Bottom-up harmonisation of management attributes describing hypervisors and virtual machines

Vitalian A. Danciu[1] danciu@mnm-team.org
Nils gentschen Felde[1] felde@mnm-team.org
Michael Kasch[2] kasch@cip.ifi.lmu.de
Martin G. Metzker[1] metzker@mnm-team.org

[1] Munich Network Management (MNM) Team
Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, Germany

[2] Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, Germany

***Abstract*— The advent of host virtualization has increased the number of management attribute classes and instances. At the same time an additional degree of heterogeneity has been introduced, due to different hypervisor products coupled with multiple guest operating systems. These changes obviate provisionary methods of harmonising management information.**

We analyse the problem dimensions of attribute harmonisation according to a common management scenario and show why heterogeneity at hypervisor and VM level is difficult to deal with at present. In response, we present a classification of bottom-up attribute matching patterns and propose a methodology for the systematic processing of management attributes. As a proof-of-concept, we describe our implementation of an attribute normalising framework extending the libvirt library.

Keywords: management attribute, heterogeneity, virtual machine, hypervisor

I. Introduction

Management attributes are the foundation of any management operation, be it in virtualized infrastructure, or in traditional service provisioning based purely on physical systems. Without current information about the managed objects, their aggregations, and the system they compose, sensible and useful management actions cannot be executed.

Yet, even very similar objects are described by their creators with attribute sets that differ in syntax, value ranges, semantics, and the manner in which they can be accessed. Management integration requires us to bridge this heterogeneity in order to create a consistent view on the information base.

Two fundamental avenues of approach lend themselves to this task:

1) the top-down specification of attributes, that should be implemented by all vendors and products, in order to adhere to the common specification
2) the bottom-up analysis of existing management information, and the creation of a normalisation layer that "translates" between the heterogeneous attribute sets

[0]Authors are listed in alphabetical order.

978-1-4577-1811-3/11/$26.00 © 2011 IEEE

With respect to effectiveness, both of these strategies carry their risks: the bottom-up approach might be plagued by frequent changes in products and in the arrangement of the provided management information with respect to a given object class; the top-down approach may be implemented in a sketchy manner, and be merely declared to have been respected, in order to stipulate adherence to a standard.

We have examined basic attribute sets of common host virtualization products, and found that, while they do provide useful management information, this management information is presented in different forms, on different value scales, distributed among several attributes or consolidated into a single value, and accessible by means of different protocols and programming interfaces.

An examination of adherence to a common representation shows, indeed, that similarity alone is insufficient in practise. On the other hand, the common abstraction mechanisms (libraries, management system plug-ins) constrain themselves to equalise the *method* of access to management attributes and function on different systems, but fall short of providing true normalisation mechanisms.

A. Increase in the number of inter-related attributes

Before virtualization changed the data centre landscape, management attributes of hosts were solely associated to the hardware, and to the single operating system running on each given machine. Due to the introduction of hypervisors, and in consequence of the possibility to operate multiple, different OSs on each machine, the number of accessible attribute instances (values) has grown significantly. At the same time, the introduction of a virtualization layer has increased the number of attribute classes, as well.

Any improvised approach to the harmonisation of attributes across different systems and vendors is increasingly impaired in its effectiveness. We require a systematic approach to the classification of attributes in order to enable the normalisation of their values for the benefit of management and operations. Thus, a practical solution should account for all harmonisation dimensions, while taking into consideration as many of their facets as possible.

B. Contribution and structure

On the basis of fundamental attributes of the VMware ESXi and Xen hypervisors, we examine the requirements on normalisation of attributes, develop a classification scheme and present a generic normalisation extension for *libvirt* (a well-known access library for hypervisors) that allows access to hypervisor and VM attributes in a consistent manner, with common value metrics, and common semantics.

We detail our problem statement and stipulate the requirements on the normalisation procedure and software in Section II. In accordance to our analysis of a set of selected attributes, we propose a classification scheme instrumental for normalisation, as well as a methodology for the classification and normalisation of attributes, in Section IV. For a small set of example attributes, we discuss an architectural extension of *libvirt* in Section V, that allows normalised access to these attributes. We discuss and evaluate this solution from a conceptual as well as a practical perspective before concluding the paper in Section VI.

II. Problem analysis

The problem at hand is illustrated by means of a typical management scenario, serving as a basis of the detailed analysis of exemplary attributes. Based on the differences encountered, we formulate the dimensions of our problem.

A. Scenario

Consider the exemplary use-case of a teaching environment illustrated in Figure 1, that comprises two virtualized compute clusters running the VMware and Xen hypervisors, respectively.

The Xen-based cluster is managed by Eucalyptus to offer cloud-like infrastructure services. The VMs operated in this cluster are intended for short term use and perform, in general, very compute-intense operations. On the other hand, the VMware cluster comprises less resource-demanding VMs, that typically function as web servers, CVS repositories etc. for longer-term projects.

From an administrative point of view, the management of the two clusters does not allow for any synergies, as every cluster requires its own management procedures and tools. Consequently, the same work to keep a clusters operational needs to be carried out using two completely different management models and front-ends. Obviously, harmonization of the clusters from a management perspective—while keeping the internal separation of VMs (namely the compute-intense, short-lived VMs versus the longer lasting, but less demanding VMs— is highly desirable.

Equivalent management information is fundamental to any consolidation of management effort. Thus, to harmonise the management of the two virtualization-enabled clusters, one has to take a closer look at the management attributes of their respective virtualization layer. However, even basic operations and information representations like VM power states, virtual machines' or the hosts' main memory differ from one vendor to the other. Section II-B provides a more in-depth examination of these examples.

