INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Entwicklung von Adaptern zum Management von Hostvirtualisierungslösungen

Marcel Michelmann

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Entwicklung von Adaptern zum Management von Hostvirtualisierungslösungen

Marcel Michelmann

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering (em.) Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Tobias Lindinger
Abgabetermin:	30. Juli 2009

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 30. Juli 2009

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Heutzutage gibt es eine Reihe von Virtualisierungslösungen, mit denen auf einem einzigen physisch vorhandenem Rechner virtuell mehrere Rechner bzw. Betriebssysteme betrieben werden können. Diese Virtualisierungslösungen bringen alle eigene Programme mit, mit denen sie verwaltet werden. Als Grundlage dieser Arbeit dient eine in einer Diplomarbeit konzipierte und prototypisch implementierte Managementplattform für unterschiedliche Virtualisierer. Die Plattform ermöglicht das Management von einer Vielzahl von Virtualisierungslösungen, ohne selbst jeweils für jede dieser Lösungen angepasst werden zu müssen. Zur Kommunikation mit den Hosts, auf denen die Virtualisierunglösungen mit virtuellen Maschinen laufen, dienen Adapter. Zu den Aufgaben eines solchen Adapters zählt u. a. Nachrichten von der Managementplattform anzunehmen, sie entsprechend im Virtualisierer umzusetzen und vom Virtualisierer kommende Daten in einem für die Managementplattform verständlichen Format darzustellen. Dabei besteht die Hauptaufgabe in den semantischen und syntaktischen Anpassungen zwischen der Managementplattform und dem Virtualisierer. Schon zu Beginn bot sich die Möglichkeit an, statt eines Adapters für nur einen Virtualisierer, wie beispielsweise Xen, einen Adapter für libvirt zu entwickeln. libvirt selbst ist kein Virtualisierer, sondern bietet seinerseits den Zugriff auf eine Vielzahl von Virtualisierern an. Durch die Entwicklung dieses Adapters und seine Integration in die bestehende Managementplattform können somit Xen, Qemu und weitere Virtualisierer verwaltet werden.

Inhaltsverzeichnis

1. Einleitung

Die Leistung heutiger Computer nimmt bekanntermaßen mehr und mehr zu. Applikationen, die früher nur in Rechenzentren oder auf speziellen Hochleistungssystemen ausgeführt werden konnten, können heutzutage auf einem handelsüblichen Desktop-PC genutzt werden. Im Gegensatz zu den immer schneller werdenden Prozessoren und den immer größeren Speicherkapazitäten, verändern sich die Anforderungen vieler Benutzer im Laufe der Zeit nicht ganz so rapide, so dass nur selten die volle Rechenkapazität ausgeschöpft wird. Dieser Überschuss kann für die sogenannte Virtualisierung genutzt werden. Virtualisierung bedeutet, dass auf einem physisch vorhandenem System (auf dem „Host“) virtuell mehrere Systeme („virtuelle Maschinen“ oder „virtuelle Systeme“) vorhanden sein können. Über diese virtuellen Systeme können auf einem Rechner mehrere Betriebssysteme eingesetzt werden, wobei für den Nutzer jedes virtuelle System wie ein eigenständiges System aussieht. Der Vorteil, den man durch diese Virtualisierung mehrerer Systeme auf nur einem real vorhandenem System erhält, ist, dass mit einfach vorhandener Hardware eine Vielzahl von Rechnern simuliert werden kann. Auf diese kann beispielsweise gleichzeitig über das Netzwerk zugegriffen werden, oder ein Nutzer kann simultan in mehreren Betriebssystemen arbeiten.

Beispiele für Virtualisierer, d. h. für Produkte, mit denen auf einem Rechner virtuelle Maschinen zum Laufen gebracht werden, sind Xen und VMware ESX.

1.1. Motivation & Ziel

So wie gewöhnliche Desktop-PCs müssen auch virtuelle Maschinen und die Hosts, auf denen sie laufen, verwaltet werden. Dazu gehören Steuerungsaufgaben, wie das Ein- und Ausschalten und das Neustarten virtueller Systeme oder die Zuteilung von Ressourcen, sowie Überwachungsaufgaben (das sogenannte „Monitoring“). Überwacht werden können beispielsweise die Prozessor- und Speicherauslastung. Dabei bringen die verschiedenen Virtualisierer jeweils eigene Werkzeuge mit, mit denen sie verwaltet werden können. Oftmals ist es in großen Projekten nicht unumgänglich, eine Vielzahl verschiedener Virtualisierer zu nutzen, da sie jeweils unterschiedliche Vor- und Nachteile mit sich bringen. Daher steigt mit der Anzahl der Virtualisierer auch die Anzahl der Programme, die genutzt werden, um sie zu verwalten. Um dem entgegenzuwirken, wurde in [Bit08] ein Konzept entwickelt, das vom Management einzelner Virtualisierer abstrahiert und eine gemeinsame Oberfläche bereitstellt, über die sie überwacht und gesteuert werden können.

Die prototypische Implementierung der in [Bit08] konzipierten Managementplattform selbst bietet aber noch keine Möglichkeit der Verwaltung von Virtualisierern. Um die Verbindung zwischen der Managementplattform und den Virtualisierern zu realisieren, wird für jeden Virtualisierer ein Adapter benötigt. Diese Adapter sind für den Nutzer nicht sichtbar, so dass es für ihn aussieht, als ob er über die Oberfläche der Managementplattform direkt mit den Virtualisierern kommuniziert.

Eigentliches Ziel dieser Arbeit war es, einen solchen Adapter für den Virtualisierer Xen zu entwickeln. Zu den Möglichkeiten der Kommunikation mit Xen zählt unter anderem libvirt.

Die Anprogrammierung libvirts bietet jedoch den großen Vorteil, nicht nur Xen, sondern eine Reihe weiterer Virtualisierer in die Managementplattform einbinden zu können. Aus diesem Grund wurde das Ziel der Arbeit, einen Adapter für Xen zu entwickeln, dadurch realisiert, dass ein Adapter für libvirt entwickelt wurde.

1.2. Aufbau der Arbeit

In der vorliegenden Arbeit werden zunächst die Grundlagen der Managementplattform aus [Bit08] und des Virtualisierers Xen erläutert. Anschließend wird das Konzept des Adapters erklärt, ehe auf seine Implementierung eingegangen wird. In der Zusammenfassung wird erörtert, welche Ziele erreicht wurden und es wird ein Ausblick gegeben.

2. Grundlagen

In diesem Kapitel werden zunächst die Grundlagen der Managementplattform aus [Bit08] erläutert, sowie ein kurzer Überblick über das Projektnetz gegeben. Anschließend wird der Virtualisierer Xen beschrieben.

2.1. Managementplattform

Zunächst soll die Idee der Managementplattform dargestellt werden, ehe dann genauer auf ihre Architektur und den Nachrichtenfluss eingegangen wird.

2.1.1. Überblick über die Managementplattform

War es bisher so, das jeder Virtualisierer eigene Hilfsmittel mit sich brachte, mit denen er verwaltet werden konnte, war die grundlegende Idee, die mit der Managementplattform verfolgt wurde, dass viele verschiedene Virtualisierer über eine gemeinsame Oberfläche verwaltet werden sollen. Das bisherige Konzept sieht also vor, das für jeden Virtualisierer das Management über eine eigene Plattform erfolgt (vgl. Abb. 2.1, angelehnt an [Bit08], S. 5).

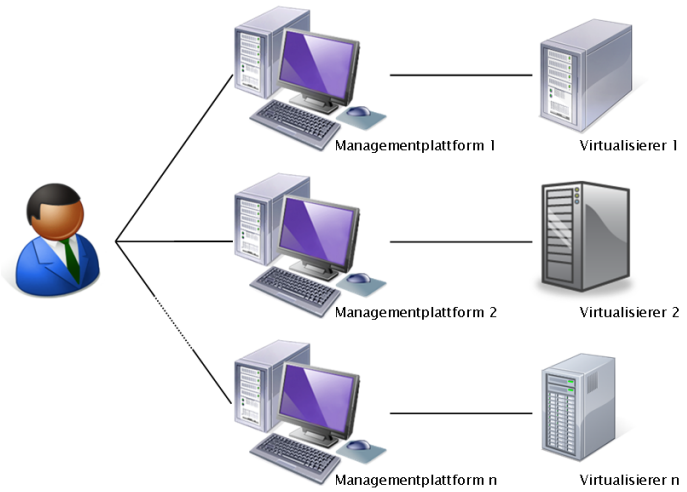

Abbildung 2.1.: Bisheriges Management beim Einsatz verschiedener Virtualisierer (vgl. [Bit08], S. 5)

Dieser Methode ist jedoch eine vorzuziehen, die für die verschiedenen eingesetzten Virtualisierer eine gemeinsame Managementplattform bereitstellt. Diese Idee wird in Abb. 2.2 (angelehnt an [Bit08], S. 6) dargestellt. Hier muss der Nutzer nur noch eine einzige Managementplattform nutzen, um seine Virtualisierer zu verwalten. Die Kommunikation zwischen der Managementplattform und den einzelnen Virtualisierern bleibt dem Nutzer verborgen.

Intern ist für jeden Virtualisierer ein Adapter notwendig. Dieser Adapter nimmt die Befehle der Managementplattform entgegen, gibt sie an den Virtualisierer weiter und gibt dann wiederum das Ergebnis an die Plattform zurück. Beispielsweise fordert die Plattform ein Update über die Speicherauslastung an. Der Adapter nimmt diesen Befehl entgegen, sorgt dafür, dass ein entsprechendes Update vom Virtualisierer kommt, und gibt dieses Update an die Plattform zurück. Dabei muss der Adapter die „Übersetzungsarbeit“ leisten, da weder der Virtualisierer die Befehle der Plattform, noch die Plattform die vom Virtualisierer gelieferten Daten verstehen kann. Da sich die Virtualisierer untereinander auch deutlich voneinander unterscheiden, sei nochmals angemerkt, dass für jeden Virtualisierer jeweils ein eigener Adapter benötigt wird. In Abb. 2.3 werden die Aufgaben des Adapters noch einmal grafisch dargestellt.

