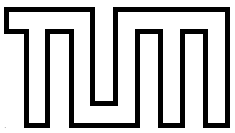


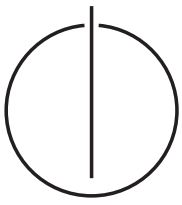
Technische Universität München

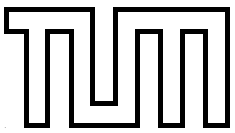
Fakultät für Informatik

Bachelorarbeit in Informatik

System- und anwendungsübergreifendes Monitoring am LRZ

Ralf Glauberman





Technische Universität München

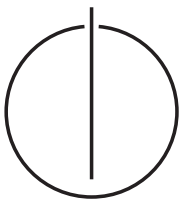
Fakultät für Informatik

Bachelorarbeit in Informatik

System- und anwendungsübergreifendes Monitoring am LRZ

Integrating system and application monitoring at LRZ

Bearbeiter:	Ralf Glauberman
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dr. Tobias Lindinger Wolfgang Fritz
Abgabedatum:	15. März 2011



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. März 2011

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Diese Bachelorarbeit behandelt die Integration verschiedener existierender Monitoringsysteme auf Basis einer gemeinsamen relationalen Datenbank. Während derzeit eine Vielzahl von zum Teil sehr ausgereiften Monitoringsystemen existiert, die jeweils einzelne Systeme und Dienste autonom überwachen können, existieren kaum Ansätze, die so gewonnenen Daten zentral zusammenzuführen um einen Gesamtüberblick über alle Systeme zu erhalten. Ziel dieser Arbeit ist es, ein derartiges System prototypisch für die Infrastruktur des Leibniz-Rechenzentrums (LRZ) zu implementieren und somit eine system- und anwendungsübergreifende Korrelation der Monitoringdaten zu ermöglichen. Dadurch wird es möglich, größere Störungen schneller zu diagnostizieren, die dafür ursächlichen Komponenten zu lokalisieren und die Probleme zu beheben. Weiterhin kann der Datenbestand Abhängigkeiten zwischen Systemen aufzeigen und dadurch bei der Planung von Wartungsarbeiten hilfreich sein. Zentrale Designaspekte bei der Implementierung sind ein hohes Maß an Skalierbarkeit und einfache zukünftige Erweiterbarkeit.

Inhaltsverzeichnis

1. Einführung

1.1. Defizite der bisherigen Monitoring-Lösungen

In jedem größeren Rechnernetz, sei es bei Firmen oder in Rechenzentren wie dem des Leibniz-Rechnerzentrums (LRZ), in dem eine Vielzahl von Systemen eingesetzt werden, ist ein effektives Monitoring heute unverzichtbar. Dabei werden je nach System eine Reihe von statischen und dynamischen Daten und Betriebsparametern erfasst, ausgewertet und gespeichert. Abhängig von den eingesetzten Monitoringwerkzeugen ist es dadurch möglich, fehlerhafte Komponenten schnell zu identifizieren, potentielle Engpässe aufzuspüren, Abschätzungen bezüglich des Wachstums der benötigten Leistungen, Speicherkapazitäten oder Netzwerkbandbreiten basierend auf deren historischer Entwicklung zu treffen und vieles mehr. Je nach zu überwachendem System hat sich daher eine große Menge an Monitorginsystemen am Markt etabliert. Die Auswahl reicht dabei von kostspieligen proprietären Systemen bis zu freier Software und von Speziallösungen, die nur ein oder wenige Systeme sehr detailliert überwachen können, bis zu erweiterbaren Monitoring-Frameworks, die das Monitoring vieler Komponenten auf einmal übernehmen und sich auch für weitere Komponenten anpassen lassen. Alle diese Monitoringprogramme sind dabei untereinander inkompatibel. Jedes System nutzt ein eigenes Modell zur Speicherung der gewonnenen Daten. Einige arbeiten dabei mit einer zentralen Datenbank, andere speichern die Daten dezentral auf den überwachten Rechnern. Jedes System nutzt dabei auch ein eigenes Dateiformat und eine eigene syntaktische Repräsentation der Daten. Leider ist es bei heterogenen Umgebungen in der Regel nicht möglich nur ein Monitoringsystem für alle zu überwachenden Systeme einzusetzen. Die Gründe dafür sind unter anderem:

- einige Netzwerkkomponenten lassen sich nur mit den vom Hersteller der Komponente gelieferten, proprietären Software überwachen und die verwendeten Schnittstellen werden vom Hersteller nicht offengelegt
- es existiert keine Software, die alle Systeme überwachen könnte oder eine Erweiterung der Software wäre zu aufwändig
- es existiert keine Software, die alle Anforderungen (z.B. Speicherung und Visualisierung von Verlaufsdaten, gezielte Suche nach bestimmten Datensätzen, plattformunabhängige Benutzeroberflächen, etc.) erfüllen kann
- aus historischen Gründen wird alte Monitoringsoftware weiterverwendet, weil eine Umstellung und gegebenenfalls die Datenübernahme in ein neues System zu aufwändig wären

In der Praxis führt dies dazu, dass bei größeren Netzwerken oft eine Vielzahl unterschiedlicher und inkompatibler Monitoringsysteme parallel eingesetzt werden. Dies ist jedoch nachteilig, da somit kein System einen Gesamtüberblick über den Status des ganzen Netzes geben kann. Um so problematischer wird dies dadurch, dass in der Regel viele Systeme und Dienste

voneinander abhängig sind, aufeinander aufbauen oder anderweitig auf einander einwirken, ein Ausfall eines Dienstes oder einer Komponente also oft auch den Ausfall einer ganzen Reihe anderer Systeme zur Folge hat. Im Falle eines Defekts werden nun viele Monitoringsysteme gleichzeitig eine große Zahl von Fehlermeldungen anzeigen. Da aber keines dieser Systeme alle Teilsysteme und deren Abhängigkeiten kennt, erhält der Nutzer auch keinerlei Hilfestellung, um die für einen größeren Ausfall ursächliche Komponenten zu identifizieren. Dies führt zu längeren Ausfallzeiten, bis der Fehler erkannt und behoben werden kann und damit zu höheren Kosten und einer schlechteren Servicequalität.

Es ist jedoch keineswegs einfach, dieses Problem zu lösen. Gerade da auf dem Markt bereits viele Systeme existieren, kann es nicht das Ziel sein, dieser Sammlung ein weiteres Instrument hinzuzufügen, welches versucht alles zu können und dabei doch an vielen Fronten scheitert. Daher soll in dieser Arbeit ein anderer Ansatz verfolgt werden.

1.2. Anforderungen an das neue Monitoring-System

Da, wie bereits erwähnt, die Chancen, ein für alle Systeme und Dienste passendes System zu finden, mit der Anzahl und Heterogenität der zu überwachenden Systeme schnell fällt, bzw. der Aufwand, um ein solches System zu schaffen, welches dabei auch alle Funktionen der bisher eingesetzten Systeme mit übernehmen kann, sehr schnell ansteigt, soll hier ein anderer Ansatz verfolgt werden. Statt die etablierten Systeme komplett zu ersetzten, bleiben diese bestehen und alle Funktionen wie gewohnt erhalten. Auch die Sammlung der Daten wird weiterhin ausschließlich von den bestehenden Systemen durchgeführt, der Wartungsaufwand auf Seiten der Clients und die Netzwerkbelastung werden also nicht durch zusätzliche Monitoringdienste oder doppelte Datensammlung erhöht. Statt dessen werden die wichtigsten Daten[1] aller bisherigen Monitoringsysteme über von diesen Systemen angebotene Schnittstellen oder direkt aus deren Datenspeichern exportiert und in ein einheitliches Datenmodell übertragen. Anschließend sollen diese Daten in einer zentralen Datenbank abgespeichert und somit für weitere Auswertungen zur Verfügung gestellt werden. Dabei ist es nicht das Ziel, so viele Daten wie möglich zu gewinnen. Vielmehr ist es erforderlich für alle unterschiedlichen Datensätze gezielt zu entscheiden, ob diese für eine zentrale Auswertung erforderlich bzw. hilfreich sind, oder ob es sich um Daten handelt, die zu spezifisch für ein bestimmtes System sind und die daher auch in Zukunft nur von den bereits existierenden Systemen verarbeitet werden sollen.

[1] welche Daten das im Einzellfall sind wird in einem späteren Kapitel detailliert behandelt

2. Bisher eingesetzte Monitoringsysteme

Am LRZ werden derzeit bereits viele unterschiedliche Systeme eingesetzt. Die wichtigsten, deren Daten für eine zentrale Auswertung besonders interessant sind und die daher in diese Arbeit auch einbezogen werden, sollen im Folgenden kurz vorgestellt werden.

2.1. Monitoring des VMware Clusters mittels VMware vSphere Client

Das LRZ verfügt über einige Virtualisierungs-Cluster auf Basis von VMware vSphere. Die Cluster bestehen derzeit[1] aus etwa 75 Serversystemen, auf denen knapp 700 virtuelle Systeme laufen. Die virtuellen Maschinen werden dabei zu etwa einem knappen Drittel unter eines Microsoft Windows Betriebssystem, zu gut zwei Drittel unter SUSE Linux Enterprise Server und zu wenigen Prozent unter anderen unixoiden Betriebssystemen betrieben. VMware vSphere basiert auf einem direkt auf der Hardware laufenden (bare metal) Hypervisor. Die einzelnen Cluster, physischen und virtuellen Maschinen können mittels des VMware vSphere Clients verwaltet und überwacht werden. Dieses Programm kann dabei nicht nur den aktuellen Status der Systeme anzeigen, sondern generiert ebenfalls Verlaufsdiagramme für bestimmte Betriebsparameter, ermöglicht das Starten oder Beenden von virtuellen Maschinen, zeigt Warnungen falls bestimmte Limits überschritten werden oder Fehler auftreten und dient auch zum Anlegen, Ändern oder Löschen von virtuellen Maschinen. VMware vSphere Client greift über einen speziellen Verwaltungsserver, den vCenter Server, auf die Daten zu. An diesem Server werden auch alle Monitoringdaten gesammelt und ausgewertet. Der schematische Aufbau kann der Abbildung 2.1 entnommen werden. VMware vSphere Client ist proprietär und steht ausschließlich als binäres Programm für Microsoft Windows basierte Systeme oder als Webfrontend (vSphere Web Access) zur Verfügung.