A unified management view is achievable in two principal ways (top-down and bottom-up), supported by different frameworks and pre-existing resources. Top-down data models like the *Common Information Model* (CIM, also refer to Section III-A for more detailed information) try to abstract generic scenarios and build a foundation to suit every given use-case. To accommodate the broadest possible range of products, the CIM (and every other model attempting specification down to the attribute level) is obliged leave a lot of degrees of freedom to the hypervisor implementers; this, in turn, may entail non-interoperability among some CIM-compliant implementations, as a consequence of different use of those degrees of freedom.

On the other hand, bottom-up data models as in this scenario given by the VMware ESXi and Xen hypervisors are developed in isolation from each other. Thus, their harmonisation must take place externally, in the context of the (integrated management) system, while no changes at the hypervisors' level are necessary. A possible architecture that facilitates this approach is for example given by the *Service Monitoring Architecture* (SMONA, refer to Section III-A.3 for more details).

B. Attribute examples

We have analysed a number of significant management attributes that should be part of an integrated management view in the scenario with respect to their semantics and syntax. We juxtaposed the attribute definition for the Xen hypervisor with that of VMware's, but also with the corresponding definition given by the Common Information Model (CIM). The following two examples demonstrate similarities but also significant differences in the presentation and semantics of the attributes.

1) Virtual machine power state: Our first example is the power state of a virtual machine. This is an attribute we expect every hypervisor to have.

The CIM models many power states as optional and supports an vendor defined state to match many hypervisors. The modelled states are as following [18] (optional states are denoted by *):

- *defined:* does not consume any resources of the virtualization platform, except persistent storage. Virtual machine is inoperable (not able to perform tasks)
- *active:* Virtual Machine is instantiated at the virtualization platform and in general able to perform tasks. But some virtual resources may not be enabled to perform tasks for various reasons, for example given a missing resource allocation.
- *paused *:* Virtual Machine is not able to perform tasks, but resources and resource allocations remain as in the active state.
- *suspended *:* Virtual Machine state and resources are

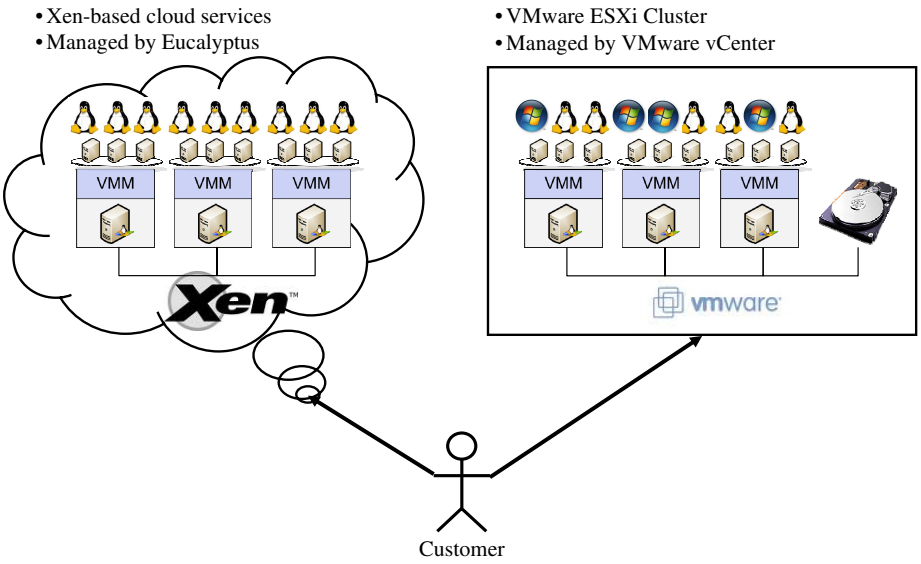


Fig. 1. A use-case scenario as given at University

saved on non-volatile storage.

- *Vendor Defined *:* The CIM gives the possibility of defining further vendor defined states.
- *unknown *:* Pseudo-state — tells that the state cannot be determined, e.g. there is no connection to the virtualization platform because of networking problems

In Xen the power state of a virtual machine has an data-type of enum vm_power_state. On the wire format the enum is represented by the string identifier of the enum element. The enum lists the following possibilities [20]:

- *Halted:* The virtual machine is not executed
- *Crashed:* Hypervisor detected a crashed guest OS; Virtual Machine has to be transitioned into the halted state
- *Paused:* Virtual machine is paused, machine state is hold in the host's working memory
- *Running:* Virtual machine is running
- *Suspended:* Virtual machine is paused, system state is stored on background storage
- *Unknown:* Some other unknown state

A VMware ESXi Server exposes a VirtualMachinePowerState enumeration type with three possible states, via the VI-SDK to the manager:

- *poweredOn:* Virtual Machine is powered on
- *poweredOff:* Virtual Machine is powered off
- *suspended:* Virtual machine is suspended, which means that its execution is paused and the machine's states are saved on a datastore

The vendor defined state in the CIM gives the possibility to match every single hypervisor, for example Xen's crashed state. The underlying issue is that these values represent different vendors' state machine models for VM status. Thus, a mapping of the attribute entails the establishment of semantic equivalence between those state machines.

2) Virtual machine and host memory: Another example for attributes in an virtualization environment are the total and free host memory and the total and free virtual machine memory.

In the CIM the total virtual machine memory is modelled by a CIM_Memory class and can be a composition of more than one memory size attribute modelled by the same class. This class has among others the following properties quoted from [15]:

- *StartingAddress: Starting address of an memory extend, in the case of total virtual machine / host memory this shall be 0.*
- *EndingAddress: Ending address of the highest memory extend compositing this memory slice*
- *BlockSize: Size of a memory block*
- *NumberOfBlocks: Number of memory blocks*
- *ConsumableBlocks: Number of available memory blocks, shall be 0 if unknown*

Units shall be kilobyte ($byte * 2^{10}$).