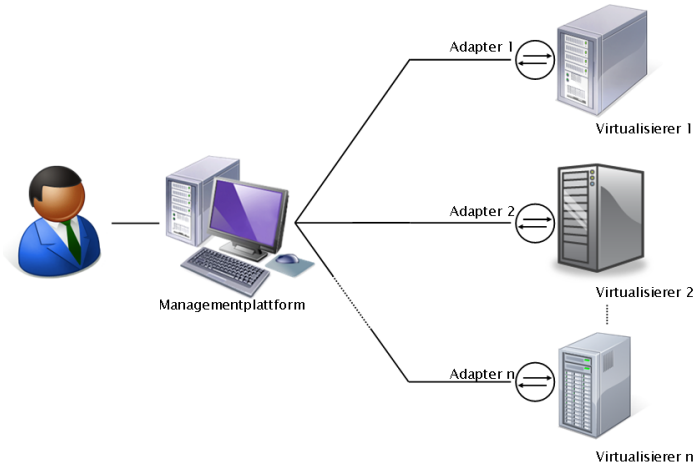

Abbildung 2.2.: Grundlegende Idee der Managementplattform (vgl. [Bit08], S. 6)

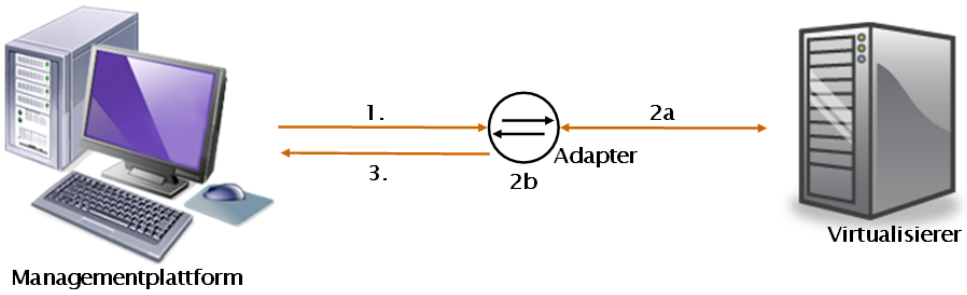

Abbildung 2.3.: Aufgaben des Adapters und Kommunikation der Komponenten

In der Implementierung der Managementplattform, die im Rahmen der Diplomarbeit [Bit08] geleistet wurde, wurden Anforderungen wie das Hinzufügen und Entfernen von Res-

sourcen, sowie deren Monitoring umgesetzt. Das Monitoring der Ressourcen war für die Prozessorauslastung beispielhaft vorimplementiert und wurde im Rahmen dieser Arbeit um das Monitoring der Arbeitsspeicherauslastung erweitert. Komplexere Funktionen, wie zum Beispiel die Überwachung der Dienstgüte der Ressourcen, das Klonen virtueller Maschinen oder das Migrieren virtueller Maschinen von einem Host zu einem anderen wurden nicht realisiert. Auch eine Auto-Discovery-Funktion wurde nicht umgesetzt. Das bedeutet, die Plattform erkennt nicht von sich aus laufende Hosts und virtuelle Maschinen, sondern sie müssen explizit angegeben werden. Einen Überblick über den Funktionsumfang der Plattform bietet Tab. 2.1 (verkürzt übernommen aus [Bit08], S. 78).

Funktionale Anforderung	*Implementierung*
Hinzufügen/Entfernen von Ressourcen	✓
Konfigurieren der Ressourcen	—
Monitoring der Ressourcen	✓
Dienstgüteüberwachung der Ressourcen	—
Zustandsüberwachung der Ressourcen	—
VM-Migration	—
VM-Steuerung	—
VM-Vorlagen	—
Klonen von VMs	—
Benutzer-/Gruppenverwaltung	—
Benachrichtigungen	—
Autodiscovery	—

Tabelle 2.1.: Funktionen der Plattform (vgl. [Bit08], S. 78)

2.1.2. Architektur

Die Managementplattform gliedert sich in folgende Bausteine, die in Abb. 2.4 ([HAN99], S. 278) zu sehen sind:

- Oberflächenbaustein
- Managementapplikationen
- Kernsystem (in [Bit08] Informationdispatcher)
- Informationsverwaltung
- Kommunikationsbaustein

Im Oberflächenbaustein werden die managementrelevanten Informationen dargestellt und der Nutzer erhält Zugriff auf die Managementapplikationen. Dabei gibt es noch keine Festlegung auf eine bestimmte Technologie, sondern es können Monitoringsysteme oder grafische Benutzerschnittstellen angeschlossen werden.

Die Managementapplikationen stellen die eigentliche Funktionalität der Plattform dar und verarbeiten die gesammelten Informationen.

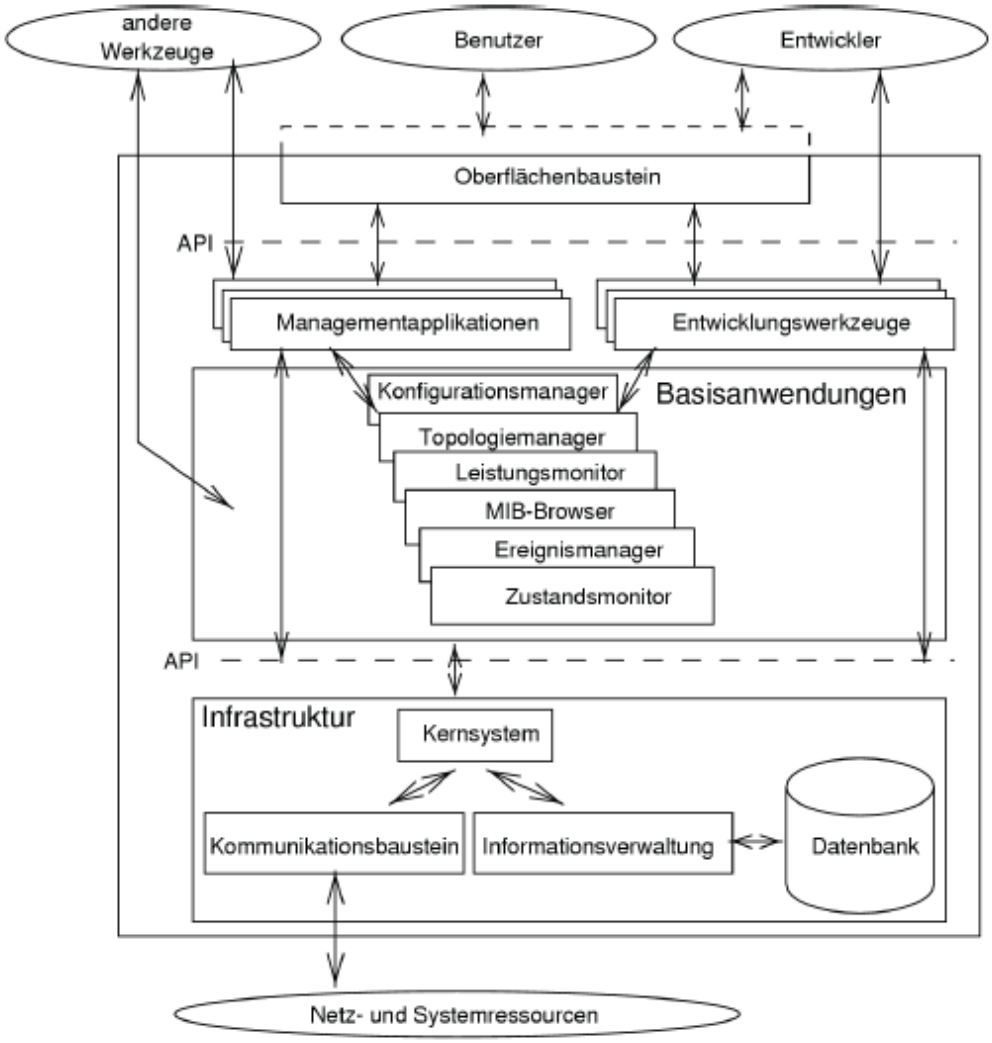

Abbildung 2.4.: Architektur der Managementplattform ([HAN99], S. 278)

Der Informationdispatcher (das Kernsystem) ist die Schaltzentrale zwischen Managementapplikationen, Kommunikationsbaustein und Informationsverwaltung. Er empfängt Informationen und leitet diese an die entsprechenden Stellen weiter.

Durch die Informationsverwaltung werden die Daten gespeichert und verwaltet. Desweiteren ist sie für die Anbindung an Datenbanken und Verzeichnisdienste zuständig.

Der Kommunikationsbaustein ist für die hier vorliegende Arbeit der relevanteste Baustein, da er die Schnittstelle zu den Ressourcen und zur Plattform darstellt. Dies wird in Abb. 2.5 (angelehnt an [Bit08], S. 30) grafisch verdeutlicht.

2.1.3. Kommunikation

Innerhalb der Managementplattform erfolgt die Kommunikation über Nachrichten („Messages“). Für den Zweck dieser Arbeit sind die Get-Nachrichten, und dabei insbesondere die

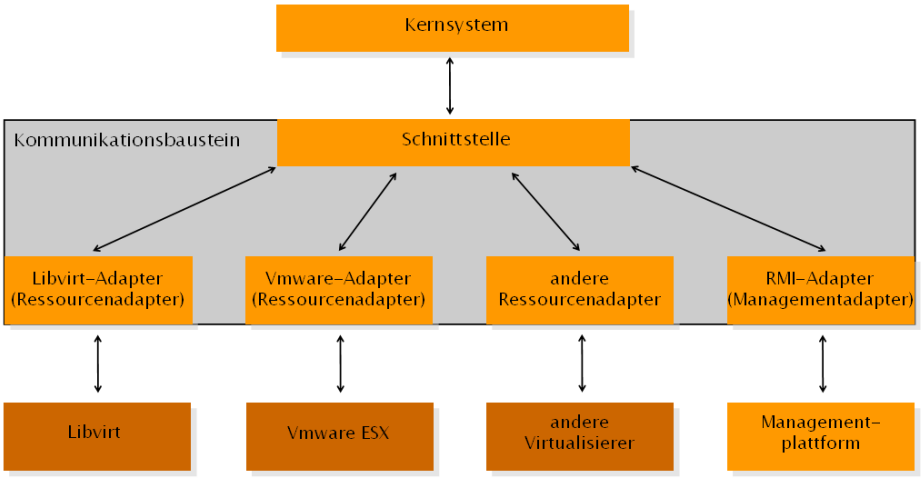

Abbildung 2.5.: Funktion des Kommunikationsbausteins (vgl. [Bit08], S. 30)

Nachrichten vom Typ GetMonitoringUpdate am bedeutendsten.

Wenn der Adapter über den Kommunikationsbaustein eine Nachricht erhält, wird diese intern im Adapter verarbeitet. Wenn die Werte, die durch das GetMonitoringUpdate angefordert wurden, ermittelt bzw. errechnet worden sind, erstellt der Adapter eine Update-Nachricht mit der gewünschten Information und sendet sie an den Informationdispatcher, der sich um deren Weiterleitung kümmert.

Diese Abfolge ist in Abb. 2.6 (angelehnt an [Bit08], S. 56) als Diagramm dargestellt.

2.1.4. Aufbau des Projektnetzes

Das verwendete Projektnetz ist in Abb. 2.7 dargestellt. Hier sind insbesondere zwei Rechner von Bedeutung. Auf einem Rechner im Netz läuft die Managementplattform mit dem hier entwickelten Adapter. Diese Maschine ist auch das Gateway für die dahinterbefindlichen Hosts und fungiert als Router, über den die anderen Rechner erreichbar sind. Die Rechner, die in der Abbildung in der oberen Reihe dargestellt sind, sind die Systeme, auf denen die Virtualisierer laufen. Für diese Arbeit ist der oben links abgebildete Rechner wichtig, da auf diesem der Virtualisierer Xen installiert ist.

Die Kommunikation zwischen diesen beiden Rechnern erfolgt über ssh.

2.2. Virtualisierer

In diesem Unterkapitel wird zunächst eine kurze Einführung in Xen gegeben, und anschließend auf Methoden für das Monitoring und Steering von Xen eingegangen. Danach wird der Ansatz von libvirt erläutert, für das letztendlich der Adapter entwickelt wurde.