2.2. Monitoring der Linux-Rechner mittels LRZmonitor

Alle linuxbasierten Systeme am LRZ werden durch den LRZmonitor überwacht. Dabei handelt es sich um eine vollständige Eigenentwicklung des LRZ, die dort seit dem Jahr 2001 entwickelt und gepflegt wird. Dieses historisch gewachsene System besteht aus einer Reihe von Shellskripten, die von den zu überwachenden Systemen in regelmäßigen Abständen automatisch ausgeführt werden. Die Skripte sammeln dabei verschiedene Betriebsparameter, unter anderem Informationen über die eingesetzte Hardware, den Füllstand der eingebundenen Dateisysteme, die Netzwerkkonfiguration und auch statische Daten wie Inventar Nummern, Aufstellungsorte und die Namen des zuständigen Administratoren. Alle diese Informationen werden in Form von einigen Dateien auf einem von drei NFS-Dateiservern im Netzwerk gespeichert[2]. Zwei dieser Server kopieren die auf ihnen abgespeicherten Dateien

[1] Stand Januar 2011

[2] Es werden insgesamt drei Server eingesetzt um die Last besser zu verteilen

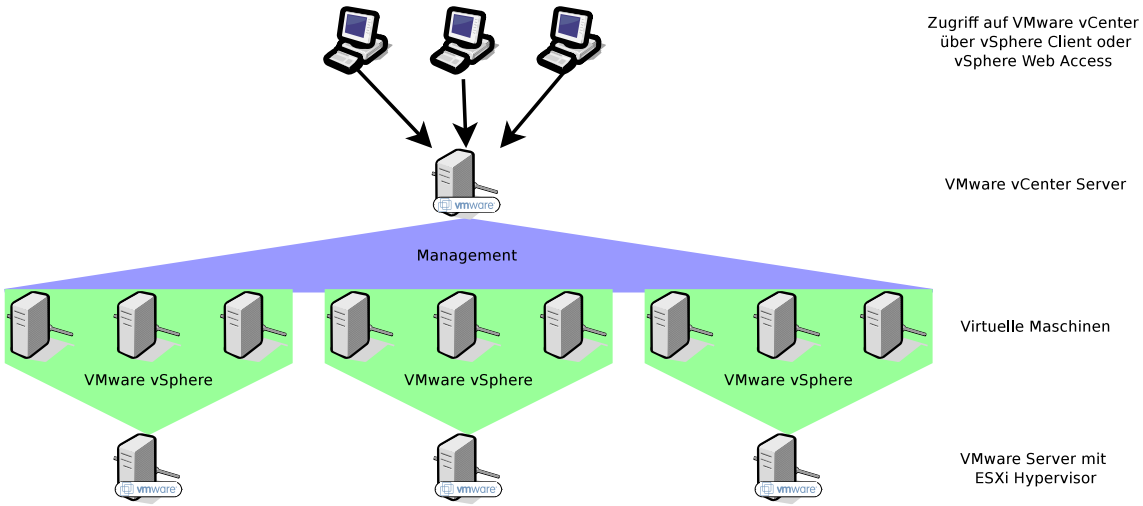


Abbildung 2.1.: Übersicht über ein VMware vSphere Cluster (vgl. [VMw09])

automatisch weiter auf den dritten Server, wo damit alle Informationen zusammenlaufen. Zur Visualisierung der Daten dient ein in PHP geschriebenes Webfrontend, welches direkt auf die Dateien zugreift, welche automatisch von dem NFS-Server auf den Webserver repliziert werden. Eine Datenbank zur Speicherung der Daten oder für eine schnelle Suche existiert nicht. Der Datenfluss im LRZmonitor ist in Abbildung 2.2 dargestellt. Der LRZmonitor dient in seiner derzeitigen Form sowohl zum Monitoring der linuxbasierten Systeme als auch als eine Art Inventarliste für alle darin erfassten Systeme. Mit diesem System werden derzeit etwa 1700 Computersysteme überwacht, deren Einsatzbereiche vom Supercomputer HLRB II[3], Web-, Mail-, Datenbank- und Bandarchivservern bis hin zu einfachen Arbeitsplatzrechnern der Mitarbeiter reichen. Der LRZmonitor stößt damit allmählich an die Grenzen seiner Skalierbarkeit und würde von einer Umstellung auf eine Datenbank zur Speicherung der Informationen sehr profitieren. Alle Quelltexte des LRZmonitors liegen vor und können bei Bedarf angepasst werden, dadurch ist eine leichte Abfrage der im System gespeicherten Informationen möglich.

2.3. Monitoring der Windows-Rechner

Für Inventarisierung und Monitoring der Microsoft Windows basierten Systeme am LRZ werden die beiden Softwaresysteme „System Center Configuration Manager“ (SCCM) und „System Center Operations Manager“ (SCOM) eingesetzt. Beide Systeme werden von Microsoft entwickelt, sind proprietär und stehen nur als ausführbares Programm zur Verfügung. Allerdings stellen diese Programme Programmierschnittstellen zur Verfügung, welche unter anderem den Zugriff auf die gespeicherten Daten erlauben. Die Programme haben unterschiedliche Einsatzgebiete und Anwendungsbereiche.

[3] Höchstleistungsrechner Bayern II

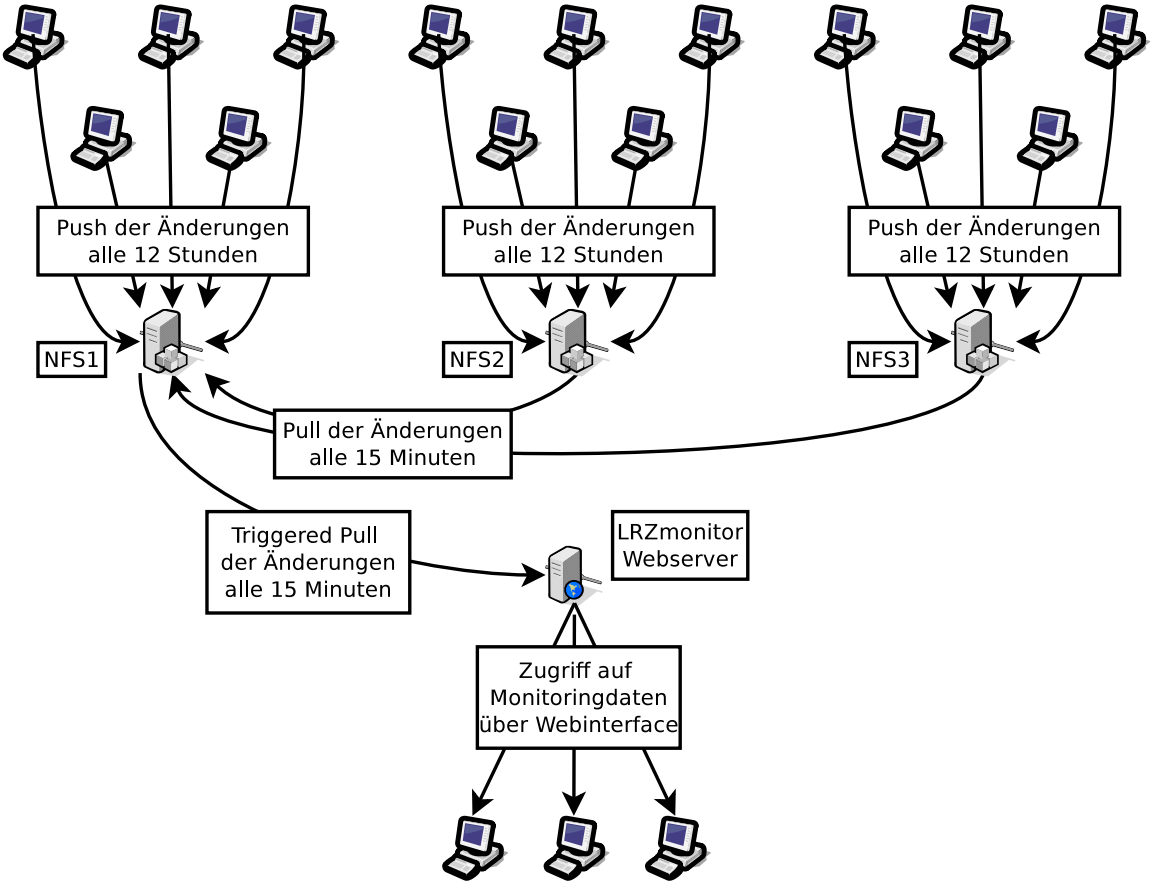


Abbildung 2.2.: Informationsfluss im LRZmonitor

2.3.1. System Center Configuration Manager

Microsoft System Center Configuration Manager ist ein Softwarepaket zur zentralen Administration und Verwaltung aller Windows-basierten Systeme eines Unternehmens. Es bietet unter anderem zentrale Softwareverteilung, Patch-Management, Betriebssysteminstallation und Verteilung von Konfigurationen für Betriebssysteme und Anwendungen. Außerdem beinhaltet das System eine Inventarliste aller damit verwalteter Computersysteme und der darauf installierten Software [4]. Dieses System eignet sich gut, um einen Überblick über die vorhandenen Computer, die darauf eingesetzte Software und deren Konfiguration und Lizenzierung zu erhalten, ermittelt aber keine dynamischen Daten über Auslastung und Zustand der überwachten Komponenten.