Neither Xen nor VMware expose information about memory sizes via their SDKs. Instead, some memory attributes describe total and free memory of hosts and virtual machines.

Xen exposes the following attributes for host memory via the host_metrics class:

- $memory_{total}$: total memory of the host, in bytes
- $memory_{free}$: free memory on the host system, in bytes

and among others the following attribute for virtual machine memory via the VM_metrics class:

- $memory_{actual}$: memory actually assigned to a virtual machine's guest OS in bytes

VMware exposes the following attributes concerning host memory:

- *memorySize:* Total size of host memory in bytes, for

example exposed via the HostHardwareInfo object
- *overallMemoryUsage:* Actual consumed memory in megabytes, exposed via the HostListSummaryQuickStats object

and among many others the following attribute concerning virtual machine memory:

- *memorySizeMB:* Configured memory size in megabytes, exposed via the VirtualMachineConfigSummary object.

It is obvious, that mapping these attributes is not trivial. A detailed account of the aspects pertaining to such mappings is given in the following Section II-C.

C. Dimensions of harmonisation

Attributes constitute mere representations of management information, and they differ in the manner in which they represent the same piece of information. The differences may be syntactic, semantic, related to the access to their values. Their harmonisation, as a prerequisite to the normalisation of attribute values, must cover all three dimensions in order to be effective.

1) Semantic differences: Attributes can be different in their semantics, even though they seem to represent the same item of management information.

A simple example is an attribute describing the size of a hard drive in megabytes: given only a value, this value may represent the size

- in units of 2^{20} bytes of raw disk capacity, i.e. before partitioning an file-system creation
- in units of 10^6 bytes of raw disk capacity
- after partitioning, subtracting some space, either in powers of two or ten
- after partitioning, available in a created file-system, again, in powers of two or ten
- available in a file-system, minus an arbitrary space reserved by the operating system

Certainly, one might point out that "megabytes" represent values in powers of ten, and that there is a proposed convention to call the power of two approximation "mebibyte" [1]. However, there is nothing to compel an attribute provider to adhere to such conventions. Short of a precise definition of attribute semantics, it is impossible to ascertain their exact meaning from their names or from their context.

This issue is exacerbated when the semantics of an attribute not only needs to be understood, but values of (seemingly) the same attribute originating from different providers need to be compared, on the same scale of units, assuming the same value bounds.

In addition to the *meaning* of the attribute, we must thus define:

- the bounds and the scale of its values,
- its dependence on other attributes, and
- the unit of its values.

2) Syntactic differences: Attributes that carry the exact same meaning can still obviate their own effective use in management by differing syntactically, i.e. in aspects pertaining to form. They are sketched in the following.

a) Names and namespaces: attributes can differ in their own names, as well as in the namespace within which they may be referenced. While the name signifies the attribute itself, its namespace states the context in which it has meaning. Often, the namespace can be mapped to an object representing the owner of the attribute. For example, VMware's attribute *memorySize* represents by this identifier the total memory size, while its namespace, the class *HostListSummaryQuickStats* denotes the context of the value to be with respect to the physical host.

b) Data structure and attribute position: Attribute values are organised in data structures that may be scalar or compound. For example, the size of a storage unit may be given as "100" or, at the whim of an API designer, as "$\{100\}$". Potential compound data structures include vectors and lists, as well as heterogeneous structures and objects.

While this simple case will not impede the use of the attribute much, in some more difficult cases, attribute values are arranged in an array, to be retrieved via the correct index value. Obviously, the *position* of the attribute value within a data structure can differ among implementations.

c) Data type and precision: The data type of an attribute is to some extent given by the quantity it represents, however it can differ in representation: in our example, the size of the storage unit may be represented by an integer, or a double precision integer, or by any variant of floating point values.

Thus, the use of such attribute values must take into the account the data type and, consequently its syntax. In some particular cases (e.g., for timer values), the *precision* of the value may be significant as well (e.g. float, double).

3) Access-related differences: The practical use of management attributes requires effective access to their values. Even attributes equal in semantics and syntax may afford different manners in which their values are retrieved.

a) Access protocols, function calls and parameters: Hypervisor vendors equip their products with different management protocols that may be employed to request attribute values. In the same manner, the programming interfaces they provide are different in the functions they provide, and in the manner in which theses function are used: they may (as the attributes themselves) differ in semantics and syntax, as well as in the formal parameters they require.

b) Access location: Due to some properties of virtual components (e.g. migration of VMs) but also as a side-effect of the quick and flexible configuration of virtualized infrastructure, the location of attributes may vary. This location denotes the position of the object harbouring the attribute values, within the infrastructure.

c) Validity in dependence of context: Some attributes are only valid in specific situations, depending on other non-permanent

properties of the system. For example, it is meaningless to evaluate an attribute describing the interface properties for a network interface not configured in the system. This context of attribute validity may differ among attribute providers.

d) Rate of change: The validity of an attribute value is also dependent on its rate of change, which may differ from one attribute provider to the other.

In this paper, we address the semantic and syntactic differences by means of classification in Section IV. The access-related differences are highly implementation specific. We show how they can be overcome in the discussion of our implementation in Section V.

III. State of the art

In the area of management concepts and their implementation in regard to virtualization, one finds management models on the one hand and management interfaces and their implementations on the other hand. In the following, we discuss a number of relevant representatives of these conceptual resources.

A. Management models

In the context of this paper, resources focusing on attributes per se are relevant, as well as those with specific reference to virtualization. Section III-A.1 discusses the *Virtualization Management* working group's popular *Open Virtualisation Format* (OVF), Section III-A.2 gives an overview of the relevant *Common Information Model Profiles* while Section III-A.3 takes a closer look at the *Service Monitoring Architecture* (SMONA).