2.2.1. Einführung in Xen

Als Virtualisierer wurde in dieser Arbeit Xen verwendet. Xen ist ein Hypervisor, auch Virtual Machine Monitor genannt, vom Typ 1. Diese Art von Hypervisor zeichnet sich dadurch

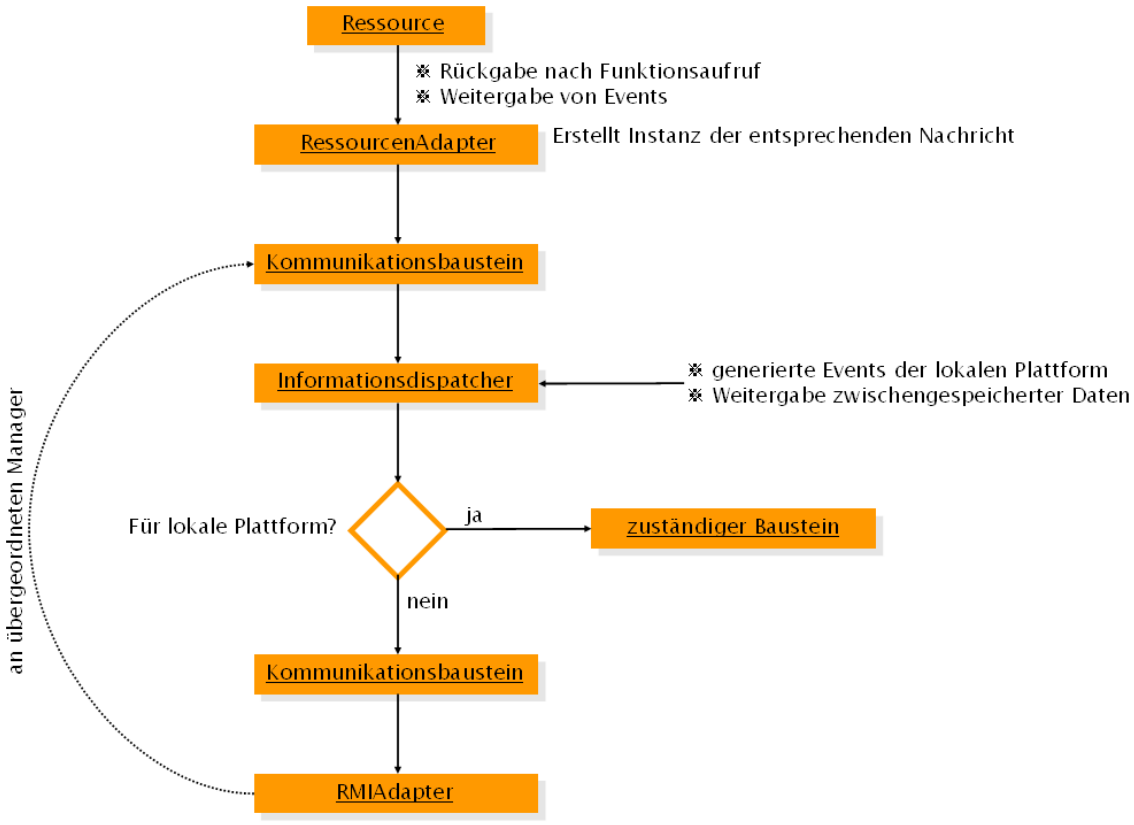

Abbildung 2.6.: Weg der vom Adapter erstellten Nachrichten (vgl. [Bit08], S. 56)

aus, dass sie direkt auf der physischen Hardware läuft, und die Betriebssysteme - von der physischen oder von den virtuellen Maschinen - dann über den Hypervisor auf die Hardware zugreifen. Ferner ist der Hypervisor zuständig für das Prozessor-Scheduling, Speicherpartitionierungen und den Betrieb der virtuellen Maschinen (vgl. [xen08a]). Der Hypervisor steht also im Gegensatz zu normalen Anwendungen, die nach dem Hochfahren des Betriebssystems gestartet und verwendet werden. Diese Vorgehensweise wird in Abb. 2.8 ([ibm05], S. 3) verdeutlicht.

Das Betriebssystem des Hosts (also des physisch vorhandenen Systems) wird vom Hypervisor auch als virtuelle Maschine betrieben, allerdings mit besonderen Zugriffsrechten auf die Ein- und Ausgabehardware. Genannt wird dieses als virtuelles Betriebssystem verwaltete Hostbetriebssystem „Domain 0“.

Im Gegensatz dazu stehen die tatsächlichen Gastsysteme. Diese virtuellen Maschinen werden Domain U genannt.

2.2.2. Monitoring und Steering von Xen

Zum Verwalten von Xen gibt es einerseits eine Programmierschnittstelle, andererseits auch eine ganze Reihe fertiger Programme. Die Programmierschnittstelle, mit der Xen gesteuert werden kann, die sogenannte „Xen Management API“, wird über XML Remote Procedure Calls (XML RPC) angesprochen. Diese Aufrufe können direkt über HTTP oder HTTPS

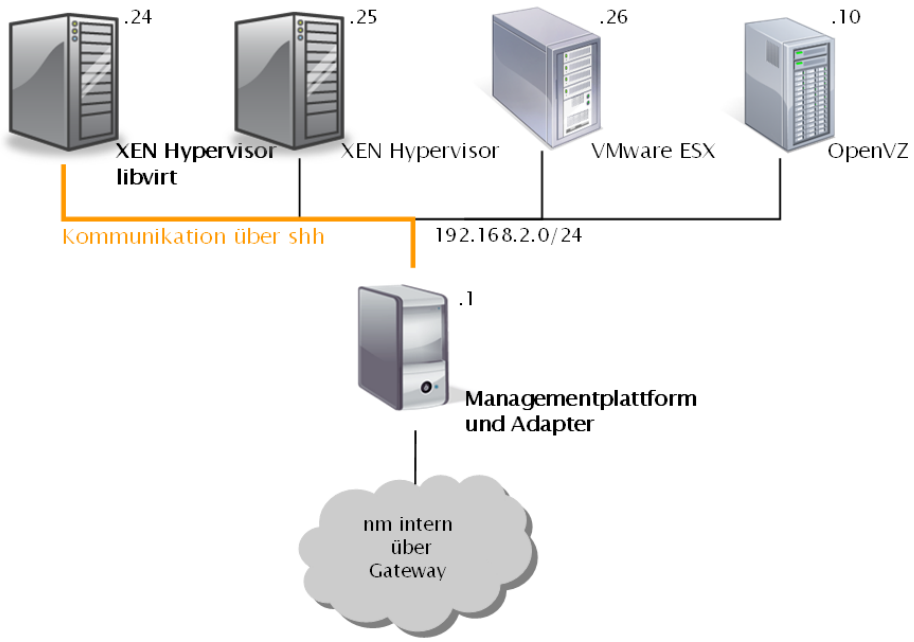

Abbildung 2.7.: Aufbau des Projektnetzes

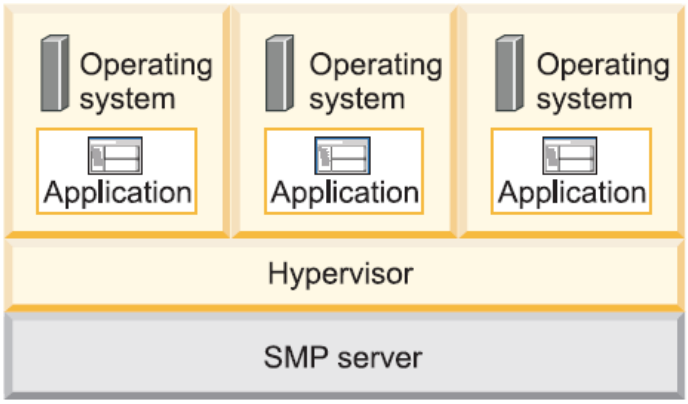

Abbildung 2.8.: Hypervisor Typ 1 als Schnittstelle zwischen Hardware und Betriebssystemen ([ibm05], S. 3)

oder mit Hilfe einer Sprachanbindung für die Programmiersprachen Python, C und Java verwendet werden (vgl. [xen08b]).

Ein einfaches Programm, das standardmäßig bei der Installation von Xen mitgeliefert wird, ist xm. Mit diesem Kommandozeilenprogramm können alle wichtigen Funktionen bezüglich der Verwaltung virtueller Maschinen erledigt werden. Dazu gehören u. a. das Erstellen, Pausieren, Neustarten oder Herunterfahren virtueller Maschinen und eine Vielzahl weiterer Funktionen. xm kommuniziert über XML RPCs mit dem Xen Daemon Xend, der wiederum die Befehle über Domain 0 im Xen Hypervisor umsetzt. Grafisch wird das in Abb. 2.9 ([xen08a], S. 6) verdeutlicht.

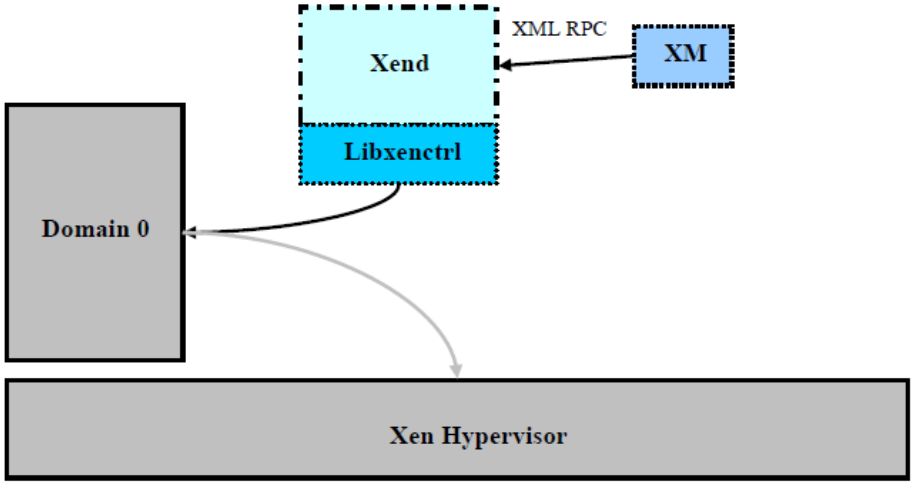

Abbildung 2.9.: Funktionsweise von xm ([xen08a], S. 6)

Weitere Programme, ohne einen Anspruch auf Vollständigkeit, sind

- xen-tools: eine Menge von Skripten zur Installation von virtuellen Maschinen unter Xen, zu finden unter `http://xen-tools.org/software/xen-tools/`
- Xen Shell: ein Kommandozeilenprogramm für einfache Managementaufgaben, zu finden unter `http://www.xen-tools.org/software/xen-shell/`
- ConVirt, ehemals XenMan: ein sehr umfangreiches Programm mit grafischer Oberfläche, zu finden unter `http://www.convirture.com/`
- DTC-xen: neben grundlegenden Funktionen, wie dem Starten und Stoppen virtueller Maschinen, können mit diesem Programm Betriebssysteme auf den virtuellen Maschinen installiert werden, in die man sich anschließend über eine ssh-Verbindung einloggen kann. Zu finden u. a. unter `http://freshmeat.net/projects/dtc-xen/`

Ein Überblick über die Möglichkeiten, Xen zu verwalten, wird in Abb. 2.10 ([Mel07]) gegeben. In dieser Abbildung sieht man, dass alle Zugriffe auf Xen über den Xen Daemon (xend) laufen. Dieser wird mittels XML RPCs über HTTP oder HTTPS angesprochen. Die Remote Procedure Calls werden durch fertige Programme oder über die Sprachanbindungen erzeugt.

Auf der obersten Schicht in Abb. 2.10 wird auch bereits libvirt dargestellt, auf das im folgenden Abschnitt eingegangen wird.