2.3.2. System Center Operations Manager

Microsoft System Center Operations Manager ist ein Monitoringsystem, dessen primärer Fokus auf der Überwachung aller Windows-basierter Computersysteme im Netzwerk liegt, allerdings können in der aktuellen Version auch Unix- und Linux-basiertes Systeme in die Überwachung einbezogen werden. Das Programm arbeitet mit einer Client-Server-Architektur, bei

[4] vgl. [Mic08b]

der auf jedem zu überwachenden System ein Clientprogramm permanent als Systemdienst ausgeführt wird, welches den Zustand von Hard- und Software und die Ereignisprotokolle des Systems überwacht und diese Informationen gegen einen Satz von Regeln, welche es von einem oder mehreren zentralen Serversystemen erhält, prüft. Diese Regelsätze werden entweder von Softwareherstellern als Vorlagen zur Verfügung gestellt oder können auch durch den Administrator selbst erstellt werden und definieren bestimmte Bedingungen, unter denen Warnmeldungen an den Systemadministrator verschickt werden sollen. Wenn das Clientprogramm feststellt, dass eine derartige Meldung verschickt werden soll, wird diese an den zentralen Server übermittelt. Die Serversoftware filtert diese Meldungen, archiviert sie und kann bei besonders kritischen Ereignissen die Warnungen auch direkt per E-Mail, SMS oder ähnlichen Diensten an den oder die zuständigen Administratoren weiterleiten[5]. Im Gegensatz zu den meisten anderen Monitoringsystemen gibt es bei SCOM keinen zentralen Speicher, in dem alle erfassten Informationen zusammenlaufen. Vielmehr werden die Informationen bereits durch die clientseitigen Programmkomponenten gefiltert und nur noch Warnmeldungen beim Eintritt vorher definierter Ereignisse an den zentralen Speicher verschickt. Dieser Ansatz ermöglicht es zwar die Netzwerkbelastung deutlich zu verringern und trotzdem beim Eintritt von kritischen Situationen sehr zeitnah zu reagieren, führt allerdings auch dazu, dass es keinen zentralen Speicher gibt, aus dem Rohdaten in das neu zu schaffende Monitoringsystem übernommen werden könnten.

2.4. Monitoring der Netzwerkkomponenten

Um einen Überblick über den Zustand aller Systeme eines Rechenzentrums und deren Abhängigkeiten von einander zu erhalten ist es natürlich unvermeidlich, auch die Netzwerkkomponenten, wie zum Beispiel Switches, Router, Wireless Access Points, et cetera zu betrachten. Da die für diese Systeme eingesetzten Monitoringprogramme aber sehr vielfältig und oft herstellerspezifisch sind, würde eine genauere Betrachtung all dieser Systeme den Rahmen dieser Arbeit sprengen. Daher wird sich diese Arbeit darauf beschränken, die Datenschemata und die Datenbank so zu konzipieren, dass sie leicht für die Anforderungen dieser Systeme erweiterbar sind. Das Monitoring dieser Komponenten soll ansonsten hier nicht weiter behandelt werden sondern bildet den Kern einer eigenständigen Arbeit, welche als [Sch11] veröffentlicht werden wird. Beide Arbeiten basieren auf kompatiblen Datenmodellen und arbeiten mit einer gemeinsamen Datenbank, um ein einheitliches neues Monitoringsystem zu schaffen.

[5] vgl. [Mic08a]

3. Entwurf des zentralen Monitoringsystems

Da das neu zu schaffende Monitoringsystem die Daten von vielen anderen Systemen erfassen und verarbeiten soll, welche in unterschiedlichsten Datenformaten und Datenmodellen vorliegen und auch zukünftige Erweiterbarkeit eine zentrale Anforderung an das System ist, ist eine genaue Planung und Modellierung des zu verwendenden Datenschemas vor Beginn der Implementierung unumgänglich. Die dabei durchgeführten Schritte sollen im Folgenden im Detail erläutert werden.

3.1. Modellierung des Datenschemas

Als erster Schritt muss ein konzeptionelles Datenschema definiert werden, welches universell genug ist, alle heute oder möglicherweise in der Zukunft anfallenden Daten zu repräsentieren. Dieses Schema ist dabei noch unabhängig von Details der jeweiligen Implementierung. Heute werden in der Regel zur Modellierung Schemata verwendet, welche mit den üblichen Praktiken der Objektorientierung versuchen, ein Abbild der Realität zu schaffen. Das Modell besteht also aus mehreren Klassen, Vererbung von einer Klasse auf ihre Kindklassen, Attributen der Klassen und den Beziehungen unter den Klassen. Zur Modellierung gibt es je nach Anwendungsgebiet unterschiedliche gebräuchliche Modelle und Beschreibungssprachen.

3.1.1. Common Information Model (CIM)

Eine weit verbreitete Möglichkeit um ein solches System zu modellieren ist das Common Information Model (CIM). Dabei handelt es sich um einen von der Distributed Management Task Force (DMTF)[1] entwickelten und verabschiedeten Standard für eine einheitliche Datenmodellierung für das Management von IT Systemen. Das CIM besteht dabei aus zwei Teilen, der CIM Spezifikation und dem CIM Schema. Die CIM Spezifikation beschreibt die verwendeten Sprach- und Namenskonventionen, eine formale Beschreibung des Modells und Möglichkeiten, CIM-Modelle auf andere Managementmodelle abzubilden. Das CIM Schema ist das eigentliche Modell. Es besteht aus formalen Beschreibungen für eine Reihe von Klassen, welche in ein universelles Core Schema und mehrere so genannten Common Schemas für verschiedene Anwendungsbereiche aufgeteilt sind. Außerdem ist es jedem, der das Modell verwendet, möglich, eigene so genannte Extension Schemas hinzuzufügen um das Modell um hersteller- oder anwendungsspezifische Eigenheiten zu erweitern. CIM bildet die Grundlage für das Web-Based Enterprise Management (WBEM), welches ebenfalls von der DMTF entwickelt wird und eine Reihe von Standardfunktionen zum Management von Computersystemen definiert. WBEM wird in vielen Enterpriseprodukten verwendet, unter anderem in den Managementsystemen von VMware vSphere und den Betriebssystemen Apple Mac OS X, SUSE Linux Enterprise Server, Red Hat Enterprise Linux, Ubuntu, Sun Solaris, HP-UX

[1] Die DMTF ist eine Normierungsorganisation zusammengesetzt aus Mitgliedern von über 200 Unternehmen aus der IT Industrie

und Microsoft Windows. Eine detaillierte Beschreibung von CIM und WBEM findet sich auf den Webseiten der DMTF unter [Dis10a] und [Dis10b].

CIM enthält also ein sehr umfangreiches Modell, in dem sehr viele Systeme bereits enthalten sind. Allerdings gibt es auch einige Eigenschaften, die gegen den Einsatz von CIM als Datenschema für das neu zu schaffende System sprechen. CIM enthält keinerlei Vorgaben für Dateiformate, Protokolle oder Schnittstellen. Es leistet also keinerlei Unterstützung bei der Implementation eines konkreten Systems. Auch für die Abbildung des Modells auf ein relationales Datenbankschema existieren keine Vorgaben. Daher werden unterschiedliche Systeme, auch wenn alle intern CIM zur Modellierung der Daten verwenden, kaum automatisch zusammenarbeiten können. Das Core Schema und die Common Schemas, welche einheitlich von der DMTF herausgegeben werden, können außerdem durch Extension Schemas nicht nur um neue abgeleitete Klassen erweitert werden, sondern es ist auch möglich, vorhandene Klassen zu ändern. So können z.B. Klassen innerhalb der Vererbungshierarchie verschoben werden, Klassen, Attribute und Beziehungen können aus dem Schema entfernt werden und Datentypen sowie die Semantik von Attributen können verändert werden. Damit ist es nicht möglich, sich in einem beliebigen auf CIM basierenden Datenmodell auf die Existenz oder Semantik irgendeiner Klasse, eines Attributs oder einer Beziehung zu verlassen, selbst wenn diese im Core Schema oder einem Common Schema modelliert sind. CIM modelliert sehr viele Strukturen, die für die zu schaffende Datenbank nicht relevant sind oder nicht abgebildet werden können. Dazu gehören zum Beispiel Aktionen und Operationen über die gespeicherten Daten und Trigger, die bei bestimmten Wertänderungen ausgelöst werden. Während dies für ein System, welches die in der Datenbank gespeicherten Daten abfragt und auswertet sicher interessante Aspekte sind, ist es für die Datenbank und die enthaltenen Daten nicht von Bedeutung und daher nicht Teil dieser Arbeit. Es wäre also für eine Implementation erforderlich, eine kleine Teilmenge von CIM auszuwählen, die verwendet werden soll. Anschließend müssten noch nicht modellierte Daten in Form eines Extension Schemas hinzugefügt und formal spezifiziert werden. All dies stellt einen erheblichen Aufwand dar. CIM erfordert außerdem viel Einarbeitung um sinnvoll damit zu arbeiten. Dies gilt auch für den Fall, dass man nur eine kleine Teilmenge verwenden will, was insbesondere problematisch ist, weil dieser Einarbeitungsaufwand von jedem verlangt würde, der später die Daten eines weiteren Monitoringsystems zur Datenbank hinzufügen will. Nur so könnte verhindert werden, dass Daten und Beziehungen mehrfach oder inkonsistent modelliert werden.

3.1.2. Eigene Lösung zur Modellierung

Aufgrund der oben genannten Probleme, um unnötige Komplexität zu vermeiden und dem Gedanken des KISS-Prinzips[2] zu folgen, wird CIM daher nicht zum modellieren des Datenschemas verwendet. Dies soll aber nicht heißen, dass das Rad ein weiteres mal neu erfunden wird. Vielmehr dienen die CIM Schemata in vielen Bereichen durchaus als Vorlage und Inspiration für das Schema. Allerdings wird darauf basierend ein neues, genau dem Anwendungszweck angepasstes und bei Bedarf erweiterbares Schema erstellt anstatt eine geeignete Teilmenge in den CIM-Schemata zu bestimmen. Außerdem wird auf die formale, maschinenlesbare Spezifikation verzichtet und statt dessen eine menschenlesbare Beschreibung verwendet, die durch UML-Diagramme visualisiert wird. Dies ist insofern ausreichend, dass das Schema nur als abstraktes Gedankenmodell für die Entwicklung dient und mit dem

[2] Akronym für „Keep it simple and stupid“, Prinzip das besagt, dass einfache Lösungen zu bevorzugen sind. Vgl. dazu auch Ockhams Rasiermesser

später zu erstellenden Entwurf des Datenbankschemas eine exakte, maschinenlesbare und für Details der Implementierung ohnehin wichtigere Spezifikation existiert.

Im Folgenden soll das erstellte Datenschema vorgestellt und wichtige Designentscheidungen und Konzepte erläutert werden. Das UML-Diagramm des Schemas findet sich in Abbildung A.3 im Anhang. Bei der Betrachtung der bisher erfassten Daten, gibt es einige Objekte, die offensichtlich sind und für alle zu überwachenden Komponenten existieren.