1) Virtualization Management Initiative (VMAN): The Virtualization Management Initiative's work is an extension to the Common Information Model. VMAN aims at managing the entire lifecycle of virtual appliances, starting with the development, packaging and distribution, deployment, management during runtime up to the final retirement of an appliance.

The *Open Virtualisation Format* (OVF) [10] probably is one of the most relevant specifications coming from the VMAN working group. The OVF basically consists of two parts. On the one hand, OVF specifies an XML-based description format for virtual machines (VM). This description contains details about the VM's properties such as amount of main memory, number of network interface cards etc. On the other hand, OVF specifies an open disk image format. This image file contains the images for the VM's hard disk drives, i.e. block devices.

2) CIM Profiles: As the *Common Information Model* (CIM) allows for extensions, the CIM Profiles do specify such extensions. Under the umbrella of the Virtualization Management Initiative, ten relevant CIM Profiles have been specified:

- The *Resource Allocation Profile* [12] on the one hand defines the resource-pool-lifecycle management and relationships, on the other hand it defines the basic resource allocation pattern for resource pools.
- The *System Virtualization Profile* [16] defines an object-oriented model for representing host systems and for discovering virtual machines. In addition, basic management operations on virtual machines as for example their creation, deletion and modification are specified.
- The *Allocation Capabilities Profile* [7] is specified as an extension to referencing profiles and adds the ability to represent the property values for resource allocation requests for a resource.
- The *Processor Resource Virtualization Profile* [11] again is an extension to referencing profiles. It adds the capabilities to describe and manage CPU-like resources to virtual machines.
- The *Memory Resource Virtualization Profile* [9] in accordance to the aforementioned Processor Resource Virtualization Profile, this profile adds the capabilities to describe and manage main memory-resources to virtual machines.
- The *Storage Resource Virtualization Profile* [14] is the third of four profiles extending the possibilities to describe and manage virtual machines' resources. In this case, the management of storage of virtual machines (i.e. virtual hard drives) is targeted.
- The *Ethernet Port Resource Virtualization Profile* [13] as the fourth profile is to represent and manage the allocation of Ethernet ports to virtual machines.
- The *Virtual System Profile* [18] is an autonomous profile. It specifies an object-oriented model to inspect virtual machines and their components. It also specifies a basic set of control operations as for example pausing or suspending a virtual machine.
- The *Generic Device Resource Virtualization Profile* [8] is similar to the Processor, Memory, Storage and Ethernet Port Resource Virtualization Profile. It comes into play when an unusual device type is to be described or managed and no more specific profile exists.
- The *Virtual Ethernet Switch Profile* [17] is an autonomous profile again. It defines an object-oriented model to inspect virtualized Ethernet switches.

In [6] it is concluded that *"there are DMTF schemas concerning the management of virtual machines, but no vendor actually implements them correctly"*.

3) The Service Monitoring Architecture: The *Service Monitoring Architecture* (SMONA) [5], [4], [3] addresses the basic requirement to an architecture supporting synthesis of service management information to reuse existing data sources, such as already deployed management tools. The SMONA defines a 5-layer architecture.

The *resource layer* allocates the resources of relevance to the management tasks and describes the management information available and the interfaces to access them in their "raw" form. The relevant information is then passed on to the *platform specific layer*. This layer represents the existing management tools that are employed to manage the resources located on the resource layer. Management information available on this layer (still vendor-specific) is gathered by adapters (agents) and relayed to the next-higher layer.

The information gathered on the platform specific layer is passed on to the *platform independent layer*. This layer provides unification of the data format and basic configuration options. The adapters are configured by the next higher *integration and configuration layer*, which also comprises a control component, the *rich-event composer*. This composer aggregates and correlates the adapters' data.

The architecture's clients are found on the top-level *application layer*. The clients in general are the management applications. The *Service Information Specification Language* (SISL) allows the specification of the attributes to be queried, the manner of their aggregation and correlation and the conditions governing their provisioning to management applications.

B. Management interfaces and their implementations

In the following, three major vendors' management products for virtualized IT-infrastructure are briefly introduced. Section III-B.1 will focus on the *VMware vSphere API*, Section III-B.2 describes the *XenAPI* in more detail, and Section III-B.3 introduces the *libvirt* programming library.

1) VMware vSphere API: The current management interface to the VMware product line (i.e. VMware vSphere and the VMware hypervisor ESXi) is called *VMware vSphere API*, which sometimes is also referred to as *Virtual Infrastructure SDK* or *VI-SDK* [21]. The VMware vSphere API specifies a Web service that is offered by VMware's hypervisors. It conforms to the *Web Services Interoperability Organization Basic Profile 1.0* and thus makes use of SOAP 1.0, WSDL 1.1 and XML Schema 1.0. The data transport is conducted under the use of HTTP/S and thus may be authenticated and encrypted.

VMware's data model is organized in a tree-like structure. The tree's root is called the *ServiceInstance*. The tree's nodes represent objects, while the leaves hold the attributes and their values. Figure 2 illustrates the tree-like structure. In order to access any managed object, the so-called *Managed Object References* (MOR) are employed. The underlying, object oriented information model implies support for inheritance.

2) XenAPI: The XenAPI [20] – sometimes also referred to as *Xen Management API* – is a library intended for the management of Xen-based host systems over the network. It specifies an interface for configuring and controlling virtual machines remotely, that are executed on a Xen host. The XenAPI has been introduced with Xen in its version 3.0.3 and was officially released with version 3.1.0. The XenAPI exists in two slightly different, but unfortunately not compatible variants: *libxen* for the management of the free/open source Xen, while *libxenserver* comes with Citrix's commercial Xen products.