2.2.3. Überblick über libvirt

Ein ähnlicher Ansatz wie der, der mit der Managementplattform verfolgt wird, wird bereits durch das Projekt libvirt verfolgt. Über libvirt können über einen gemeinsamen Befehlssatz Xen, QEMU, KVM, LXC, OpenVZ, User Mode Linux und VirtualBox verwaltet werden (vgl. [lib09c]). Zusätzlich zu den in der Programmiersprache C verfügbaren Paketen von libvirt sind auch Sprachanbindungen für die Programmiersprachen C++, Python, Perl, OCaml,

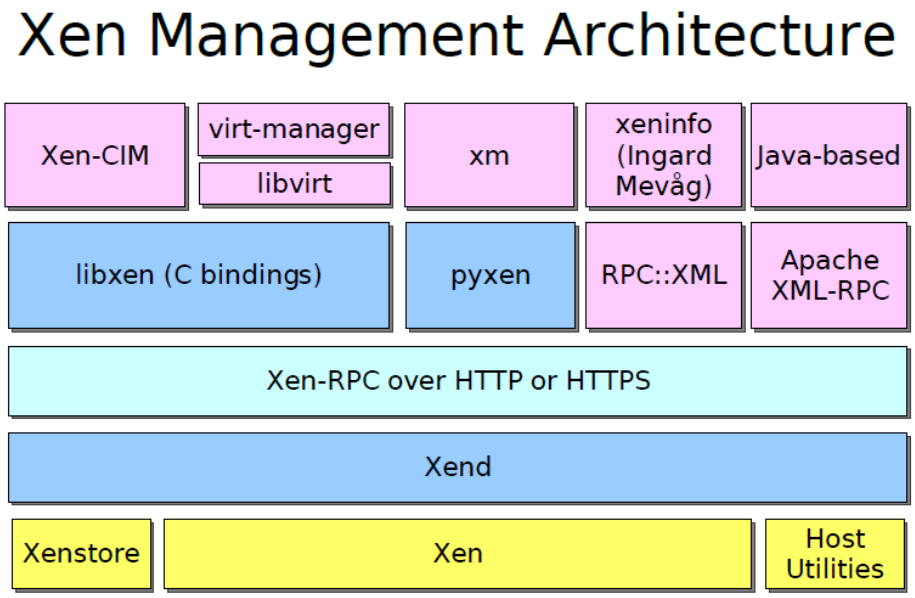

Abbildung 2.10.: Möglichkeiten für das Monitoring und Steering von Xen ([Mel07])

Ruby, Java und C# verfügbar (vgl. [lib09a]). Dabei ist libvirt „(...) eine Programmbibliothek (Library) für die Entwicklung von Managementlösungen für virtualisierte Systeme. Die libvirt bildet eine zusätzliche Abstraktionsebene und bietet dem Anwendungsentwickler eine definierte Schnittstelle zu den Virtualisierungsfunktionen des Betriebssystems“ [Pic09].

Wenn also statt der Entwicklung eines Adapters für Xen die eines Adapters für libvirt angestrebt wird, kann damit nicht nur das Management von Xen realisiert werden, sondern man erhält gleichzeitig die Möglichkeit, eine ganze Reihe anderer Virtualisierer zu verwalten.

Dieses Modell wird in Abb. 2.11 noch einmal dargestellt. Über die Managementplattform auf der obersten Schicht kann über den libvirt-Adapter die libvirt-API angesteuert werden, die wiederum beispielsweise einen Xen-Host ansteuert. Dabei kann das Element auf der untersten Schicht auch mit einem anderen, von libvirt unterstützten System, getauscht werden.

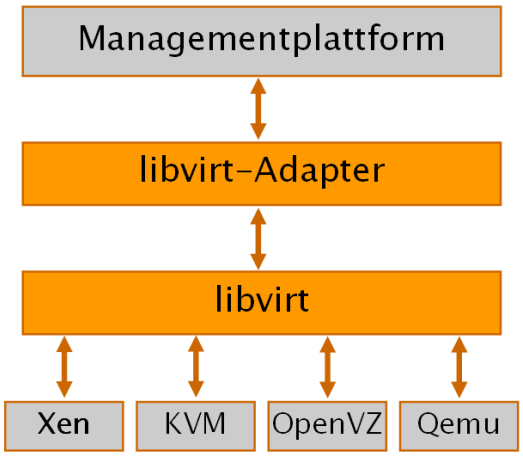

Abbildung 2.11.: Schichten der Architektur

In dieser Arbeit wurde die libvirt Sprachanbindung für Java verwendet. Die Funktionsaufrufe geschehen hierbei in gewohnter Art, über eine API, deren Beschreibung unter [lib09d] zu finden ist. Die wichtigsten Klassen sind hierbei Connect zur Verbindung mit dem Host, Domain für Verbindungen mit virtuellen Maschinen sowie NodeInfo und DomainInfo für weitergehende Informationen über den Host bzw. die virtuelle Maschine. Diese sind im Klassendiagramm der Klassen aus dem Paket org.libvirt in Abb. 2.12 in einem helleren Farbton unterlegt.

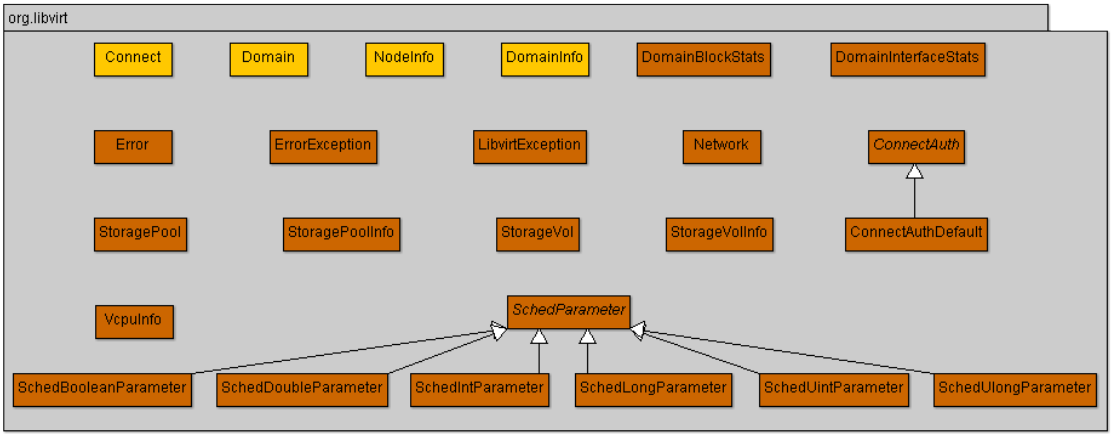

Abbildung 2.12.: Übersicht über die Klassen des Pakets org.libvirt

Die Kommunikation zwischen dem Adapter und dem Virtualisierer wird über ssh realisiert. Die Remote URI, die im allgemeinen Fall

driver[+transport]://[username@][hostname][:port]/[path][?extraparameters]

([lib09b]) lautet, ist hier, bei einem Xen-Host mit der IP-Adresse 192.168.2.24, der über ssh angesprichen werden soll,

xen+ssh://192.168.2.24/.

Getestet werden kann dies im Übrigen auch ohne die Programmierung eines eigenen Programmes mit Hilfe von virsh. virsh ist ein kleines Programm zum Management virtueller Maschinen, das mit libvirt mitinstalliert wird. Der Aufruf dazu lautet in diesem Fall

```
virsh -c xen+ssh://192.168.2.24.
```

Wie mit den über die Java-Anbindung gewonnenen Informationen weiter verfahren wird, wird in Kapitel 3.4 näher erläutert.

3. Konzept des Adapters

In diesem Kapitel soll zuerst kurz auf die Funktionalität des erstellten Adapters eingegangen werden. Anschließend werden dessen Architektur beschrieben und die semantischen und syntaktischen Konvertierungen erläutert, die nötig waren, um die Kommunikation zwischen Managementplattform und Virtualisierer zu ermöglichen.

3.1. Funktionalität des Adapters

Die Funktionen des Adapters lassen sich wie folgt in drei Kategorien unterteilen.

- Allgemeine Aufgaben
  - Verbindungen zum Host und zu darauf laufenden virtuellen Maschinen aufbauen
  - Name des Hosts ermitteln
  - Typ des Virtualisierers ermittlen
  - Name der virtuellen Maschinen ermitteln
- Dynamische Werte des Hosts
  - Prozessorauslastung
  - Arbeitsspeicherauslastung
- Dynamische Werte der virtuellen Maschine
  - Prozessorauslastung
  - Arbeitsspeicherauslastung

Kapitel 4.2 widmet sich der Implementierung dieser Funktionen.

3.2. Notwendigkeit des Adapters

Aus Tab. 3.1 wird ersichtlich, warum ein Adapter nötig ist. Dort ist spezifiziert, was die Anforderungen der Managementplattform sind, was libvirt dafür bereitstellt, und was der Adapter leisten muss, um eventuelle Differenzen auszugleichen. Da es sich um eine erweiterbare Managementplattform für verschiedene Virtualisierer handelt, ist die linke Spalte der Tabelle, also die Situation, die durch die Managementplattform gegeben ist, für jeden Virtualisierer gleich. Jedoch ist die mittlere Spalte für jeden Virtualisierer unterschiedlich. Die Anpassungen zwischen beiden Seiten, die der Adapter übernimmt, sind in der rechten Spalte dargestellt. Hinzu kommen vor der Ausführung der angegebenen Aufgabe noch jeweils die Analyse der vom Kommunikationsbaustein kommenden Nachricht und nach Beendigung der Aufgabe das Verschicken einer entsprechenden Nachricht an den Kommunikationsbaustein. Auf die genaue Ausführung der Aufgaben wird in Kapitel 3.4 und 4.2 eingegangen.

Anforderung der Managementplattform	*Durch libvirt angeboten*	*Aufgabe des Adapters*
Name der virtuellen Maschine	Name der virtuellen Maschine als String	libvirt-API-Aufruf
Name des Hosts	Name des Hosts als String	
Produktname des Virtualisierers	Produktname als String	
Größe des Arbeitsspeichers der virtuellen Maschine in Megabyte	Größe des Arbeitsspeichers der virtuellen Maschine in Kilobyte	libvirt-API-Aufruf und Umrechnung
Größe des Arbeitsspeichers des Hosts in Megabyte	Größe des Arbeitsspeichers des Hosts in Kilobyte	
CPU-Geschwindigkeit des Hosts in Megahertz in der Form „... MHz“	CPU-Geschwindigkeit des Hosts in Megahertz	libvirt-API-Aufruf und Konkatenation mit „ MHz“
CPU-Geschwindigkeit der virtuellen Maschine in Megahertz in der Form „... MHz“	nicht verfügbar	nicht umgesetzt, da für weiteren Verlauf ohne Bedeutung
RAM-Auslastung der virtuellen Maschine in Prozent (ganzzahlig)	gesamter vorhandener und momentan verwendeter Arbeitsspeicher in Kilobyte	Inbezugsetzung der Werte und Rundung auf ganzzahlige Prozentangabe
RAM-Auslastung des Hosts in Prozent (ganzzahlig)	nicht verfügbar	Iteration über alle auf dem Host befindlichen virtuellen Maschinen und Aufsummieren der jeweiligen RAM-Auslastungen
CPU-Auslastung der virtuellen Maschine in Prozent (ganzzahlig)	nicht verfügbar	Errechnung über zwei Zeitmessungen und die dabei vergangene reale und CPU-Zeit
CPU-Auslastung des Hosts in Prozent (ganzzahlig)	nicht verfügbar	Iteration über alle auf dem Host befindlichen virtuellen Maschinen und Aufsummieren der jeweiligen CPU-Auslastungen

Tabelle 3.1.: Adapter zwischen Managementplattform und Virtualisierer

3.3. Architektur

Die gesamte Funktionalität des Adapters wurde auf drei Klassen aufgeteilt, die in Abb. 3.1 in einem Klassendiagramm dargestellt sind.

Die Klasse LibvirtHostInformation stellt Informationen über den Host bereit. Dazu gehören beispielsweise Informationen über die Anzahl der CPUs, die Größe des Hauptspeichers und die Namen der auf diesem Host laufenden virtuellen Maschinen.