ComputerSystem

In diesem Fall war das auffälligste das *ComputerSystem*, repräsentiert durch die gleichnamige Klasse, also die Hardware als ganzes. Darunter fallen nicht nur PCs und Server, sondern auch z.B. Netzwerkswitches und andere Geräte. Alle diese Geräte haben bestimmte Eigenschaften (Attribute), z.B. eine Inventarnummer, einen Aufstellungsort, einen Hersteller, Produktbezeichnung und Seriennummer. Bei „echten“ Computern, also z.B. PCs, Servern, etc., kann man zudem die Art und Anzahl der Prozessoren und den verbauten Arbeitsspeicher als weitere Attribute hinzunehmen. Besondere Eigenschaften, die nur für VMware virtuelle Maschinen existieren, werden in der abgeleiteten Klasse *VMComputerSystem* erfasst.

OsSetup

Dem gegenüber steht die Softwareseite, also das Betriebssystem, dargestellt durch die Klasse *OsSetup*. Deren Attribute enthalten Informationen über das Betriebssystem, also z.B. Namen und Version, aber auch einige Konfigurationsparameter wie Hostnamen und Domänennamen sowie dynamische Daten, wie der Zeitpunkt des letzten Neustarts. Für einige Attribute, die nur bei linuxbasierten Betriebssystemen existieren, gibt es darüber hinaus die abgeleitete Klasse *LinuxOsSetup*, für VMware Hostsysteme existiert die abgeleitete Klasse *VMHost*. Offensichtlich besteht zwischen den beiden Klassen OsSetup und ComputerSystem eine Beziehung, denn eine Betriebssysteminstallation kann kaum ohne ein Computersystem existieren. Ebenso gehört eine Betriebssysteminstallation immer nur zu genau einem Computersystem. Das Computersystem kann allerdings, z.B. im Falle von einfachen Netzwerkswitches, gut ohne ein nennenswertes Betriebssystem existieren oder im Falle von multi-boot Systemen sogar mehrere Betriebssysteminstallationen beherbergen. Diese beiden Klassen bilden den Kern des gesamten Schemas, welches anschließend um diese Klassen herum erweitert werden kann, um noch nicht repräsentierbare Daten abzubilden.

DeviceInfo

Die Trennung zwischen Hardware und Softwarekonfiguration wird auch im Weiteren fortgesetzt. Peripheriegeräte, Erweiterungskarten, Laufwerke, etc. werden durch die Klasse *DeviceInfo* und davon abgeleitete Klassen repräsentiert. Dabei werden nur Informationen zur Hardware selbst, nicht zu deren Konfiguration erfasst. Beispiele für solche Klassen sind *GraphiccardInfo* für verbaute Graphikkarten, *MonitorInfo* für angeschlossene Bildschirme, *NicInfo* bzw. *VMNicInfo* für verbaute Netzwerkkarten sowie *VMDrive* und die davon abgeleiteten Klassen *VMHarddiskDrive*, *VMRemoveableDrive*, *VMFloppyDrive* und *VMCdDrive* für Laufwerke von VMware virtuellen Maschinen.

DeviceConfig

Die Softwarekonfiguration dieser Geräte wird in Klassen, welche von der Basisklasse *DeviceConfig* abgeleitet sind, repräsentiert. Beispiele dafür sind die Klassen *MonitorConfig* für Monitoreinstellungen, *KeyboardConfig* für Einstellungen für die angeschlossene Tastatur und *NicConfig* für die Konfiguration von Netzwerkkarten, z.B. MAC-Adressen und verwendete Treiber.

ServiceConfig

Neben der Konfiguration der angeschlossenen Hardware gibt es auf vielen Betriebssystemen eine Reihe von Diensten, deren Eigenschaften und Konfigurationen ebenfalls erfasst werden können. Dies geschieht durch die Klasse *ServiceConfig* sowie die davon abgeleiteten Klassen *TsmServiceConfig* für die Konfiguration des Tivoli Storage Manager Clients[3] und *LrzMonServiceConfig*, welches den LRZmonitor Monitoring Dienst selbst repräsentiert.

weitere Klassen

Diese Klassen beschreiben den Kern des Systems, daneben existieren noch eine Reihe weiterer Klassen für Details von Netzwerkverbindungen, weitere Eigenschaften im VMware Cluster, Dateisysteme und den Zugriff über das Netzwerk auf zentrale Speichersysteme. Diese sollen hier nicht im Detail beschrieben werden, sondern werden im Rahmen des Datenbankschemas genauer erklärt.

3.2. Entwurf der Datenbank

Der nächste Schritt in der Entwicklung des Datenbanksystems ist es nun, dieses Datenschema in ein anderes Schema zu überführen, welches sich in einer relationalen Datenbank implementieren lässt. Dies ist erforderlich, da relationale Datenbanken nur Tabellen, Datensätze in diesen Tabellen und deren Beziehungen zu einander kennen, nicht aber Konzepte wie Vererbung. Darüber hinaus ist es erforderlich Schlüsselattribute für alle Datensätze zu identifizieren oder auch erst zu definieren, durch die ein Datensatz eindeutig identifizierbar ist. Im Gegensatz zu objektorientierten Schemata, in denen ein Objekt in sich selbst eine Identität hat, gibt es im Datenmodell relationaler Datenbanken keine Möglichkeit zwei Datensätze, die in allen Attributen übereinstimmen, zu unterscheiden. Verfahren, mit denen diese Probleme gelöst werden können, sollen im Folgenden vorgestellt werden.

3.2.1. Entwurf des relationalen Datenbankschemas

Vererbung

Wie bereits erwähnt, gibt es in relationalen Datenbanken keine Vererbung. Vielmehr ist es erforderlich, die Vererbungshierarchie auf eine oder mehrere Tabellen abzubilden. Dafür gibt es drei gängige Varianten, die alle gewisse Vorteile und Nachteile haben[4]. Diese Varianten werden im Folgenden anhand des Beispiels in Abbildung 3.1 beschrieben. Diese Abbildung

[3] Der Tivoli Storage Manager ist ein Datensicherungsprogramm von IBM und wird am LRZ auf vielen Computern eingesetzt

[4] nach [LR06] und [Hor07]

zeigt einen Ausschnitt des Datenschemas, der für die Speicherung von Netzwerkadressen der Vermittlungsschicht verantwortlich ist, im Beispiel für IP-Adressen der Protokollversionen IPv4 und IPv6.

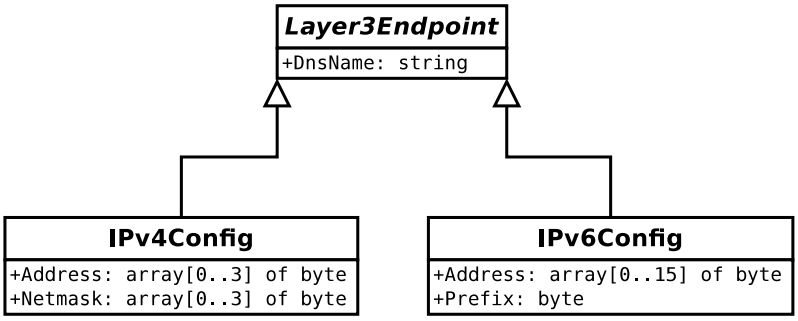


Abbildung 3.1.: Beispielausschnitt aus dem objektorientierten Schema

Eine Tabelle pro Klasse (Class Table Inheritance) Hierbei wird für jede Klasse, also sowohl für abstrakte als auch für konkrete Klassen, je eine Tabelle geschaffen, welche alle Attribute dieser Klasse enthält. Datensätze in Tabellen für untergeordnete Klassen enthalten Referenzen auf die zugehörigen Datensätze in der Tabelle, die die Elternklasse repräsentiert. Die Umsetzung des Beispiels aus Abbildung 3.1 ist in Abbildung 3.2 dargestellt. Zusätzliche Schlüssel- und Fremdschlüsselfelder wurden in der Abbildung nicht angegeben.

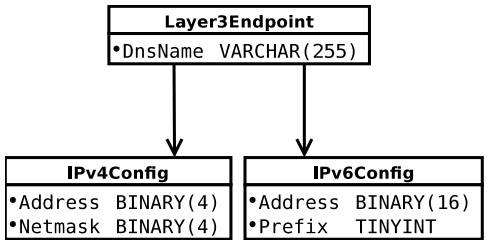


Abbildung 3.2.: Realisierung der Vererbung mittels Class Table Inheritance

Vorteile:

- sehr ähnlich zum objektorientierten Modell
- leicht erweiterbar um zusätzliche Unterklassen
- Abfragen, die nur Attribute der Oberklassen umfassen, sind leicht durchzuführen

Nachteile:

- Abfragen, welche Attribute einer Kindklasse und der Oberklasse umfassen, sind aufwändiger
- Bei der Speicherung von neuen Einträgen sind Schreibzugriffe auf mehrere Tabellen erforderlich. Dadurch kann die Leistungsfähigkeit der Datenbank negativ beeinflusst werden

- Abfragen, die alle Attribute nur einer Kindklasse aus der Oberklasse abfragen, sind aufwändig, da sich an den Attributen der Oberklasse selbst nicht erkennen lässt, um welche Kindklasse es sich handelt

Eine Tabelle pro konkreter Klasse (Concrete Table Inheritance) Bei dieser Variante wird nur für nicht abstrakte Klassen je eine eigene Tabelle geschaffen. Attribute von abstrakten Oberklassen werden in jeder Tabelle einer konkreten Unterklasse dupliziert. Vererbungen von konkreten Oberklassen werden wie bei der Class Table Inheritance durch Verweise realisiert. Die Umsetzung des Beispiels aus Abbildung 3.1 ist in Abbildung 3.3 dargestellt.

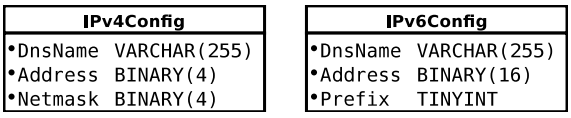


Abbildung 3.3.: Realisierung der Vererbung mittels Concrete Table Inheritance

Vorteile:

- ebenfalls leicht erweiterbar um zusätzliche Unterklassen
- Abfragen, die nur eine Klasse betreffen, erfordern meist keine Joins sondern nur der Zugriff auf eine Tabelle und sind dadurch schnell und übersichtlich
- Einfügeoperationen betreffen meist nur eine Tabelle und sind daher performanter

Nachteile:

- Abfragen, welche sich auf Attribute einer abstrakten Oberklasse beziehen und sich über alle Unterklassen erstrecken, sind recht aufwändig, da hierzu in mehreren Tabellen gesucht werden muss

Eine Tabelle für alle Klassen der Hierarchie (Single Table Inheritance) Bei dieser Alternative existiert nur eine Tabelle, welche alle Attribute aller Klassen enthält. Tabellenspalten, also Attribute, die für bestimmte Datensätze nicht definiert sind, werden einfach leer gelassen. Die Umsetzung des Beispiels aus Abbildung 3.1 ist in Abbildung 3.4 dargestellt.