The XenAPI is specified as a collection of remote procedure calls based on XML-RPC. It grants access to a large variety of management attributes. For this purpose, get()-methods as well as set()-methods are implemented.

The data model is build of a collection of classes which are referenced among each other. Each class then comprises several attributes.

The XML-RPC transfer is performed using HTTP/S and thus authenticated and encrypted, and it allows access to the XenAPI from within every programming language. Among others, there are library implementations for C, Java and Python.

3) libvirt: The libvirt-project provides a library for abstracting the management of host virtualization and to hide the underlying heterogeneity of the hypervisors. libvirt is distributed for several hypervisors, among them Xen, VMware ESXi and GSX, KVM, Microsoft's HyperV and VirtualBox.

Mainly, libvirt focuses on the management of VMs during the execution phase of their life-cycle. Consequently, great parts of libvirt's specification are on operations on the operational state of VMs (e.g. starting, pausing, stopping a VM), but scarce on monitoring and/or manipulating status information and the VMs' configurations.

For describing VMs, libvirt specifies an XML-based data format. These documents always have the same structure, but (probably to conserve implementation effort) libvirt's descriptions of a VM are product-specific in part. In addition, they cannot be shared between different libvirt distribution (as the output is only interpreted correctly by the same driver that created it), which prevents interoperability among the different hypervisors.

IV. Normalisation procedure

Based on our observations with regard to the Xen and VMware hypervisors and management interfaces, we propose a generic procedure for the normalisation of attributes.

A. Methodology for attribute harmonisation

We employ a bottom-up methodology applicable to single attributes in order to facilitate their harmonisation. This methodology has been instrumental in the implementation of normalised variants of the attributes of different hypervisors. It consists of the following main steps:

1) Collect attributes and assess attribute semantics
2) Identify corresponding attributes in all relevant hypervisors.
3) Classify and map attributes
4) Define normalisation function
5) Implement mapping and normalisation

Once a useful attribute has been identified, its meaning is analysed in all relevant hypervisor implementations. The associations between the attribute in one hypervisor and its peers in other hypervisors create pairs of mappings in our classification scheme. After the classifications and mapping have been found, for each pair of hypervisors, normalisation functions for the attribute are defined, according to the mapping class. In our work, these functions are implemented as an extension framework for the libvirt library.

While the normalisation functions ensure the mapping of semantics and syntax, their practical implementation can, in

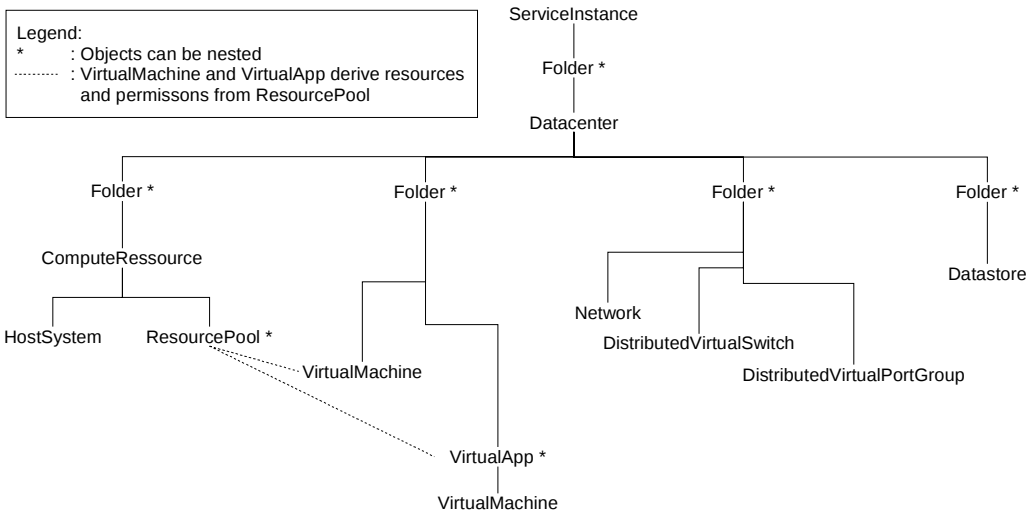


Fig. 2. VMware's tree-like data model [21]

addition, take into account access-related differences between the different representation of an attribute. In the following, we highlight the most important steps of this methodology by example.

B. Classification

As stated in Section II, to harmonise management information and its modification, we need to overcome differences in the syntactic and semantic representations in the underlying information models. As a generic mapping between management attributes could not be found, in this section we propose *attribute mapping classes* as a method of finding semantic matches between a hypervisor specific data model and the CIM. When the mapping class of a CIM attribute is determined, a corresponding projection of hypervisor attributes can be realised.

In a CIM based view onto managed systems, CIM elements' attributes are projections from attributes selected from the hypervisor-specific data model. Hence, the subset of representable CIM attributes is defined by the underlying data model and the correlation of its attributes. Further, normalisation of attributes in terms of syntactic differences is part of the projection. Sensible selections and projections are based on *semantic equivalence* between a CIM attribute and a projection of a set of attributes provided by a hypervisor H. Let A_{CIM} be the host of all CIM attributes and A_H the host of all attributes provided by H. The projection $\pi_{a,H}$ of a selection of attributes $b \in 2^{A_H}$ is semantically equivalent to an attribute $a \in A_{CIM}$, if a deterministic function on b yields a result that semantically matches the specification of a in the CIM. Semantic equivalence of a and $\pi_{a,H}(b)$ is denoted as:

$$a \equiv_s \pi_{a,H}(b).$$

C. Mapping classes

With $b \in 2^{A_H}$ the definition of semantic equivalence allows for multiple $\pi_{a,H}$. For consistent mapping and consideration of all hypervisor attributes, a single $\pi_{a,H}$ for each a must be selected. As demanding that all members of b must be required is not sufficient if there is redundancy in the hypervisor's data model, we define π_H as a set of hypervisor-specific projections $\pi_{a,H}$ as:

$$\begin{aligned}\pi_H := \{\pi_{a,H} : b \mapsto a \mid \\ \exists a \in A_{CIM}, b \in 2^{A_H} : a \equiv_s \pi_{a,H}(b) \quad \wedge \\ \forall c \subseteq b, a \equiv_s \pi_{a,H}(c) : \\ \forall d \subseteq c, a \equiv_s \pi_{a,H}(c) : \\ d = c \quad \}\end{aligned}$$