Die Klasse LibvirtVMInformation stellt Informationen über einzelne virtuelle Maschinen bereit. Dazu gehören unter anderem die maximale und die tatsächlich verwendete Größe des Hauptspeichers, das verwendete Betriebssystem und die aktuelle Prozessorauslastung.

Diese Informationen werden von der Klasse Libvirtadapter, einer Unterklasse von ResourceAdapter, verwendet, um sie der Managementplattform aufbereitet zu liefern. Der Libvirtadapter ist somit das Bindeglied zwischen der Managementplattform und dem Virtualisierer. Eine weitere wichtige Aufgabe der Klasse Libvirtadapter ist es, die in der Managementplattform intern verwendeten Objekte zu erstellen. Bei Initialisierung des Adapters werden für die Plattform intern Instanzen der Klassen PhysicalSystem bzw. VirtualSystem erzeugt, die wiederum die Elemente Ram und PhysicalCpu bzw. VirtualCpu haben. Die Eigenschaften dieser Instanzen, wie z. B. die Größe des Rams oder die Geschwindigkeit der CPU in MHz wird mit echten Werten belegt, die über die Klassen LibvirtHostInformation bzw. LibvirtVMInformation bezogen werden.

Diese Dreiteilung in das „Bindeglied“ (Libvirtadapter), Informationen über den Host (LibvirtHostInformation) und Informationen über virtuelle Maschinen (LibvirtVMInformation) dient der Abstraktion und Modularisierung des entwickelten Adapters. Der Vorteil daraus ist, dass die Klassen jeweils von Veränderungen in einer der anderen Klassen unberührt bleiben. Ändert sich z. B. der Zugriff auf einen Host, muss dieser nur in der Klasse LibvirtHostInformation angepasst werden. Desweiteren bietet die Dreiteilung mehr Übersichtlichkeit durch die Aufteilung nach Zuständigkeiten.

3.4. Semantische und syntaktische Konvertierungen

Da sich die in der Managementplattform geforderten Informationen in Semantik und Syntax von den über die libvirt-API gelieferten Werte unterscheiden, muss im Adapter eine semantische und syntaktische Konvertierung durchgeführt werden.

Bei der Anzeige der verwendeten CPU-Kapazitäten (die sich nicht direkt auslesen lässt, sondern selbst errechnet werden muss, siehe Kapitel 4.2.3), wird von der Managementplattform ein Integer-Wert verlangt. Bei den Berechnungen im Adapter wird zunächst eine Gleitkommazahl errechnet, die dann (mit Math.ceil()) aufgerundet und in ein Integer gecastet wird. Der Grund, warum statt des eigentlich korrekten kaufmännischen Rundens die Ceil-Funktion verwendet wird, ist die Überlegung, dass sonst bei oft auftretender geringer Last (<0,5%) stets eine Nulllast angezeigt werden würde.

Auch bei der Arbeitsspeicherauslastung verlangt die Managementplattform eine ganzzahlige Prozentangabe. Der eigentliche genauere Wert, der durch Division der Größe des verwendeten Hauptspeichers durch die Größe des verfügbaren Hauptspeichers errechnet wird, wird durch die Funktion Math.round() und anschließendes Casten in die geforderte Form gebracht.

In Kapitel 3.3 wurde bereits erwähnt, dass die Managementplattform mit Hilfe des Adap-

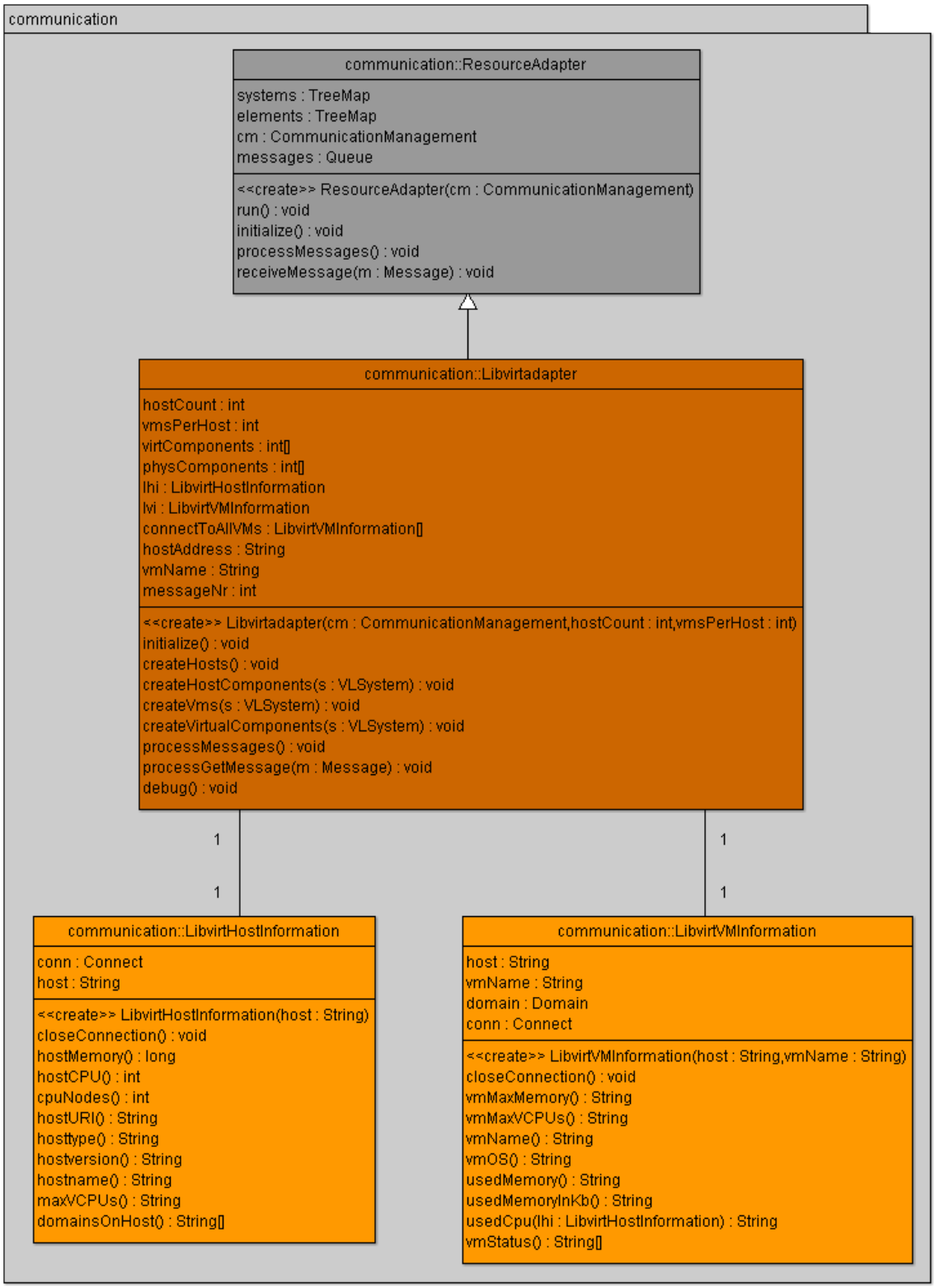

Abbildung 3.1.: Klassendiagramm für die drei erstellten Adapterklassen

ters interne Komponenten erstellt. Diese sind für den Host ein PhysicalSystem mit PhysicalCpu und Ram sowie für die virtuellen Maschinen jeweils ein VirtualSystem mit VirtualCpu und Ram. Die Eigenschaften dieser Komponenten, also Geschwindigkeit des Prozessors und Größe des Arbeitsspeichers werden mit echten Werten belegt, die vorher wie folgt konvertiert werden müssen. Die CPU-Geschwindigkeit der PhysicalCpu kann mit Hilfe des Adapters direkt ausgelesen werden. Diese Zahl muss nur noch in einen String konvertiert und mit „ Mhz“ konkateniert werden. Die Größe des RAMs wird durch den Adapter in Kilobyte geliefert; da für die Erstellung des Objekts der Klasse „Ram“ diese Größe jedoch in Megabyte benötigt wird, muss der gelieferte Wert zunächst umgewandelt und in einen Integer gecastet werden. Die Größe des Arbeitsspeichers der virtuellen Maschine erhält man nicht direkt, sondern nur über Umwege. Man kann sich den Status der virtuellen Maschine ausgeben lassen, der beispielsweise wie folgt aussieht:

```
state:VIR_DOMAIN_BLOCKED
maxMem:131072
memory:79872
nrVirtCpu:1
cpuTime:144551614561
```

Benötigt wird die Zahl, die hinter maxMem steht. Der ausgegebe Status kann mit `split("\\n")` in ein String-Array, dessen Name hier vmStatus sei, gespeichert werden, aus dem dann mit folgender Befehlskette der benötigte Wert in Megabyte errechnet wird.

```
String ramSizeString = vmStatus[2];
String[] ramSizeToken = ramSizeString.split(":");
int ramSize = (int) (Integer.parseInt(ramSizeToken[1]) / 1024);
```

Abschließend kann dann mit

```
Ram ram = new Ram();
ram.setSize(ramSize);
```

die interne Komponente erstellt und mit der korrekten Größe belegt werden.

Für die VirtualCpu hingegen ist die Belegung mit der aktuellen Geschwindigkeit nicht möglich, da diese nicht ausgelesen werden kann. Da diese Eigenschaft aber in der Managementplattform bisher noch keine weitere Verwendung findet, kann die Geschwindigkeit ohne Einschränkung einfach statisch mit einem beliebigen Wert belegt werden.

4. Implementierung

Dieses Kapitel widmet sich zunächst den Vorbereitungen, die auf den Rechnern im Projektnetz durchgeführt werden mussten, um darauf aufbauend die Implementierung des Adapters zu erläutern.

4.1. Grundlegende Vorbereitungen

Die Vorbereitungen müssen auf zwei verschiedenen Rechnern durchgeführt werden. Zum einen auf dem Rechner, auf dem der Virtualisierer läuft, dem Virtualisierungshost, zum anderen auf dem Rechner, auf dem die Managementplattform läuft, dem Managementhost. Auf beiden Rechnern wird als Betriebssystem Linux eingesetzt, genauer gesagt Debian in der aktuell stabilen Version 5.0, genannt Debian Lenny.

4.1.1. Vorbereitungen auf dem Virtualisierungshost

In diesem Abschnitt wird davon ausgegangen, dass der gewünschte Virtualisierer, in diesem Fall Xen, schon auf dem Virtualisierungshost installiert wurde. Zur Installation von Xen auf Linux Debian kann [Bit08], S. S.86-88, befolgt werden.

Zunächst muss libvirt installiert werden. Die zum Fortfahren essentiellen Pakete können mit

```
apt-get install gcc libc6-dev pkg-config
```

in der jeweils aktuellsten Version installiert werden.

Um auf dem Host libvirt verwenden zu können, benötigt man noch libvirt0 und libvirt-bin, die mit

```
apt-get install libvirt0
apt-get install libvirt-bin
```

installiert werden können. Hierbei ist zu beachten, dass bei den beiden Paketen die Version 0.4.6-10 verwendet wird.

Auf `http://libvirt.org/sources/` muss die Datei libvirt-0.4.6.tar.gz heruntergeladen, entpackt und installiert werden:

```
wget http://libvirt.org/sources/libvirt-0.4.6.tar.gz
tar xvf libvirt-0.4.6.tar.gz
cd libvirt-0.4.6
./configure
make
make install
```

Der Zugriff per ssh auf die Maschine darf nicht mehr über die Passworteingabe, sondern muss über eine Public-Key-Authentifizierung erfolgen. Auf dem Rechner, auf dem die Plattform laufen soll, wird durch

```
ssh-keygen -t dsa
```

ein Schlüsselpaar erzeugt. Per

```
scp ~/.ssh/id_dsa.pub root@192.168.2.24:
```

wird der Public-Key des Managementhosts auf den Virtualisierungshost kopiert. Dieser muss noch an die richtige Stelle geschrieben werden:

```
ssh root@192.168.2.24
mkdir .ssh
cat id_dsa.pub >>.ssh/authorized_keys
rm id_dsa.pub
```

Anschließend wird der ssh-Zugriff beschleunigt, in dem man in der Datei /etc/ssh/sshd_config die Eigenschaft UseDNS auf „no“ stellt.