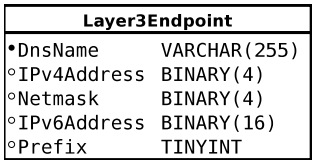


Abbildung 3.4.: Realisierung der Vererbung mittels Single Table Inheritance

Vorteile:

- Abfragen, die Daten aller Klassen betreffen, sind einfach und performant, da nur eine Tabelle betroffen ist
- Einfügeoperationen sind ebenfalls einfach und performant, da nur eine Tabelle betroffen ist

Nachteile:

- schlecht erweiterbar, da hierzu eine Änderung der bestehenden Tabelle erforderlich ist
- Abfragen, die Attribute nur einer Kindklasse abfragen, sind aufwändig, da sich an den Einträgen nicht mehr trivial erkennen lässt, um welche Kindklasse es sich handelt
- unter Umständen wird durch leere Tabellenzellen viel Speicherplatz verschwendet
- unübersichtlich, wenn sehr viele Klassen zu einer Tabelle verschmolzen werden
- Gültigkeitsregeln für Datensätze müssen leere Felder zulassen. Es ist nicht möglich zu verlangen, dass bestimmte Attribute für Datensätze aus einer bestimmten Kindklasse definiert sein müssen

Einige der oben angeführten Probleme lassen sich einfach lösen oder entschärfen. So kann es zum Beispiel sehr hilfreich sein, in der Tabelle der Oberklasse eine zusätzliche Spalte, einen sogenannten Diskriminator einzuführen, in der gespeichert wird, um welche konkrete Klasse es sich bei dem Datensatz handelt. Im Falle der Single Table Inheritance und wenn sich die Klasse nicht zuverlässig aus den gespeicherten Daten rekonstruieren lässt ist dies sogar erforderlich, da ansonsten keine verlustfreie Abbildung möglich wäre.

Fest steht jedoch, dass keine dieser Strategien den anderen immer und absolut überlegen ist. Vielmehr ist es erforderlich, Einzelfallentscheidungen zu treffen und dabei stets abzuwägen, welche Lösung für den konkreten Anwendungsfall, die Art der zu speichernden Daten, die zu erwartenden Abfragen und das eingesetzte Datenbankverwaltungssystem am besten geeignet ist. Daher wurde bei der Umsetzung des objektorientierten in ein relationales Schema für das neue Monitoringsystem auch nicht eine Lösung gewählt, vielmehr wurden unterschiedliche Lösungen für unterschiedliche Teilbereiche des Schemas gewählt und teilweise auch miteinander kombiniert. Beispielsweise wurde die Klasse VMNicInfo mit ihrer Basisklasse NicInfo nach dem Modell der Single Table Inheritance in einer Tabelle zusammengeführt, da die Unterschiede zwischen diesen beiden Klassen nur minimal sind. Es wäre aber nicht zielführend gewesen, DeviceInfo und alle davon abgeleiteten Klassen in nur einer Tabelle darzustellen, da diese Klassen kaum Gemeinsamkeiten haben und daher die resultierende Tabelle größtenteils leer wäre und aufgrund der vielen Attribute auch sehr unübersichtlich werden würde, weshalb hier die Verwendung von Class Table Inheritance besser geeignet ist. Auch hier sind jedoch Ausnahmen möglich. So haben die beiden von Layer3Endpoint abgeleiteten Klassen IPv4Config und IPv6Config auch nur wenig Gemeinsamkeiten, werden aber trotzdem in nur einer Tabelle erfasst. Dies liegt daran, dass sehr viele Anfragen sich auf beide Arten von IP-Adressen beziehen und der Vorteil, nur eine Tabelle abfragen zu müssen schwerer wiegt als der Nachteil durch die leeren Felder. Auch ist aufgrund der insgesamt weiterhin geringen Zahl von Attributen kein Verlust an Übersichtlichkeit zu befürchten. Concrete Table Inheritance wurde bei der Umsetzung nicht verwendet, dies liegt jedoch nicht darin begründet, dass diese grundsätzliche Nachteile hätte, sondern ausschließlich daran, dass die meisten abstrakten Basisklassen dieses konkreten Schemas trivial sind und die wenigen nicht trivialen Fälle sich durch eine der beiden anderen Varianten in diesem Fall besser modellieren ließen.

Primärschlüssel

Bei der Konzeption eines relationalen Datenbankschemas ist es außerdem erforderlich einen Schlüssel, durch den jeder Datensatz einer Tabelle eindeutig identifiziert wird, zu bestimmen. Dieser so genannte Primärschlüssel kann aus einem oder mehreren Attributen bestehen. Sofern es innerhalb des Datensatzes bereits Attribute gibt, die diesen eindeutig identifizieren, können diese als Primärschlüssel verwendet werden. Existieren solche Attribute nicht, kann auch ein künstlicher Primärschlüssel (Surrogate Key) in Form eines zusätzlichen Attributs, welches zum Beispiel einen Zähler enthält, geschaffen werden. Diese künstlichen Schlüssel bieten außerdem eine Reihe weiterer Vorteile[5]:

- sie bestehen nur aus einer Spalte der Tabelle. Dadurch werden Beziehungen einfacher darstellbar und Redundanzen vermieden
- während sich normale Attribute unter Umständen ändern können und dabei die Änderungen für alle Beziehungen weitergegeben werden müssen, ist eine Änderung eines künstlichen Schlüssels fast nie erforderlich.

Bei der Konzeption des relationalen Datenbankschemas für das neue Monitoringsystem wurden für fast alle Tabellen künstliche Schlüssel eingeführt, da es sich bei der Analyse der erfassten Monitoringdaten als sehr kompliziert herausgestellt hat, innerhalb der erfassten Daten eindeutige Schlüsselkandidaten zu finden, welche immer definiert sind und bei denen es nicht zu Verwechslungen kommen kann. Außerdem erfordert die Behandlung von Fällen, in denen mehrere Monitoringsysteme Daten zu einem Computersystem liefern, dass unter Umständen mehr als ein Datensatz pro realem System in der Datenbank existiert. Dies ist genauer im Kapitel 3.4 beschrieben.

3.2.2. Entwurf einzelner Tabellen

Nachdem im vorherigen Abschnitt die grundlegenden Verfahren zur Erstellung eines relationalen Datenbankschemas auf Grundlage eines objektorientierten Schemas gezeigt wurden, soll nun die konkrete Vorgehensweise beim Entwurf des Datenbankschemas für das neue Monitoringsystem vorgestellt werden. Dabei entsteht Schritt für Schritt aus dem objektorientierten Schema in Abbildung A.3 das relationale Schema in Abbildung A.2.

Direkte Übernahme aus dem Modell

Ein Großteil der Tabellen bzw. Tabellenspalten ist sehr einfach zu erstellen, da hierzu lediglich eine Tabelle pro Klasse erstellt werden muss. Dies gilt insbesondere bei Klassen, die weder von anderen Klassen mit vielen Attributen abgeleitet sind noch derartige Kindklassen besitzen. Bei diesen Klassen müssen lediglich geeignete Primärschlüssel definiert oder hinzugefügt werden und die Datentypen durch geeignete Datentypen des Datenbanksystems ersetzt werden. Attribute, die zur Speicherung von Beziehungen dienen, entfallen und werden durch geeignete Fremdschlüssel ersetzt. Ein schönes Beispiel für eine derartig leicht zu erstellende Tabelle ist die Tabelle ComputerSystem, welche mit den oben angegebenen Änderungen direkt aus der gleichnamigen Klasse erstellt wurde.

[5] vgl. auch [LR06]

Die Tabelle IpConfig

Andere Klassen sind etwas schwieriger zu übersetzen und erfordern dabei auch die Techniken aus Kapitel 3.2.1. Ein Beispiel hierfür sind die Klassen IPv4Config, IPv6Config sowie ihre abstrakte Oberklasse Layer3Endpoint. Diese Vererbungshierarchie wurde zu einer einzigen Tabelle mit Namen IpConfig, die alle Attribute der Klassen enthält. Um feststellen zu können, welcher Klasse die gespeicherten Daten angehörten, wurde der Tabelle eine zusätzliche Spalte ipVersion als Diskriminator hinzugefügt. Streng genommen wäre das hier zwar nicht erforderlich gewesen, da bei Datensätzen der Klasse IPv4Config die Spalten der Klasse IPv6Config frei bleiben und umgekehrt, wodurch sich die Information, zu welcher Klasse der Datensatz gehörte, auch rein aus den Daten regenerieren lässt, jedoch benötigt der Diskriminator kaum zusätzlichen Speicherplatz und vereinfacht die Formulierung von Suchanfragen, welche nur Daten einer bestimmten Klasse betreffen. Das Verschmelzen der Klassen erzeugt zwar in der Tabelle viele leere Spalten und vergrößert dadurch den Speicherplatzbedarf, jedoch ist für die in der Praxis häufige Suchanfrage nach allen IP-Adressen eines Systems nur noch eine Suche innerhalb einer Tabelle erforderlich.

Vereinfachungen des Modells

Andere Klassen wurden auch verschmolzen, obwohl sie nicht Teil der gleichen Vererbungshierarchie sind. So wurde zum Beispiel die Klasse KeyboardConfig in die Klasse OsSetup integriert. Dies stellt nicht nur eine Vereinfachung des Modells dar, sondern ändert auch die Semantik, denn während es nach dem objektorientierten Modell möglich ist, an einem Computer mehrere Tastaturen mit unterschiedlicher Konfiguration angeschlossen zu haben, ist dies nach der Verschmelzung im Datenbankschema nicht mehr möglich. Dies hat jedoch für die praktische Anwendbarkeit keine Auswirkungen, da eine solche Konfiguration in der Praxis nicht zu erwarten ist.