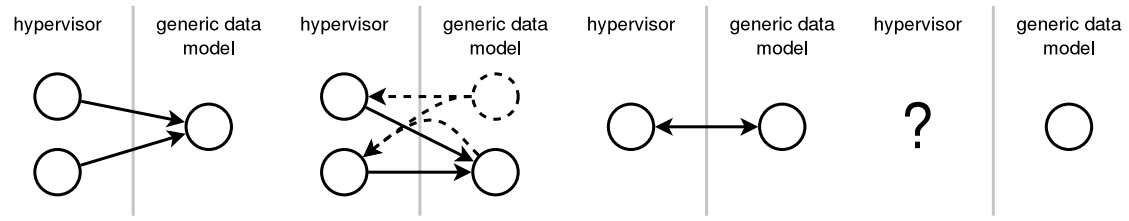
Consequently, the host of CIM attributes representable by a specific hypervisor $A_{CIM,H}$ is:

$$A_{CIM,H} := \{a \in A_{CIM} \mid \exists \pi_{a,H} \in \pi_H\}$$

We have identified five projection classes of $\pi_{a,H}$, based on the calculation effort required to adequately express CIM elements: *n:1 mappable attributes*, *n:1 invertible mappable attributes*, *1:1 bijective mappable attributes* and *unmappable attributes*.

1) n:1 mappable attributes: An Attribute a is n:1 mappable, if there exists a selection $b \in 2^{A,H}$ and a projection $\pi_{a,H}$ in such a way, that the projection of b is semantically equivalent to a. By definition $A_{CIM,H}$ are the n:1 mappable attributes.

2) n:1 invertable mappable attributes: For supporting manipulation of a CIM attribute a by a management system, the projection $\pi_{a,H}$ must be invertible to propagate a change to a back to the hypervisor's managed objects. This is not a general property of projections in π_H. If such an inversion of $\pi_{a,H}$



(a) n:1 mappable (b) n:1 invertable mappable (c) 1:1 bijective mappable (d) Unmappable attribute

Fig. 3. Attribute matching classes

exists, a is a n:1 invertable mappable attribute, $a \in I_{CIM,H}$. Formally $I_{CIM,H}$ is defined as:

$$\begin{aligned} I_{CIM,H} = \{a \in A_{CIM} \mid \\ \exists \pi_{a,H} \in \pi_H : a \equiv_s \pi_{a,H}(b) \quad \wedge \\ \exists i \in 2^{A_{CIM,H}} : \exists \pi_{a,H}^{-1} : i \mapsto b \quad \} \end{aligned}$$

It's trivial to prove that $I_{CIM,H} \subseteq A_{CIM,H}$ applies.

3) 1:1 bijectively mappable attributes: Attribute a, mappable selecting only one hypervisor attribute so that $|b| = 1$, is a 1:1 bijectively mappable attribute $a \in B_{CIM,H}$. In this special case either $a \equiv_s b$ and no projection is required, or $\pi_{a,H}$ is bijective and the attribute mapping is possible mutually.

$$\begin{aligned} B_{CIM,H} = \{a \in A_{CIM} \mid \\ \exists \pi_{a,H} : a \equiv_s \pi_{a,H}(b) \quad \wedge \\ |b| = 1 \quad \wedge \\ \pi_{a,H} \text{ is bijective} \quad \} \end{aligned}$$

Let $a \in B_{CIM,H}$, then there exists a bijective function $\pi_{a,H} : b \mapsto a$, so there is a subset of $2^{A_{CIM,H}}$ namely a, and a function $\pi_{a,H}^{-1}$ so that $a = \pi_{a,H}^{-1}(b)$. Therefore applies

$$B_{CIM,H} \subseteq I_{CIM,H} \subseteq A_{CIM,H}.$$

4) Unmappable attributes: Unmappable attributes $U_{CIM,H}$ are CIM attributes not representable by a specific hypervisor H.

$$U_{CIM,H} = A_{CIM} \backslash A_{CIM,H}$$

D. Normalising projections

Information required to represent a CIM model attribute is selected as a set of information attributes from a hypervisor's data model. Projecting this data to form a meaningful value for the CIM attribute may involve correlation as well as transformation of the information provided by the hypervisor. Projection serves the purpose of eliminating semantic differences, e.g. by defining Units, an compensates for syntactic divergence by aligning the data to a common data model.

E. Mapping of the Example Attributes

1) Virtual machine power state: Using the semantics as described in II-B the following mapping is used to map Xen's power states to the CIM.

- Crashed: Can be mapped to the CIM by a vendor defined state.
- Halted: Maps to the CIM's defined state
- Paused: Maps to the CIM's paused state
- Running: Maps to the CIM's active state
- Suspended: Maps to the CIM's suspended state
- Unknown: Maps to the CIM's unknown state

Also VMware's power states are mappable to the CIM:

- poweredOn: Maps to the CIM's Running state
- poweredOff: Maps to the CIM's Halted state
- suspended: Maps to the CIM's Suspended state

Using our classification scheme this attribute is *1:1 bijective mappable*.

