を用いた仮想マシンと PC の一元管理

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 Intel 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．

As PCs are being used at every department, the number of PCs that administrators have to manage is increasing. Administrators sometimes go to the location of PCs to repair them on failures. To reduce such a burden, the latest PCs are equipped with Intel AMT. AMT enables administrators to remotely manage PCs in a unified fashion. However, due to the prevalence of virtual desktops, PCs and virtual desktops intermingle inside an organization in recent years. Since AMT can manage only PCs, virtual desktops have to be managed independently. In this paper, we propose vAMT, which is virtual AMT for a virtual machine (VM), to enable unified management of virtual desktops and PCs. vAMT manages a VM by providing the same interface as AMT does for a physical machine.

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 Intel 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

Unified Management of Virtual Machines and PCs Using Virtual AMT.
Kouki Oozono, Kenichi Kourai, 九州工業大学, Kyushu Institute of Technology.

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 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 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．最後に 6 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

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，保護がある．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．

障害回復機能としては，PC に接続されているキーボードやマウス，PC の電源をリモートから操作することができる．AMT 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．OS が起動しない場合は，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．

保護機能としては，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，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，NIC の制御を行うことでネットワークへの接続を制限したりすることができる．

近年，仮想デスクトップと呼ばれる利用形態が普及してきている．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 1 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．

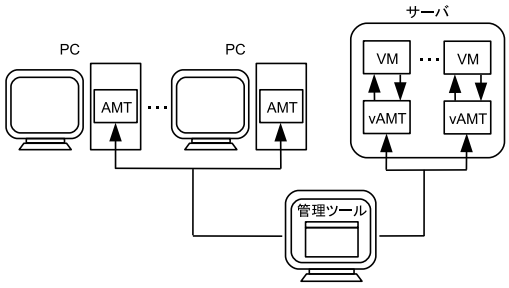

図 1　システムの全体構成

3.1　CIM インタフェース

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 2 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，説明を簡単にするために，図 2 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ある．`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．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．

3.2　その他のインタフェース

VNC 接続を行うことで，リモートから VM の画面を確認したりキーボードやマウスを操作したりすることができる．AMT を経由した VNC 接続にはデフォルトポートを使用する方法とリダイレクションポートを使用する方法の 2 種類がある．デフォルトポートを使用すると，通常の VNC 接続を行うことができる．リダイレクションポートを使用すると，AMT 独自のプロトコルを用いて TLS によるセキュリティの高い接続を行える．

```
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;
};
```

図 2　MOF で記述された CIM クラスの例

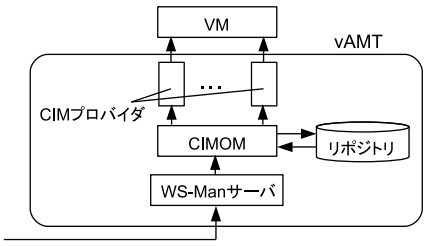

図 3　vAMT の構成

4　実装

4.1　システム構成

vAMT は図 3 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 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 [6] 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．OpenPegasus で vAMT へのリクエストを処理できるようにするには 2 つの修正を行う必要があった．1 つは名前空間が指定されていないときにデフォルトの名前空間を使うようにすることであり，もう 1 つは参照先のアドレス指定が “default” のときに CIM サーバを指すようにすることである．

現在の実装では，仮想化ソフトウェアとして Linux KVM を対象としている．vAMT はホスト OS 上で動作し，同じくホスト OS 上で動作する VM に対して管理を行う．

4.2　CIM プロバイダ

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．

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．我々は AMT で使用される CIM クラスの MOF から全ての CIM プロバイダの雛形を生成した．その際に，AMT 用に文法拡張された CIM クラスが存在したため，CIMPLE の文法チェックを修正した．

図 2 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 4 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

```
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 4　enum_instances の記述例

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．

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 5 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．enum_instances や get_instance によって対応する CIM クラスが持つインスタンスを返すために用いられる CIM プロバイダをインスタンスプロバイダと呼ぶ．

図 6 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 2 の CIM_Realizes の CIM プロバイダに enum_instances 関数を記述する例を図 7 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