Der Host ist nun fertig vorbereitet. Der libvirt-Daemon muss noch mit

```
libvirtd&
```

von einem beliebigem Pfad aus gestartet werden.

4.1.2. Vorbereitungen auf dem Managementhost

Auch auf dem Rechner, auf dem die Plattform laufen soll, müssen Vorbereitungen getroffen werden. Wie die Plattform und insbesondere die Web-Oberfläche in Betrieb genommen werden, kann in [Bit08], S. 81-83, verfolgt werden.

Um Java in der benötigten, aktuellen Version sun-java6-sdk zu installieren, muss in der Datei /etc/apt/sources.list die Zeile

```
deb http://ftp.de.debian.org/debian/ lenny main non-free contrib
```

hinzugefügt werden. Anschließend kann Java per

```
apt-get install sun-java6-sdk
```

installiert werden.

Die libvirt Java Sprachanbindung erhält man über

```
wget http://libvirt.org/sources/java/libvirt-java-0.2.1.tar.gz
tar xvf libvirt-java-0.2.1.tar.gz
cd libvirt-java-0.2.1
```

In der im Verzeichnis libvirt-java-0.2.1 liegenden Datei configure muss in Zeile 128 JAVA_HOME auf /usr/lib/jvm/java-6-sun gesetzt sein.

Anschließend wird

```
./configure --prefix=/usr
make
make install
```

ausgeführt.

Die Datei libvirt-0.2.1.jar muss sich in /usr/share/java sowie in dem Ordner befinden, in dem der Quellcode der Plattform ist.

4.2. Programmierung des Adapters

Die tatsächliche Implementierung der Funktionalitäten geschieht, wie in Kapitel 3.3 erläutert, in den drei Klassen Libvirtadapter, LibvirtVMInformation und LibvirtHostInformation. Die folgenden Unterkapitel gliedern sich nach der Kategorisierung in Kapitel 3.1.

4.2.1. Allgemeine Aufgaben

Der Verbindungsaufbau zum Host wird in der Klasse Libvirtadaper gekapselt über einen Konstruktoraufruf hergestellt:

```
LibvirtHostInformation lhi = new LibvirtHostInformation(hostAddress);
```

Dort wird dann dann die Verbindung aufgebaut:

```
Connect conn = new Connect(hostAddress);
```

Die Klasse Connect, der im Konstruktor die Hostadresse übergeben wird, ist Teil der libvirt-Bibliothek.

Der Verbindungsaufbau zur virtuellen Maschine wird ebenso in der Klasse Libvirtadapter über einen Konstruktoraufruf gekapselt:

```
LibvirtVMInformation lvi = new LibvirtVMInformation(hostAddress,vmName);
```

Die Umsetzung dort ähnelt der vorigen, nur das noch ein zusätzlicher Schritt nötig ist, um die Verbindung zur virtuellen Maschine an ein Objekt der Klasse Domain zu binden:

```
Connect conn = new Connect(hostAddress);
domain = conn.domainLookupByName(vmName);
```

Der Name des Hosts sowie der Name des Virtualisierers lassen sich über Methoden aus der Klasse Connect ermitteln. Auch diese Aufrufe sind für den Libvirtadapter über die LibvirtHostInformation in den Methoden hostname() und hosttype() gekapselt. Diese Informationen werden verwendet, um dem Nutzer über die Web-Oberfläche anzuzeigen, welche Maschine im Moment verwaltet wird.

4.2.2. Dynamische Werte des Virtualisierungshosts

Die Werte des Hosts, auf dem die virtuellen Maschinen laufen, werden über die Werte der einzelnen virtuellen Maschinen berechnet. D. h. die Prozessor- (bzw. Arbeitsspeicher-) Auslastung des Hosts wird aus den Prozessor- (bzw. Arbeitsspeicher-) Auslastungen aller auf dem Host laufenden virtuellen Maschinen berechnet. Zu diesem Zweck werden beim erstmaligen Aufruf des Adapters Verbindungen zu allen momentan laufenden virtuellen Maschinen erstellt. Diese Verbindungen werden in einem Array gehalten.

```
LibvirtVMInformation[] connectToAllVMs =
        new LibvirtVMInformation[domainsOnHost.length];

for (int j=0; j<domainsOnHost.length; j++) {
        connectToAllVMs[j] =
                new LibvirtVMInformation(hostAddress, domainsOnHost[j]);
}
```

domainsOnHost ist ein String-Array, in dem die Namen aller laufenden virtuellen Maschinen stehen. Da man über die libvirt-API zunächst nur die IDs der laufenden virtuellen Maschinen erhält, muss es wie folgt über domainsOnHost() in der Klasse LibvirtHostInformation erstellt werden:

```
int[] names = conn.listDomains(); //libvirt zeigt nur IDs der VMs
String[] namesString = new String[names.length];
for (int i=0; i<names.length; i++) {
        Domain d = conn.domainLookupByID(names[i]); //liefert VM zur ID
        String domName = d.getName(); //liefert den Namen der VM
        namesString[i] = domName;
}
return namesString;
```

Nun kann, wenn Werte des Hosts angefordert werden, zunächst über das Array mit den Verbindungen zu den virtuellen Maschinen iteriert werden. Entsprechend des angeforderten Updates wird die Summe der Prozessor- bzw. Arbeitsspeicherauslastungen der virtuellen Maschinen aus dem Array errechnet. Wie die einzelnen Werte für die virtuellen Maschinen zu Stande kommen, wird im folgenden Abschnitt erklärt.

4.2.3. Dynamische Werte der virtuellen Maschine

Zur Errechnung der aktuellen RAM-Auslastung erhält man über die libvirt-Bibliothek den momentan verwendeten Arbeitsspeicher in Kilobyte, sowie den insgesamt vorhanden Arbeitsspeicher, ebenfalls in Kilobyte. Wenn man diese beiden Werte durcheinander dividiert, mit 100 multipliziert und rundet, erhält man die Arbeitsspeicherauslastung in Prozent.

$$RAM\,Auslastung[\%] = round(100 \cdot \frac{verwendeter\,Arbeitsspeicher}{gesamter\,Arbeitsspeicher})$$

Da es leider nicht möglich ist, sich die Prozessorauslastung direkt ausgeben zu lassen, muss sie errechnet werden. Dazu wird eine Methode verwendet, die sie mit Hilfe zweier Messungen über die Zeit, die zwischen den Messungen liegt und die verbrauchte CPU-Zeit ermittelt. Laut der verwendeten Formel ergibt sich die Prozessorauslastung wie folgt (vgl. [Jon08]):

$$Prozessorauslastung[\%] = 100 \cdot \frac{\Delta CpuZeit[ns]}{\Delta Zeit[s] \cdot AnzahlCpuKerne \cdot 10^9}$$

4.3. Erweiterung der Plattform

Die Managementplattform wurde um die Möglichkeit erweitert, die Arbeitsspeicherauslastung des Hosts und der virtuellen Maschine auszulesen. Dazu waren Änderungen nötig, die hier nur kurz erklärt werden sollen. Die genauen Ergänzungen bzw. Ersetzungen im Quellcode können im Anhang, Abschnitt A.1, nachvollzogen werden.

Um die Arbeitsspeicherauslastung in der Web-Oberfläche zu visualisieren, waren Änderungen in der Datei /opt/vlmanagement/Code/src/WebUI/protected/pages/Home.php notwendig. Für die „Ampel“, die, wie auch bei der Prozessorauslastung, je nach Auslastung ein grünes, gelbes oder rotes Licht anzeigt, muss in der Funktion showSystem($id) folgender Fall hinzugefügt werden:

```
if($this->soap->systemHasElems($id, "Ram")){
        $ram = $this->soap->getRam($id);
        $ramUsage = $this->soap->getRamUsage($ram->id);
        $content = $content."RAM Usage: ".$ramUsage."%";

        $ramUsageDurch10 = $ramUsage/10;
        $usagePicRam = $this->getUsagePic($ramUsageDurch10);
        $content = $content."   ".$usagePicRam."<br>";
}
```

Mit Hilfe von JpGraph (zu beziehen unter `http://www.aditus.nu/jpgraph/`) wird für die History der Prozessor- und der Arbeitsspeicherauslastung ein gemeinsamer Graph erzeugt.

```
function sumPic($cpuHist, $ramHist, $picfile){
        $prod1=$cpuHist; //Daten für CPU-History
        $prod2=$ramHist; //Daten für RAM-History
        $graph = new Graph(700,300, "auto");
        $graph->SetScale('linlin',0,100);
        $graph->img->SetMargin(50,20,20,50);
        $graph->SetBackgroundGradient('red:1.1','green:1.6',GRAD_HOR,
            BGRAD_PLOT);
        $graph->ygrid->SetColor("azure3");
        $graph->title->Set("CPU and RAM usage");

        $lineplot1=new LinePlot($prod1);
        $lineplot1->SetColor("white");
        $lineplot1->SetWeight(2);
        $lineplot2=new LinePlot($prod2);
        $lineplot2->SetColor("mediumblue");
        $lineplot2->SetWeight(2);

        $graph->yaxis->title->Set("Usage in %");
        $lineplot1->SetLegend("CPU");
        $lineplot2->SetLegend("RAM");
        $graph->legend->SetLayout(LEGEND_HOR);
        $graph->legend->Pos(0.4,0.95,"center","bottom");
        $graph->xaxis->HideLabels();

        $graph->Add($lineplot1);
        $graph->Add($lineplot2);
        $graph->Stroke($picfile);
}
```

Der Graph muss noch in der Funktion showSystem($id) eingefügt werden:

```
$sumPic = $this->sumPic($this->soap->getSystemCpuHistory($id), $this->
    soap->getSystemRamHistory($id), "images/".session_id()."-".$id."-
    systemSumGraph.png");
$content = $content."<img src='images/".session_id()."-".$id."-
    systemSumGraph.png#".time()."'/>";
```

Die Funktionen „getSystemRamHistory(systemId)“ und „getSystemRamUsage(systemId)“ müssen noch in den Dateien

- /opt/vlmanagement/Code/src/WebUI/protected/modules/soap.php

- /opt/vlmanagement/Code/src/Managementplattform2/src/access/SoapAdapter.java
- /opt/vlmanagement/Code/src/Managementplattform2/src/application/PlatformInterface.java
- /opt/vlmanagement/Code/src/Managementplattform2/src/application/ConfigurationManager.java
- /opt/vlmanagement/Code/src/Managementplattform2/src/information/InformationManagement.java

eingefügt werden. Die konkrete Implementierung ist im Anhang, A.1, zu finden.
In der Datei /opt/vlmanagement/Code/src/WebUI/protected/modules/vlmib/HostPerf.php ist die Zeile

```
public $ramUsage;
```

einzufügen.

Die verschiedenen neuen Nachrichtentypen, Funktionsaufrufe mit Ein- und Rückgabetypen müssen noch in der Datei /opt/vlmanagement/Code/src/WebUI/wsdl.xml im XML-Format notiert werden.