2) Virtual machine and host memory: As stated before neither Xen nor the VMware ESXi expose information about memory extends over their SDKs, so cannot map them properly to the CIM. Assuming having only one memory extend it's possible to map memory information about the total memory of Xen or VMware ESXi hypervisors to the CIM. This assumption is essential also, if we only care about total memory, because else it could span Memory gaps that are not covered by a composing memory extend. Without this assumption EndingAddress, BlockSize, NumberOfBlocks would be unmappable attributes.

The mapping to the StartingAddress can be easily done in both cases, host and virtual machine memory, from both hypervisors by a constant 0-function.

$$\pi_{\text{StartingAddress},H} = 0;$$

The mapping to the EndingAddress can be done by the following projections:

$$\pi_{\text{Host.EndingAddr,Xen}} = \frac{\text{memory}_{total}}{2^{10}}$$

$$\pi_{\text{VM.EndingAddr,Xen}} = \frac{\text{memory}_{actual}}{2^{10}}$$

$$\pi_{\text{Host.EndingAddr,ESXi}} = \frac{\text{memorySize}}{2^{10}}$$

$$\pi_{\text{VM.EndingAddr,ESXi}} = \text{memorySizeMB} * 2^{10}$$

Because there is no information about block sizes available, the only possibility not to let the rest of the attributes *unmappable* will be another assumption: We assume a BlockSize of one should fit most needs. Then the NumberOfBlocks will be equivalent to the EndingAddress.

For the inversion of the mappings for the EndingAddress and NumberOfBlocks, the BlockSize and the possibility of gaps in the Memory area should be taken in account. So the best way to calculate it should be via BlockSize and NumberOfBlocks. So this attributes are *n:1 invertable mappable*, and particularly not *1:1 bijective mappable*. E.g.

$$\pi^{-1}_{\text{Host.EndingAddr,Xen}} = \text{BlockSize} * \#\text{Blocks} * 2^{10}$$

To map to ConsumableBlocks the two cases of host and virtual machine memory have to be looked at separately. In the virtual machine case we have no information and according to the CIM Profile then 0 has to be used, therefore it would be *n:1 mappable*. In the host case we are *1:n invertable mappable* still using the assumptions:

$$\pi_{\text{ConsumableBlocks,Xen}} = \frac{\text{memory}_{total} - \text{memory}_{free}}{2^{10}}$$

$$\pi_{\text{ConsumableBlocks,ESXi}} = \frac{\text{memorySize}}{2^{10}} - \text{overallMemoryUsage} * 2^{10}$$

V. Technical realisation and evaluation

For practical implementation of our harmonising method we chose to extent the *libvirt* to provide unified presentation of management information from the Xen and VMware ESXi hypervisors. We first did an analysis of the data models and the information provided by these hypervisors. By assessing this information using the mapping classes introduced in Section 3 we developed a generalised data model, suited to model management information of both hypervisors and their VMs.

Figure 4 gives an architectural overview of libvirt and also shows where we added our extensions (bold borders). There are three software layers: 1) the *access layer* is the user faced part of libvirt, offering a unified API to multiple virtualization implementations. 2) the *abstraction layer* includes libvirt's module management, where functions and data sources from different drivers are combined to answer requests from the access layer. 3) the *instance layer* with the actual functions.

With this three-layered architecture, libvirt's actual functionality is implemented as *drivers* in the instance layer. While most drivers are interfaces to hypervisors and other virtualization software, the same concept is applied for all other functionality, hence the indication of miscellaneous drivers in Figure 4. We use this driver facility to add normalization to libvirt.

Our generalised data model, offering normalised representation of hypervisors and virtual machines, is accessed via an extension to the access layer of libvirt. Hypervisor specific normalisation drivers implement the projections of attributes to their generalised counterparts. The problems pertaining to accessibility of management information is already addressed in the hypervisor drivers provided by libvirt. The abstraction layer allows us to combine these new drivers to retrieve the necessary information if requested by a user through the access layer.

As we constructed generic attributes from the information available through the Xen and VMware management APIs, this prototypical implementation of our concept is an implicit comparison of these virtualizers' data models. We have identified and mapped 32 attributes that are now presented in a uniform manner.

Eight attributes are names and identifiers like hostnames, MAC-addresses and UUIDs. Together with the numbers of physical CPU cores and physical RAM available/allocated to a host or VM, we identified 13 attributes belonging to the class *1:1 bijectively mappable attributes*. It turns out that while the data models are very different the managed objects are very similar. As every object needs at least one identifier, there is an overlap in the information models thus yielding this high number of semantically equivalent attributes. Challenges in normalising these attributes usually arise from access-related differences (cf. Section II).

In our work, where the generic data model is based on the implementations of VMware and Xen, we identified five attributes from the *n:1 mappable attributes* class and eleven members belonging to the *n:1 invertible mappable attributes* class. Whether an attribute's mapping is invertible or not is mainly decided by the granularity of available management information. For example, VMware ESXi only gives information about the number of threads currently active on a host, while Xen produces the number of currently active threads for each core individually. There is not enough information to reconstruct the value for each core from the unified attribute *runningThreads*. All five not invertible mappings are due to the divergence in granularity. This outcome shows that VMware's and Xen's data models vary in their syntax and data structures, but hardly in their granularity. Following this train of thought this outcome is an indication, that mapping to a very detailed model like the CIM is very likely to be not invertible. This could be problematic if the generic model is supposed to enable attribute changing management operations.

During the implementation of our method we discovered that in this particular instance most challenges are of syntactical nature, i.e. where in the data models a piece of information can be found and how it is stored. Semantic transformation is required for most attributes, due to different implementations. While semantic transformations are time consuming, we didn't encounter any major problems. The third identified dimension in Section II where difference have to be compensated are access-related differences. In this case the libvirt project shows that this problem can be handled very well.