Die Klasse LinuxOsSetup wurde ebenfalls mit ihrer Elternklasse OsSetup zu nur einer Tabelle verschmolzen. Auch hier bleiben einige Felder häufig ungenutzt, dies wird aber ausgeglichen durch die verringerte Komplexität des Systems durch weniger Tabellen und die dadurch einfacheren Suchanfragen. Aus den gleichen Gründen wurden auch die Klassen NicInfo und VMNicInfo sowie die Klassen StorageDevice, PhysicalVolume und Filesystem zu je einer Tabelle zusammengeführt.

Die Klasse ComponentStateColor konnte entfallen, da sie in einer 1:1-Beziehung mit der Klasse LrzMonServiceConfig steht und somit komplett in deren Tabelle übernommen werden kann.

Die Klasse Fileserver enthält keine Attribute sondern definiert lediglich eine Beziehung zwischen den Klassen SharedStorage und OsSetup und konnte daher in der Datenbank entfallen.

Alle abstrakten Basisklassen, die keine Attribute definieren und lediglich der Veranschaulichung der Vererbungshierarchie dienen, kommen im Datenbankschema nicht mehr vor. Dies sind die Klassen DeviceConfig, DeviceInfo, HierarchicalElement und ServiceConfig.

VMDrive und abgeleitete Klassen

Die größte Kombination unterschiedlicher Klassen zu nur einer Tabelle konnte im Bereich von VMware virtualisierten Laufwerken stattfinden. Dies betrifft die Klasse VMDrive sowie die davon abgeleiteten Klassen VMHarddiskDrive, VMRemoveableDrive, VMFloppyDrive

und VMCdDrive. Eine Kombination war hier naheliegend, da sich die Klassen einen großen Teil ihrer Attribute teilen oder im Fall von VMCdDrive und VMFloppyDrive sogar komplett identisch sind. Da hier nicht aus den gespeicherten Attributen zuverlässig die Art des Laufwerks bestimmt werden kann, war die Einführung eines Diskriminators, der dies speichert, in diesem Fall unumgänglich. Die Datenbank enthält jedoch auch einige Tabellen, die keiner Klasse im objektorientierten Schema entsprechen. Dies sind zum einen die Tabellen NicConfigParentDevice und OsSetupUsesFilesystem. Diese Tabellen sind reine Verknüpfungstabellen zur Realisierung von n:m-Beziehungen. Ähnlich verhält es sich mit der Tabelle ComputerSystemInFolder, jedoch existiert auch bei dieser Tabelle ein semantischer Unterschied zum objektorientierten Modell, in dem die Beziehung zwischen ComputerSystem und Folder keine n:m-Beziehung sondern nur eine n:1-Beziehung ist. Dies bedeutet, dass ein Computer in der Datenbank an mehreren Stellen der Ordnerhierarchie erscheinen kann. Dies mag zwar auf den ersten Blick etwas ungewöhnlich erscheinen, ist aber durchaus sinnvoll. Die Gründe dafür werden im Kapitel 3.2.3 noch genauer beschrieben.

Datenquellen und Zeitstempel

Als letzte große Änderung zwischen dem objektorientierten und dem relationalen Schema wurden jeder Tabelle die Spalten timestamp und datasource hinzugefügt. Erstere hält fest, zu welchem Zeitpunkt die Daten erfasst beziehungsweise zuletzt aktualisiert wurden. Sie kann genutzt werden, um veraltete Daten in der Datenbank zu erkennen und geeignet darauf zu reagieren. Die Spalte datasource hält fest, von welchem Monitoringsystem die Daten ursprünglich stammen. Dadurch wird es zum Beispiel deutlich einfacher, die Quelle von fehlerhaften Einträgen zu finden oder die Zuverlässigkeit der erfassten Daten auf Basis des erfassenden Systems zu bewerten.

3.2.3. Die Ordnerhierarchie

Eine kleine Besonderheit innerhalb der Datenbank bildet die Ordnerhierarchie. Diese besteht aus Ordnern, wie man sie aus den gängigen hierarchischen Dateisystemen kennt, welche entweder andere Ordner oder Computersysteme enthalten können. Es handelt sich also um einen baumförmigen Graphen, dessen innere Knoten Ordner aus der Tabelle Folder und dessen äußere Knoten entweder Ordner oder Computersysteme aus der Tabelle ComputerSystem sind. Es gibt unterschiedliche Arten von Ordnern, die durch die Spalte kind der Tabelle Folder unterschieden werden können, sich jedoch in Ihrem Verhalten bezüglich der Ordnerhierarchie nicht unterscheiden. Die Computersysteme unterscheiden sich von den Ordnern nicht nur dadurch, dass sie keine Unterelemente haben können, sondern auch darin, dass ein System in mehreren Ordnern, also an unterschiedlichen Stellen des Baumes enthalten sein kann. Die Ordnerhierarchie bildet Sortierungen der einzelnen Monitoringsysteme ab, dient aber primär der Sortierung und Kategorisierung der Systeme und kann zum Beispiel von einer graphischen Benutzeroberfläche, welche die erfassten Daten darstellt, genutzt werden, um dem Nutzer einen besseren Überblick über die Daten zu verschaffen. Die Möglichkeit, dass ein System in mehreren Ordnern dargestellt wird, ist dabei sehr hilfreich. Angenommen wir definieren einen Ordner, in dessen Unterordnern wir alle Computer nach den angebotenen Diensten einsortieren. Zum Beispiel gebe es einen Ordner für alle Webserver, einen Ordner für alle Datenbankserver und einen Ordner für alle Mailserver. Ein Computersystem, welches sowohl als Datenbankserver, als auch als Webserver genutzt wird, kann dabei in beiden Ord-

nern dargestellt und so durch den Benutzer immer leicht gefunden werden. Das Konzept ist also nicht, dass ein Computersystem in einem Ordner *enthalten* ist, sondern vielmehr, dass ein Computersystem in unterschiedlichen Ordnern *angezeigt* wird, welche unterschiedliche Sichtweisen auf den gleichen Datenbestand darstellen.

3.2.4. Das Metaschema

Die Tabelle DataSource

Neben den Tabellen, die aus dem objektorientierten Modell abgeleitet sind und direkt zur Speicherung der Daten der Monitoringsysteme dienen, gibt es auch noch ein paar Tabellen, die Daten über die Datenbank und ihr Schema selbst beinhalten. Die erste davon ist die Tabelle DataSource. Diese Tabelle definiert eine eindeutige Identifikationsnummer, einen Namen und eine Beschreibung für jede Datenquelle, aus der Daten in die Datenbank einfließen. Die oben bereits beschriebenen datasource-Spalten in den einzelnen Tabellen referenzieren dabei die IDs der Datenquellen der DataSource-Tabelle. Für jede Datenquelle sollte immer eine eindeutig ID vergeben werden. Eine Datenquelle ist jedes Monitoringsystem, aus dem Daten in die gemeinsame Datenbank einfließen. Beispiele sind z.B. der LRZmonitor oder das VMware Monitoring System. Ein Programm, welches Daten in die Datenbank einfügen möchte, muss daher zuerst in der DataSource-Tabelle seine ID anhand des eindeutigen Namens herausfinden und diese ID bei allen eingefügten Datensätzen mit angeben.

Die Tabelle Schema

Die zweite wichtige Tabelle des Meta-Schemas ist die Tabelle Schema. Diese hat den Zweck, zukünftige Erweiterungen zu vereinfachen. Dazu wird eine Revisionsnummer für das Datenbankschema definiert. Bei jeder Änderung oder Erweiterung des Schemas wird die Revisionsnummer erhöht. In der Tabelle Schema ist gespeichert, was die aktuelle Versionsnummer des Datenbankschemas ist und mit welchen älteren Versionen des Schemas diese Version noch ganz oder teilweise kompatibel ist. Außerdem wird eine Art Revisionsgeschichte gespeichert, in der angegeben ist, wann und durch wen die Änderungen am Datenbankschema jeweils durchgeführt wurden. In jeder Anwendung, die die Datenbank verwendet, muss hinterlegt sein, mit welchen Versionen des Datenbankschemas diese Anwendung kompatibel ist. Bevor eine Anwendung versucht mit der Datenbank zu arbeiten, muss sie anhand der Tabelle Schema überprüfen, ob die Version der Datenbank kompatibel ist. Dies soll anhand eines kleinen Beispiels verdeutlicht werden: Eine Anwendung wurde für die Version eins des Datenbankschemas entwickelt. In der Tabelle Schema finden sich folgende Einträge (nur relevante Spalten):

revision	**isCurrent**	**readCompatible**	**writeCompatible**
1	no	yes	no
2	no	yes	yes
3	yes	yes	yes

Die Datenbank liegt also momentan in Revision drei vor. Die Datenbank ist zu Programmen, die für Version zwei entwickelt wurden, voll kompatibel. Dies kann zum Beispiel bedeuten, dass der Datenbank nur neue Tabellen hinzugefügt wurden, bisherige Tabellen aber nicht geändert wurden. Somit können auch Programme, die die neuen Tabellen noch nicht kennen, die Datenbank voll nutzen. Die Datenbank ist auch zu Programmen, die für Version eins der Datenbank entwickelt wurden, noch teilweise kompatibel, allerdings dürfen

solche Programme nur noch aus der Datenbank lesen und keine Datensätze mehr einfügen, ändern oder löschen. Diese Situation tritt z.B. auf, wenn in eine bestehende Tabelle neue Spalten eingefügt werden, die nicht leer sein dürfen. Ebenso kann dies verwendet werden, wenn Tabellen durch inhaltlich identische aber nicht schreibbare Views ersetzt werden um die Kompatibilität zu erhalten. Auch der umgekehrte Fall ist denkbar, dass eine Anwendung für eine ältere Version zwar schreiben, aber nicht mehr lesen darf. Dies wäre zum Beispiel der Fall, wenn sich Spalten einer Tabelle inkompatibel geändert haben, Schreibzugriffe aber mit Hilfe von Triggern anpassen lassen.

Unsere Beispielanwendung müsste nun also als erstes die Schema-Tabelle prüfen und würde dabei feststellen, dass die Datenbank sich geändert hat, lesender Zugriff aber noch möglich ist. Sofern dies für den Zweck der Anwendung ausreicht, kann die Anwendung nun mit dem Zugriff auf die Datenbank fortfahren, anderenfalls muss die Anwendung abbrechen und darf nicht versuchen, Datensätze in der Datenbank zu verändern.