4.4. Registrierung des Adapters

Um die Web-Oberfläche sowie die restliche Managementplattform von der Existenz des Adapters in Kenntnis zu setzen, sind einige Änderungen nötig. Die in diesem Abschnitt erläuterten Veränderungen werden im Anhang, Abschnitt A.2, genau dokumentiert.

Zunächst wird die Web-Oberfläche für den neuen Adapter angepasst. Hierzu wird in der Datei /opt/vlmanagement/Code/src/WebUI/protected/pages/Home.page der Button button_add_dummy dupliziert und entsprechend benannt. Die Tabellenzeile mit den Beschriftungen der Buttons wird angepasst und der soeben erstellte Button wird mit der Aufschrift „libvirt hinzufügen“ versehen.

Um ihm genug Platz zu bieten, muss das Feld, in dem in der Web-Oberfläche die Nachrichten erscheinen, etwas verschoben werden. Hierfür sind Änderungen in der Datei /opt/vlmanagement/Code/src/WebUI/vlgui.css nötig.

Um den Button mit einer Funktion zu belegen, muss in der Datei /opt/vlmanagement/Code/src/WebUI/protected/pages/Home.php in der Funktion button_clicked($sender, $param) ein Fall für den gerade angelegten Button eingefügt werden.

Die Methode addLibvirtHost muss nun noch in folgenden Dateien eingefügt werden:

- /opt/vlmanagement/Code/src/Managementplattform2/src/access/SoapAdapter.java
- /opt/vlmanagement/Code/src/WebUI/protected/modules/soap.php
- /opt/vlmanagement/Code/src/Managementplattform2/src/application/PlatformInterface.java
- /opt/vlmanagement/Code/src/Managementplattform2/src/application/ConfigurationManager.java
- /opt/vlmanagement/Code/src/Managementplattform2/src/access/AccessManager.java

In der Datei /opt/vlmanagement/Code/src/Managementplattform2/src/message/add/AddLibvirtHost.java muss eine neue Klasse AddLibvirtHost erstellt werden, durch die es Nachrichten vom Typ AddLibvirtHost gibt.

Für den Fall, dass eine Nachricht vom Typ AddLibvirtHost erkannt wird, muss der entsprechende Adapter gestartet werden. Dies wird in der Datei /opt/vlmanagement/Code/src/Managementplattform2/src/communication/Communicationmanagement.java festgelegt.

Die dazu nötigen neuen Nachrichtentypen und Funktionsaufrufe werden in der Datei /opt/vlmanagement/Code/src/WebUI/wsdl.xml notiert.

5. Zusammenfassung

Mit dieser Arbeit wurde die bestehende Managementplattform um einen Adapter zur Kommunikation mit libvirt erweitert. Damit ist es möglich, eine ganze Reihe von Virtualisierern zu verwalten, darunter Xen, Qemu, und KVM. Da libvirt als Open-Source-Projekt ständig weiterentwickelt wird, wird die Liste der unterstützten Virtualisierer in Zukunft wohl noch länger werden. Beispielsweise wird an der Uni Paderborn an einer Anbindung von VMware an libvirt gearbeitet (vgl. [Bol09]). Die Funktionen der Managementplattform wurden im Rahmen dieser Arbeit um die Anzeige der Arbeitsspeicherauslastung erweitert. Desweiteren wurde die Web-Oberfläche dahingehend erweitert, diese Funktion auch anzeigen zu können. Nachdem der Nutzer per Knopfdruck den zu verwaltenden Virtualisierer ausgewählt hat, bekommt er ihn links mit den laufenden virtuellen Maschinen in einer Liste angezeigt. Durch Klicken auf einen der Einträge erhält er Monitoringinformationen zur Prozessor- und Arbeitsspeicherauslastung des entsprechenden Systems textuell und in einem Graphen dargestellt (vgl. Abb. 5.1).

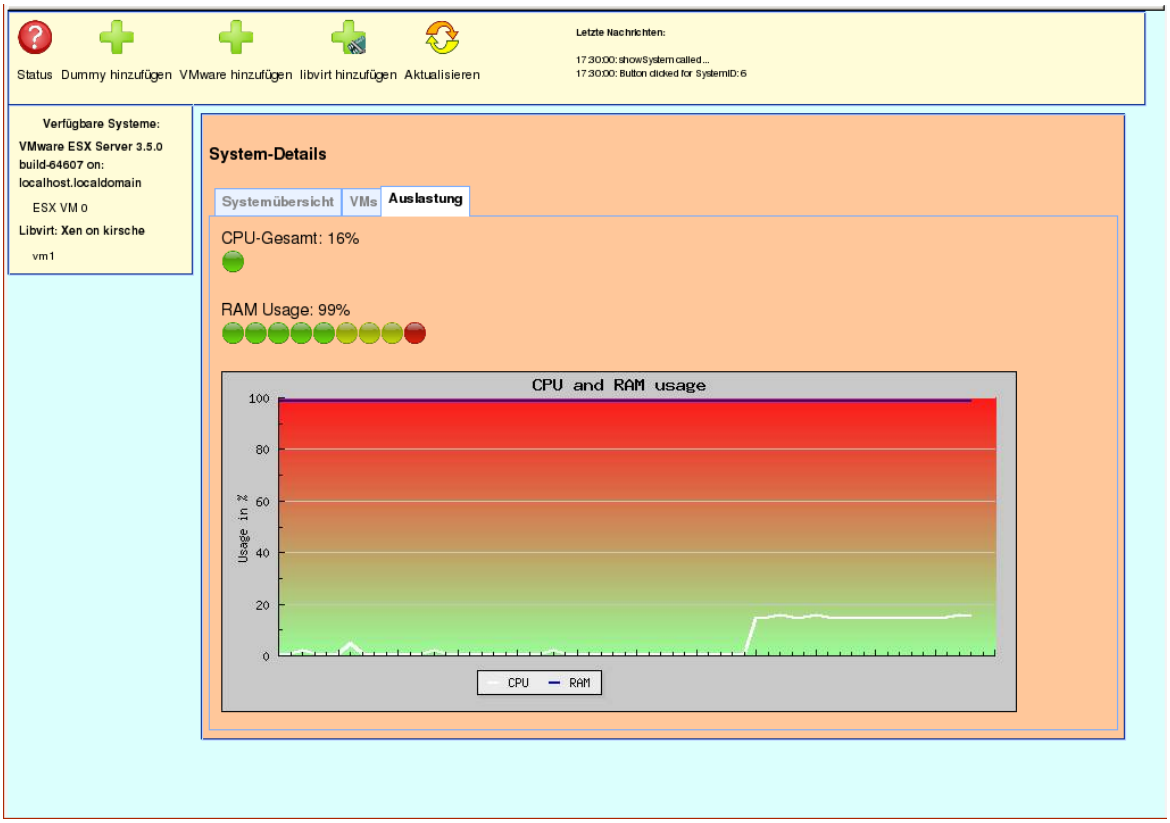

Abbildung 5.1.: Screenshot der Weboberfläche

5.1. Besondere Gegebenheiten durch die Verwendung von libvirt

Da libvirt eine Vielzahl von Virtualisierern unterstützt, muss es wohl als kleinste Schnittmenge ihrer Funktionen betrachtet werden. Ausgefallene Funktionen, die nur von einem kleinen Teil der Virtualisierer behandelt werden, bleiben von libvirt unbehandelt.

So konnte die Prozessorauslastung nicht direkt ausgelesen werden, sondern musste über eine mathematische Formel ermittelt werden. Um die Werte des Hosts zu ermitteln, musste jeweils über alle virtuellen Maschinen iteriert und die Summe ihrer Auslastungen gebildet werden. Dies kann eventuell zu Messungenauigkeiten oder Verlangsamungen im Ablauf führen.

5.2. Ausblick

Weiteren Erweiterungen des Adapters und der Managementplattform sind keine Grenzen gesetzt. Es gibt noch eine Vielzahl statischer Daten, die ausgelesen und visualisiert werden können. Davon werden momentan nur der Virtualisierertyp und der Hostname verwendet. Es bieten sich aber noch viele weitere Daten, wie die Größe des Arbeitsspeichers oder die Prozessorgeschwindigkeit, an, um in der Web-Oberfläche dargestellt zu werden.

Desweiteren ist in dieser Fassung der Name der virtuellen Maschine, die überwacht werden soll, noch statisch programmiert. Eine mögliche Erweiterung wäre, dass beim Klick auf den Knopf zum Hinzufügen des Adapters in der Web-Oberfläche ein Fenster geöffnet wird, in das der Nutzer die Hostadresse und den Namen der virtuellen Maschine eingeben kann. Auch eine Erweiterung um einen Auto-Discovery-Mechanismus ist denkbar. Damit müssten die virtuellen Maschinen nicht mehr von Hand eingegeben werden, sondern würden selbstständig gefunden werden.

In dieser Arbeit wurde vom Management nur die Monitoring-Seite betrachtet. Die Manipulation von Werten, wie die Anpassung des verwendbaren Arbeitsspeichers für jede virtuelle Maschine, geschieht hier vorerst noch nicht über die Managementplattform, sondern wird über externe Tools realisiert. Die Nachrichtentypen, die erforderlich sind, um Werte zu manipulieren sind bereits vorhanden; eine Erweiterung des Projekts um die Möglichkeit, Werte zu manipulieren ist also durchaus gegeben.

A. Anhang

A.1. Quellcode für Kapitel 4.3: Erweiterung der Plattform

- /opt/vlmanagement/Code/src/WebUI/protected/pages/Home.php
  - In der Funktion showSystem($id) muss für die Arbeitsspeicherauslastung die Ampelanzeige hinzugefügt werden, die, wie schon bei der Prozessorauslastung, je nach Auslastung ein grünes, gelbes oder rotes Licht anzeigt:

```
if($this->soap->systemHasElems($id, "Ram")){
        $ram = $this->soap->getRam($id);
        $ramUsage = $this->soap->getRamUsage($ram->id);
        $content = $content."RAM Usage: ".$ramUsage."%";

        $ramUsageDurch10 = $ramUsage/10;
        $usagePicRam = $this->getUsagePic($ramUsageDurch10);
        $content = $content."    ".$usagePicRam."<br>";
}
```

  - Für die History der Prozessor- und der Arbeitsspeicherauslastung wird mit Hilfe von JpGraph ein gemeinsamer Graph erzeugt.

```
function sumPic($cpuHist, $ramHist, $picfile){
        $prod1=$cpuHist; //Daten für CPU-History
        $prod2=$ramHist; //Daten für RAM-History
        $graph = new Graph(700,300, "auto");
        $graph->SetScale('linlin',0,100);
        $graph->img->SetMargin(50,20,20,50);
        $graph->SetBackgroundGradient('red:1.1','green:1.6',
            GRAD_HOR,BGRAD_PLOT);
        $graph->ygrid->SetColor("azure3");
        $graph->title->Set("CPU and RAM usage");

        $lineplot1=new LinePlot($prod1);
        $lineplot1->SetColor("white");
        $lineplot1->SetWeight(2);
        $lineplot2=new LinePlot($prod2);
        $lineplot2->SetColor("mediumblue");
        $lineplot2->SetWeight(2);

        $graph->yaxis->title->Set("Usage in %");
        $lineplot1->SetLegend("CPU");
        $lineplot2->SetLegend("RAM");
        $graph->legend->SetLayout(LEGEND_HOR);
        $graph->legend->Pos(0.4,0.95,"center","bottom");
        $graph->xaxis->HideLabels();
```

```
        $graph->Add($lineplot1);
        $graph->Add($lineplot2);
        $graph->Stroke($picfile);
}
```