VI. Conclusion

The harmonisation of management information provided by established but heterogeneous products is a difficult and tedious task. In a bottom-up approach, we have analysed raw attributes of two common hypervisors and mapped them se-

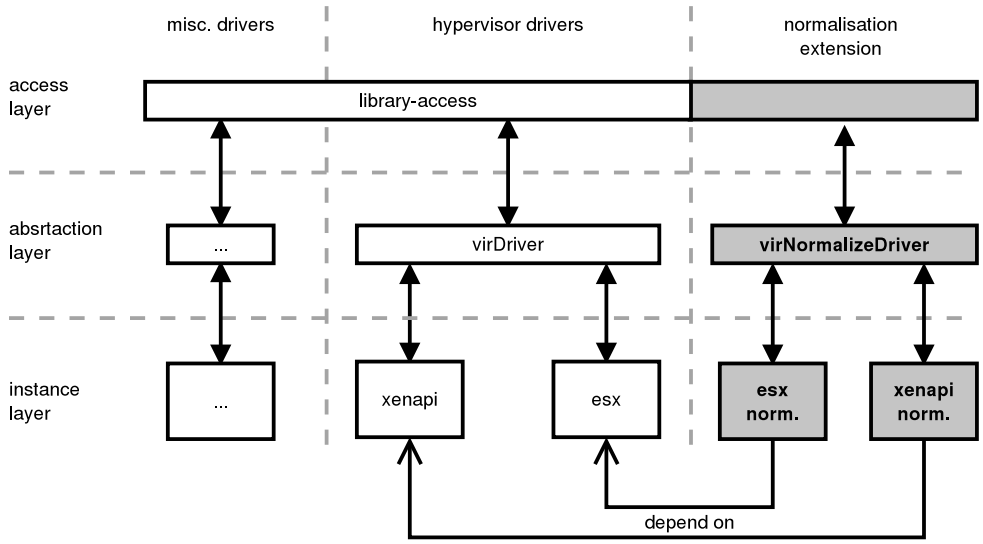


Fig. 4. Architectural overview of libvirt and our extension

mantically and syntactically, before implementing them within an agent based on the *libvirt* library, that provided the access-related mapping. During analysis, it became obvious, that even common hypervisors differ syntactically in most management information items they offer via their management interfaces.

Based on a list detailing the aspects of projection, we have developed a classification scheme and a methodology that enabled the mapping in a systematic manner, and that can be readily applied to additional attributes. While this approach requires a lot of effort and domain knowledge, it may be employed in order to harmonise management information incrementally, in contrast to a top-down approach. Our comparison of attribute semantics and syntax with those specified within the CIM suggest that our approach could even be employed to feed attribute valued into the specified profiles. We selected two prominent hypervisor products, without loss of generality. Using the same method this set can be extended to cover both more attributes and additional hypervisors. In particular, those relying on non-full-virtualization schemes could prove interesting, as their own choice of management attributes can be expected to reflect their different modus operandi.

ACKNOWLEDGMENT

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of this paper. The MNM Team directed by Prof. Dr. Dieter Kranzlmüller and Prof. Dr. Heinz-Gerd Hegering is a group of researchers at Ludwig-Maximilians-Universität München (LMU), Technische Universität München (TUM), the University of the Federal Armed Forces and the Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Science and Humanities. (www.mnm-team.org)

REFERENCES

[1] Letter symbols to be used in electrical technology – Part 2: Telecommunications and electronics. IEC 60027-2 (Ed.3.0), August 2005.
[2] Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehorster, and André Brinkmann. Non-intrusive virtualization management using libvirt. In *DATE*, pages 574–579, 2010.
[3] V. Danciu, N. gentschen Felde, and M. Sailer. Declarative specification of service management attributes. In *Moving From Bits to Business Value: Proceedings of the 2007 Integrated Management Symposium*, volume 2007, München, May 2007. IFIP/IEEE.
[4] V. Danciu, A. Hanemann, H.-G. Hegering, and M. Sailer. IT Service Management: Getting the View. In *Kern, E. M., Hegering, H.-G. und Brügge, B. (Hrsg): Managing Development and Application of Digital Technologies*, volume 2006, pages 110–130. Springer–Verlag, München, Germany, June 2006.
[5] V. Danciu and M. Sailer. A monitoring architecture supporting service management data composition. In *Proceedings of the 12th Annual Workshop of HP OpenView University Association*, number 972–9171–48–3, pages 393–396, Porto, Portugal, July 2005. HP.
[6] DMTF. *CIM System Virtualization Model White Paper*, November 2007.
[7] DMTF. *Allocation Capabilities Profile*, June 2009.
[8] DMTF. *Generic Device Resource Virtualization Profile*, July 2009.
[9] DMTF. *Memory Ressource Virtualization Profile*, July 2009.
[10] DMTF. *Open Virtualization Format White Paper*, June 2009.
[11] DMTF. *Processor Resource Virtualization Profile*, April 2009.
[12] DMTF. *Resource Allocation Profile*, June 2009.
[13] DMTF. *Ethernet Port Resource Virtualization Profile*, October 2010.
[14] DMTF. *Storage Resource Virtualization Profile*, April 2010.
[15] DMTF. *System Memory Profile*, June 2010.
[16] DMTF. *System Virtualization Profile*, April 2010.
[17] DMTF. *Virtual Ethernet Switch Profile*, October 2010.
[18] DMTF. *Virtual System Profile*, April 2010.
[19] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999.
[20] Ewan Mellor, Richard Sharp, David Scott, et al. *Xen Management API - API Revision 1.0.10*, January 2010.
[21] VMware, Inc, Palo Alto. *vSphere Web Services SDK Programming Guide, vSphere Web Services SDK 4.1*, July 2010.