3.3. Abfrage der Daten aus den bestehenden Systemen

Nachdem in den vorangegangenen Kapiteln der Entwurf der Datenbank erläutert wurde, geht es in diesem Kapitel darum, diese Datenbank auch mit Inhalten zu füllen. Dazu sollen wie eingangs beschrieben die Monitoringdaten aus den Datenspeichern der vorhandenen Monitoringsysteme exportiert und übernommen werden. Die Vorgehensweise dabei unterscheidet sich natürlich von System zu System, aber die Datenübernahme aus den im Kapitel 2 vorgestellten Systemen bietet einen guten Überblick über mögliche Vorgehensweisen. Da es aber für eine zukünftige Erweiterbarkeit nicht hilfreich sein kann, wenn viele unterschiedliche Systeme und Programme Daten direkt in die Datenbank schreiben, wird im Folgenden zuerst ein generelles Schema vorgestellt, nach dem die Übernahme der Daten in die Datenbank ablaufen soll.

3.3.1. Konzeption der Abfrageagenten

Da die unterschiedlichen Monitoringsysteme, deren Daten übernommen werden sollen, jeweils eigene Datenmodelle, Speicherformate und Schnittstellen einsetzen ist es nicht möglich, die Daten einfach in die Datenbank zu übernehmen. Vielmehr müssen die Daten vorher gefiltert, auf Konsistenz überprüft und für das Datenmodell der gemeinsamen Datenbank angepasst werden. Dabei ist eine universelle Lösung leider nicht realisierbar sondern es muss für jede Datenquelle ein Programm individuell entwickelt werde, welches diese Aufgabe übernimmt. Diese Programme, die als Bindeglied zwischen den unterschiedlichen Datenquellen und der Datenbank fungieren, werden im Folgenden als Abfrageagenten bezeichnet. Die Abfrageagenten können in beliebigen Programmiersprachen entwickelt werden und auf beliebigen Computersystemen und Plattformen ausgeführt werden, sofern ein Zugriff auf die Datenbank möglich ist. Je nach Art des Monitoringsystems und gewünschter Aktualität der Daten gibt es unterschiedliche Methoden für den Aufruf des jeweiligen Abfrageagenten:

- In regelmäßigen zeitlichen Abständen durch einen Taskplaner, zum Beispiel alle fünfzehn Minuten
- Bei Änderung des Datenbestandes des Monitoringsystems ausgelöst durch dieses

Die erste Methode bietet den Vorteil, dass sie fast immer funktioniert und keine besondere Schnittstelle oder Unterstützung von Seiten des Monitoringsystems, welches die Daten zur Verfügung stellt, erforderlich ist. Allerdings hat diese Methode auch einige Nachteile. So sind die Daten nicht immer aktuell, da aktuellere Daten erst beim nächsten Programmdurchlauf übernommen werden. Da jeder Programmdurchlauf alle Datensätze auf Aktualisierungen überprüfen und diese gegebenenfalls übernehmen muss, erzeugt jeder Programmdurchlauf auch eine nicht unerhebliche Last auf die Datenbank und eventuell auch auf das abgefragt Monitoringsystem. Daher muss eine Abwägung zwischen der Aktualität der Daten in der Datenbank und der vertretbaren Belastung der involvierten Systeme getroffen werden.

Die zweite Methode, die direkte Übernahme nur der Änderungen, ist insofern deutlich überlegen, dass hierbei nur eine Verzögerung von einigen Sekunden bei der Übername der Daten aus dem Datenbestand des ursprünglichen Monitoringsystems entsteht. Auch werden beide Systeme viel weniger belastet, da nicht mehr alle gespeicherten Daten abgeglichen werden müssen um die Änderungen zu ermitteln, sondern die Änderungen schon bekannt sind und nur noch übernommen werden müssen. Leider kann diese Methode aber nicht immer angewendet werden, da hierzu die Unterstützung des abzufragenden Systems erforderlich ist. Dieses muss eine Möglichkeit bieten, ein anderes Programm bei Änderungen der Daten über die Änderung zu informieren, also irgendeine Form von Benachrichtigungssystem unterstützen. Die denkbaren Implementierungen davon sind sehr vielfältig und reichen von einem einfach Aufruf eines anderen Programms und Übergabe der Änderungen über alle Formen von Interprozesskommunikation bis hin zur Weiterleitung der Änderungen an einen anderen Server über das Netzwerk. Leider bieten viele Systeme keinerlei derartige Schnittstellen um die Anbindung externer Programme zu ermöglichen. In diesem Fall ist der vollständige Datenabgleich in bestimmten Zeitintervallen leider oft die einzige Möglichkeit.

Da die MySQL-Datenbank den Zugriff über das Netzwerk erlaubt, ist es nicht erforderlich, dass die Abfrageagenten auf dem gleichen Server wie die Datenbanken ausgeführt werden. Damit ergibt sich für das neue Monitoringsystem das in Abbildung 3.5 dargestellte Schema. Die Abbildung zeigt der Datenfluss im neuen Monitoringsystem von den Datenquellen

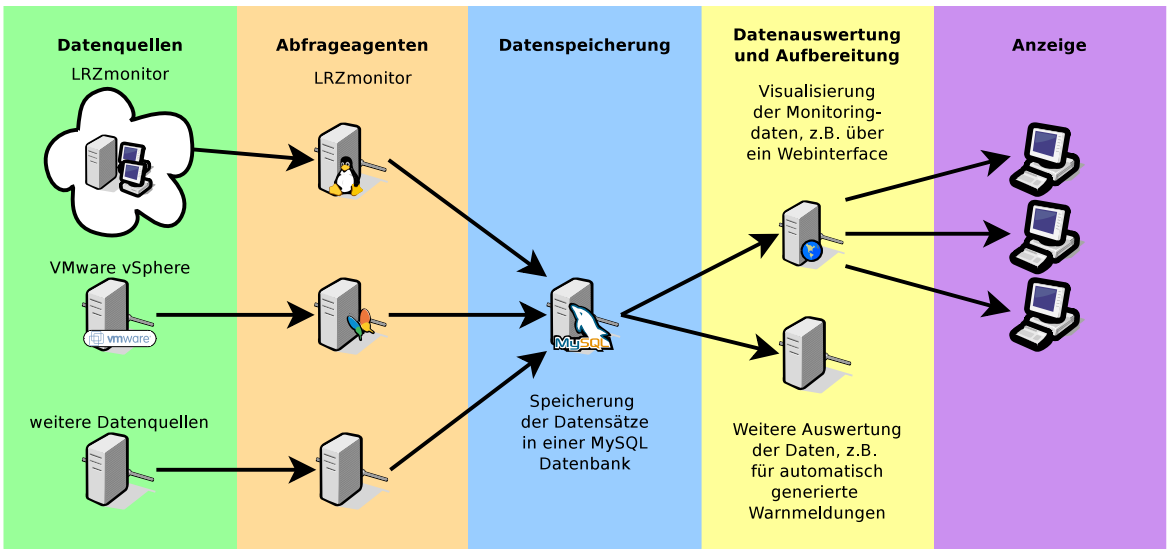


Abbildung 3.5.: Schematische Darstellung des Datenflusses im neuen Monitoringsystem

über die Abfrageagenten in die Datenbank und von dort zu unterschiedlichen Auswertesystemen bis hin zur Darstellung für die Nutzer. Selbstverständlich ist es auch möglich, dass ein Serversystem in dem Schema mehr als eine Funktion übernimmt.

3.3.2. Datenübernahme vom LRZmonitor

Nachdem die grundsätzlichen Möglichkeiten zur Übernahme von Daten aus den bisherigen Monitoringsystemen bereits beschrieben wurde, soll nun genauer darauf eingegangen werden, wie dies für die im Abschnitt 2 beschriebenen Monitoringsysteme realisiert werden kann.

Da der LRZmonitor alle seine Daten in einfach zu lesenden Textdateien speichert, wurden an die Zielarchitektur oder Programmiersprache des Abfrageagenten keine besonderen Anforderungen gestellt. Daher wurde entschieden, diesen in PHP zu entwickeln, da diese Sprache ausgereifte Möglichkeiten zum Zugriff auf MySQL Datenbanken bietet, nativ mit Zeichenketten umgehen kann und dabei einen großen Funktionsreichtum besitzt, viele Charakteristika einer modernen Programmiersprache, wie zum Beispiel automatische Garbage-Collection unterstützt und außerdem plattformübergreifend eingesetzt werden kann. Trotz des einfachen Zugriffs auf den Datenbestand stellte sich die Entwicklung eines Abfrageagenten für dieses System aber als recht kompliziert heraus. Die Ursache dafür lag primär in schlecht spezifizierten oder nicht eindeutigen Datenformaten, aus denen die Informationen geparsed werden mussten, begründet. Die Datenübername wurde so gestaltet, dass der Abfrageagent in regelmäßigen Zeitintervallen durch einen Taskplaner gestartet wird und daraufhin alle gespeicherten Daten des LRZmonitors mit dem Inhalt der Datenbank abgleicht und gegebenenfalls aktualisiert. Da dies, wie im Kapitel 3.3.1 beschrieben, einige Nachteile hat, wurde bei der Entwicklung darauf geachtet, dass sich der Abfrageagent leicht derart modifizieren lässt, dass er nur noch bei Änderungen aktiv wird und diese gezielt übernimmt. Da hierzu aber auch größere Änderungen am LRZmonitor erforderlich sind, deren Umsetzung einige Zeit benötigt, kann hier nur ein Ansatzpunkt für weitere Arbeiten geboten werden.

3.3.3. Datenübernahme aus dem VMware Cluster

Während der VMware vSphere Client selbst keine Schnittstellen anbietet, existiert eine ebenfalls von VMware produzierte und unter dem Namen „VMware vSphere PowerCLI“ veröffentlichte Sammlung von Tools, so genannten „Cmdlets“ für die Windows PowerShell. Damit ist es möglich in PowerShell Skripten alle Funktionen, die auch innerhalb des VMware vSphere Clients zur Verfügung stehen, zu nutzen und damit den Datenexport zu automatisieren. Die Windows PowerShell kann unter Microsoft Windows XP oder neuer eingesetzt werden. Eine vollständig plattformunabhängige Lösung war daher für diesen Abfrageagenten nicht möglich. Dieser Agent besteht aus zwei Programmteilen. Der erste Teil ist ein Skript für die PowerShell, welches alle benötigten Daten aus dem VMware Cluster extrahiert und in temporären Dateien zwischenspeichert. Anschließend wird ein in PHP geschriebenes Programm gestartet, welches die Daten aus diesen temporären Dateien einliest, überprüft, dem Datenschema der Datenbank anpasst und die Daten in diese einfügt. Dabei werden bei jedem Programmdurchlauf alle Daten des VMware Clusters in der Datenbank durch die neuesten Daten ersetzt. Auch dieser Agent wird in regelmäßigen Zeitabständen durch einen Taskplaner ausgeführt. Leider ist die vom VMware bereitgestellte Schnittstelle in Form der PowerShell Cmdlets sehr langsam sodass es nicht möglich ist die Daten in der Datenbank stets aktuell zu halten. Der Datenfluss der Monitoringdaten im VMware Cluster ist in Abbildung 3.6

dargestellt.