- Der Graph muss noch in der Funktion showSystem($id) eingefügt werden:

```
$sumPic = $this->sumPic($this->soap->getSystemCpuHistory($id),
    $this->soap->getSystemRamHistory($id), "images/".session_id
    ()."-".$id."-systemSumGraph.png");
$content = $content."<img src='images/".session_id()."-".$id."-
    systemSumGraph.png#".time()."'/>";
```

- /opt/vlmanagement/Code/src/WebUI/protected/modules/soap.php
  - Hier müssen zwei neue Funktionen eingefügt werden.

```
public function getSystemRamHistory($systemId){
        return (array) $this->client->getSystemRamHistory(
            $systemId)->item;
}

public function getSystemRamUsage($sysId){
        return $this->client->getSystemRamUsage($sysId);
}
```

- /opt/vlmanagement/Code/src/Managementplattform2/src/access/SoapAdapter.java
  - Zwei weitere Methoden müssen hinzugefügt werden:

```
@WebMethod(operationName="getSystemRamHistory")
public int[] getSystemRamHistory(int systemId){
        return this.am.pi.getSystemRamHistory(systemId);
}

@WebMethod(operationName="getSystemRamUsage")
public int getSystemRamUsage(int systemId){
        return this.am.pi.getSystemRamUsage(systemId);
}
```

- /opt/vlmanagement/Code/src/Managementplattform2/src/application/PlatformInterface.java
  - Hier müssen Funktionen mit den gleichen Namen wie eben eingefügt werden:

```
public int[] getSystemRamHistory(int systemId){
        return this.cm.getSystemRamHistory(systemId);
}

public int getSystemRamUsage(int systemId){
        return this.cm.getSystemRamUsage(systemId);
}
```

- /opt/vlmanagement/Code/src/Managementplattform2/src/application/ConfigurationManager.java

- Genau wie eben auch hier die beiden Funktionen:

```
public int[] getSystemRamHistory(int systemId){
        return this.am.getInformationManagement().
            getSystemRamHistory(systemId);
}

public int getSystemRamUsage(int systemId){
        return this.am.getInformationManagement().
            getSystemRamUsage(systemId);
}
```

- /opt/vlmanagement/Code/src/Managementplattform2/src/information/InformationManagement.java
  - Auch hier müssen die Funktionen noch hinzugefügt werden:

```
public int[] getSystemRamHistory(int systemId){
        Calendar cal = new GregorianCalendar(TimeZone.
            getTimeZone("ECT"));
        int now = (int) java.lang.System.currentTimeMillis();
        mib.VLSystem s = (mib.VLSystem) this.elements.get(
            systemId);
        Collection c = s.getStat(now-60000, now).values();
        int[] list = new int[c.size()];
        int i=0;
        for(Object o: c){
                list[i] = ((SystemData) o).getRamUsage();
    i++;
        }
        return list;
}

public int getSystemRamUsage(int systemId){
        mib.VLSystem sys = (mib.VLSystem) this.elements.get(
            systemId);
        SystemData sd = (SystemData) sys.getLastStat();
        return sd.getRamUsage();
}
```

- /opt/vlmanagement/Code/src/WebUI/protected/modules/vlmib/HostPerf.php
  - Einzufügen ist diese Zeile:

```
public $ramUsage;
```

- /opt/vlmanagement/Code/src/WebUI/wsdl.xml
  Folgende Zeilen müssen eingefügt werden:
  -

```
<message name="getSystemRamUsage">
        <part name="arg0" type="xsd:int"></part>
</message>
```

```
  <operation name="getSystemRamHistory">
          <soap:operation soapAction=""></soap:operation>
          <input>
                  <soap:body use="literal" namespace="http://
                      access/"></soap:body>
          </input>
          <output>
                  <soap:body use="literal" namespace="http://
                      access/"></soap:body>
          </output>
  </operation>


- <operation name="getSystemRamUsage" parameterOrder="arg0">
          <input message="tns:getSystemRamUsage"></input>
          <output message="tns:getSystemRamUsageResponse"></output
              >
  </operation>

  <operation name="getSystemRamHistory" parameterOrder="arg0">
          <input message="tns:getSystemRamHistory"></input>
          <output message="tns:getSystemRamHistoryResponse"></
              output>
  </operation>


- <message name="getSystemRamHistoryResponse">
          <part xmlns:ns4="http://jaxb.dev.java.net/array" name="
              return" type="ns4:intArray"></part>
  </message>

  <message name="getSystemRamHistory">
          <part name="arg0" type="xsd:int"></part>
  </message>


- <operation name="getSystemRamUsage">
          <soap:operation soapAction=""></soap:operation>
          <input>
                  <soap:body use="literal" namespace="http://
                      access/"></soap:body>
          </input>
          <output>
                  <soap:body use="literal" namespace="http://
                      access/"></soap:body>
          </output>
  </operation>

  <xs:complexType name="hostPerf">
          <xs:sequence>
                  <xs:element name="cpuUsage" type="xs:float"></xs
                      :element>
                  <xs:element name="id" type="xs:int"></xs:element
                      >
```

```
                    <xs:element name="name" type="xs:string"
                        minOccurs="0"></xs:element>
                    <xs:element name="vmCount" type="xs:int"></xs:
                        element>
                    <xs:element name="ramUsage" type="xs:float"></xs
                        :element> <!-- diese Zeile ist neu -->
            </xs:sequence>
    </xs:complexType>
```

A.2. Quellcode für Kapitel 4.4: Registrierung des Adapters

- /opt/vlmanagement/Code/src/WebUI/protected/pages/Home.page
  - Code für den Button „Dummy hinzufügen“ duplizieren (button_add_dummy) und entsprechend benennen
  - Tabellenzeile mit den Beschriftungen der Buttons entsprechend anpassen
- /opt/vlmanagement/Code/src/WebUI/vlgui.css
  - Um den gerade erstellten Buttons genug Platz zu bieten, muss das Feld für die Nachrichten etwas verschoben werden. Bei #messages ist der Wert von left: 500px (war 380px) und der Wert von width: 500px (war 800px).
- /opt/vlmanagement/Code/src/WebUI/protected/pages/Home.php
  - In der Funktion button_clicked($sender, $param) muss ein Fall (Case) für den gerade angelegten Button eingefügt werden:

    ```
    case "button_add_libvirt":
            $this->soap->addLibvirtHost(1,1);
            $this->CallbackClient->hide("ajax-loader");
            break;
    ```

    Die beiden Integers (1,1) geben für den Aufruf des libvirt-Adapters die Anzahl der Hosts und der darauf laufenden virtuellen Maschinen an.
- /opt/vlmanagement/Code/src/Managementplattform2/src/access/SoapAdapter.java
  - Die Methode addLibvirtHost muss noch registriert werden. Analog zu @WebMethod(operationName="addDummyHost") mit den darauffolgendem Code wird mit @WebMethod(operationName="addLibvirtHost") verfahren.
- /opt/vlmanagement/Code/src/WebUI/protected/modules/soap.php
  - Ähnliches geschieht hier. Analog zum Dummy-Host wird folgendes eingefügt:

    ```
    public function addLibvirtHost($hosts, $vms){
            $this->client->addLibvirtHost($hosts, $vms);
    }
    ```
- /opt/vlmanagement/Code/src/Managementplattform2/src/message/add/AddDummyHost.java
  - Eine entsprechende Datei AddLibvirtHost.java muss erstellt werden. Durch diese Klasse gibt es Nachrichten vom Typ AddLibvirtHost.

- /opt/vlmanagement/Code/src/Managementplattform2/src/communication/Communicationmanagement.java
  - Für den Fall, dass eine Nachricht vom Typ AddLibvirtHost erkannt wird, muss der entsprechende Adapter gestartet werden. In der Methode void processAddMessage(Message m) wird also folgender Fall eingefügt:

```
if(m instanceof AddLibvirtHost) {
        AddLibvirtHost nm = (AddLibvirtHost) m;
        Libvirtadapter da;
        Thread a = new Thread(da = new Libvirtadapter(
                this, nm.getHostCount(), nm.getVmsPerHost()));
        this.resourceAdapters.add(da);
        a.start();
}
```

- /opt/vlmanagement/Code/src/Managementplattform2/src/application/PlatformInterface.java
  - Folgende Methode wird eingefügt:

```
public void addLibvirtHost(int hosts, int vms){
        this.cm.addLibvirtHost(hosts, vms);
}
```

- /opt/vlmanagement/Code/src/Managementplattform2/src/application/ConfigurationManager.java
  - Folgende Methode wird eingefügt:

```
public void addLibvirtHost(int hosts, int vms){
        this.am.receiveFromApp(new AddLibvirtHost(hosts, vms));
}
```

- /opt/vlmanagement/Code/src/Managementplattform2/src/access/AccessManager.java
  - Folgende Methode wird eingefügt:

```
public void addLibvirtHost(int hosts, int vms){
        this.pi.addLibvirtHost(hosts, vms);
}
```

- /opt/vlmanagement/Code/src/WebUI/wsdl.xml
  Folgende Zeilen müssen eingefügt werden:
  -
```
<message name="addLibvirtHost">
        <part name="arg0" type="xsd:int"></part>
        <part name="arg1" type="xsd:int"></part>
</message>
```
  -
```
<message name="addLibvirtHostResponse"></message>
```
  -
```
<operation name="addLibvirtHost" parameterOrder="arg0 arg1">
        <input message="tns:addLibvirtHost"></input>
        <output message="tns:addLibvirtHostResponse"></output>
</operation>
```

```
- <operation name="addLibvirtHost">
        <soap:operation soapAction=""></soap:operation>
        <input>
                <soap:body use="literal" namespace="http://
                    access/">
                </soap:body>
        </input>
        <output>
                <soap:body use="literal" namespace="http://
                    access/">
                </soap:body>
        </output>
  </operation>
```

Abbildungsverzeichnis

Literaturverzeichnis

[Bit08] Bittner, Florian: *Eine erweiterbare Management-Plattform für Hostvirtualisierungslösungen*, Oktober 2008.

[Bol09] Bolte, Matthias: *VMware ESX driver announcement*, 2009. https://www.redhat.com/archives/libvir-list/2009-May/msg00431.html.

[HAN99] Hegering, Heinz-Gerd, Sebastian Abeck und Berhard Neumair: *Integriertes Management vernetzter Systeme: Konzepte, Architekturen und deren betrieblicher Einsatz*. dpunkt Verlag, Heidelberg, 3. Auflage, 1999.

[ibm05] *IBM Systems - Virtualization, Version 2 Release 1*, 2005. http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf.

[Jon08] Jones, Richard: *Virt-top documentation*, 2008. http://et.redhat.com/~rjones/virt-top/faq.html.

[lib09a] *libvirt: Bindings for other languages*, 2009. http://libvirt.org/bindings.html.

[lib09b] *libvirt: Remote support*, 2009. http://libvirt.org/remote.html.

[lib09c] *libvirt: The virtualization API*, 2009. http://libvirt.org/index.html.

[lib09d] *org.libvirt (Libvirt java 0.2.0 API Reference)*, 2009. http://libvirt.org/org/libvirt/package-summary.html.

[Mel07] Mellor, Ewan: *The Xen-API*, 2007. http://www.xen.org/files/xensummit_4/XenSummit_API_Slides_2007-04-18_Ewan.pdf.

[Pic09] Picht, Hans-Joachim: *XEN Kochbuch*. O'Reilly, Köln, 1. Auflage, 2009.

[xen08a] *Xen Architecture Overview*, 2008. http://wiki.xensource.com/xenwiki/XenArchitecture?action=AttachFile&do=get&target=Xen+Architecture_Q1+2008.pdf.

[xen08b] *Xen Wiki - XenApi*, 2008. http://wiki.xensource.com/xenwiki/XenApi.