```
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 5　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 6　Invoke Method の記述例

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．このように，関連付けを行うために用いられる CIM プロバイダを関連プロバイダと呼ぶ．

4.3　開発した CIM プロバイダ

これまでにいくつかの vAMT 用の CIM プロバイダを作成し，以下の機能を実現した．

- 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．そこで，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．
- 電源の制御

```
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 7　関連づけの記述例

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し，CIM_PowerManagementService クラスの RequestPowerStateChange メソッドによって，取得した情報を持つ VM の電源操作を行う．VM の電源操作は libvirt [4] を用いて行った．

- AssetDisplay

  AssetDisplay は AMT の SDK に含まれる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．VM の場合は物理的なハードウェアがないため，例えば，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．システム情報を取得するために 3 つのインスタンスプロバイダ，CPU 情報を取得するために 3 つのインスタンスプロバイダと 2 つの関連プロバイダをそれぞれ作成した．

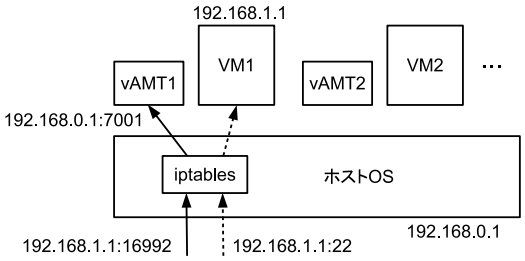

図 8　NAPT による変換

4.4　vAMT へのアクセス

リモートの管理ツールが AMT にアクセスする時には，AMT が搭載された PC の IP アドレスを指定して接続を行う．同様に，vAMT にアクセスする時には，対応する VM の IP アドレスを指定して接続を行えるようにすべきである．しかし，vAMT はホスト OS 上で動作しているため，ホスト OS の IP アドレスを指定しなければアクセスすることができない．また，同一ホスト OS 上で複数の vAMT が動作するため，それらすべてが標準の 16992 番ポートを用いることはできない．

そこで，図 8 のようにホスト OS で静的 NAPT の設定を行い，vAMT へのアクセスを転送する．VM の IP アドレスに対する 16992 番ポート宛てのパケットは，ホスト OS の IP アドレスと vAMT が用いるポート番号に変換する．同様に，VNC 接続についても，VM の 5900 番ポートへのアクセスはホスト OS の vAMT のポート番号に変換する．これにより，管理ツールは AMT と同じアドレス，ポートの指定方法で vAMT にアクセスすることができる．

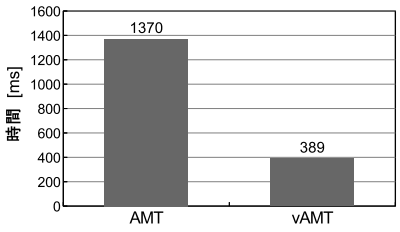

図 9　CPU 情報の取得にかかる時間

5　実験

AMT SDK の AssetDisplay を AMT と vAMT に対して実行した．CPU 情報を取得する場合，3 種類のインスタンスプロバイダと 2 種類の関連プロバイダが使用される．実行の結果，AMT と同じように vAMT から CPU 情報を取得することができた．

次に，CPU 情報を取得するのにかかる時間を比較した．図 9 に測定結果を示す．通常の AMT ではリクエストを送り始めてから情報を取得し終えるまでに 1370ms かかったのに対して，vAMT では 389ms しかかからなかった．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．

6　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．AMT と同様のインタフェースを vAMT にも持たせることによって，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，CIMPLE を用いて 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 の基本機能の中の保護機能を実現できるようにすることである．そのためにまず，ネットワークの設定を行うのに必要な CIM プロバイダを実装して，管理者側からリモートでパケットフィルタリング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] Red Hat: libvirt, http://libvirt.org/.
[5] Schopmeyer, K and Brasher, M.: CIMPLE, http://simplewbem.org/.
[6] The Open Group: OpenPegasus, http://www.openpegasus.org/.