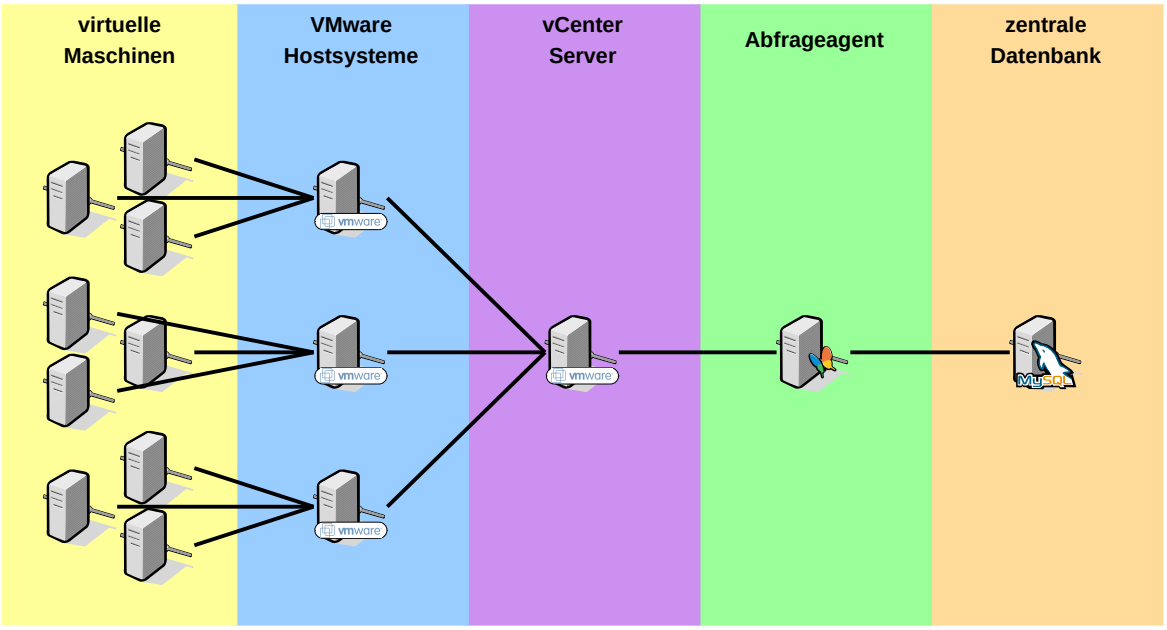


Abbildung 3.6.: Schematische Darstellung des Datenflusses im VMware Cluster

3.3.4. Datenübernahme von System Center Configuration Manager und System Center Operations Manager

Wie bereits im Kapitel 2.3 beschrieben enthält der System Center Operations Manager keine Informationen, die sich in das zentrale Monitoringsystem übernehmen ließen, da die zur Verfügung gestellten Benachrichtigungen bereits stark gefiltert sind und keine Rohdaten mehr enthalten. Diese Meldungen könnten zwar theoretisch im neuen System gespeichert werden, allerdings würde sich daraus kein Mehrwert für das System ergeben, da ein Vergleich oder eine automatische Auswertung dieser Meldungen kaum möglich ist und das reine Betrachten der Meldungen in der Originalsoftware ohnehin besser möglich ist. Aus diesem Grund wurde darauf verzichtet, einen Agenten zur Abfrage von Daten aus diesem System zu erstellen.

System Center Configuration Manager enthält zwar nutzbare Daten, welche ohne große Probleme übernommen werden könnten (zum Beispiel die Inventardaten), allerdings handelt es sich dabei primär um statische Daten welche wenig Rückschlüsse über den Betriebszustand der betroffenen Systeme oder deren Abhängigkeiten untereinander erlauben. Mit Blick auf die zukünftigen Anwendungsmöglichkeiten des neuen Monitoringsystems, den begrenzten zeitlichen Rahmen dieser Arbeit und den geringen Mehrwert, den diese Daten darstellen würden, wird auch hier auf eine Übernahme der Daten verzichtet. Allerdings sollte es keinen allzu großen Aufwand darstellen, diese Daten in Zukunft zu übernehmen, sofern dies notwendig werden sollte.

3.4. Kumulation der Datensätze unterschiedlicher Monitoringsysteme

Wenn die Daten unterschiedlicher Monitoringsysteme in einer zentralen Datenbank zusammengeführt werden ist es natürlich auch sinnvoll, diese Daten mit einander zu verknüpfen. Da, wie in Abbildung 3.7 dargestellt, einzelne Systeme oft durch mehr als ein Monitoringsystem überwacht werden, ist es erforderlich, dass alle Datensätze, welche das gleiche System, die gleiche Hardware- oder Softwarekomponente oder ein beliebiges anderes Datenbankobjekt beschreiben, zu nur einem Datensatz zusammengesetzt (kumuliert) werden, welcher mehr Informationen über das beschriebene Objekt enthält, als jeder einzelne Datensatz. Dies soll an

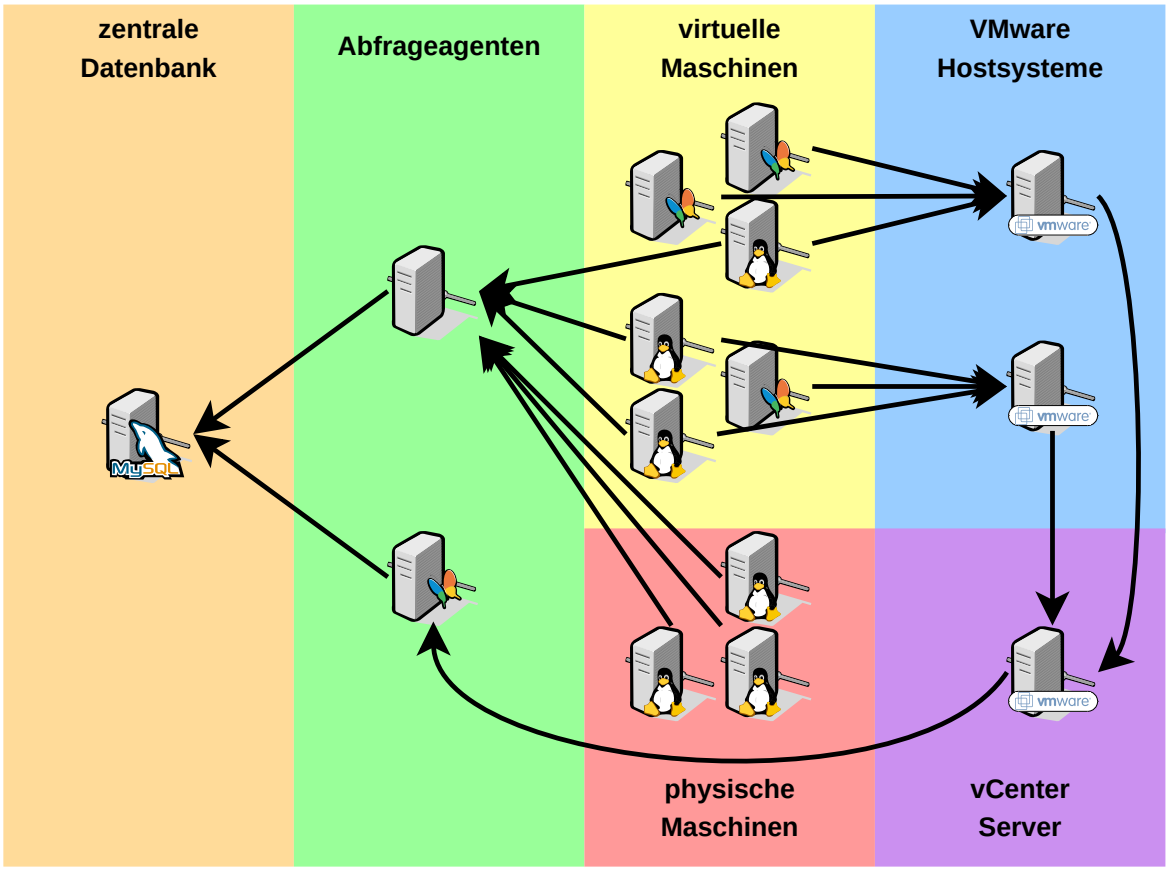


Abbildung 3.7.: Datenfluss bei Überwachung durch unterschiedliche Monitoringsysteme

einem kleinen Beispiel erläutert werden[6]:

Datenquelle	cpuCount	manufacturer	firstPwr	pwrState
lrzmon	1	-	15.01.2010	-
vmmon	1	VMware	-	on
kumulierter Datensatz	1	VMware	15.01.2010	on

Ohne diese Verknüpfung der Daten bietet die zentrale Datenbank kaum Vorteile gegenüber den einzelnen zugrunde liegenden Monitoringsystemen. Es gibt mehrere unterschiedliche

[6]Die Tabelle enthält nur einen Ausschnitt der Datenbanktabelle. Die Inhalte sind Beispiele und diesen nur der Verdeutlichung

Ansätze, wann diese Verknüpfung der Daten durchgeführt werden kann, welche alle gewisse Vorteile aber auch Nachteile besitzen.

beim Einfügen der Datensätze in die Datenbank Bei dieser Alternative werden die Datensätze direkt bei Einfügen in die Datenbank mit den darin bereits enthaltenen Daten verglichen und sofern eine Übereinstimmung mit einem bereits vorhandenen Datensatz festgestellt wird werden die beiden Datensätze zu nur einem Datensatz kombiniert und der bereits vorhanden Datensatz mit den neuen Daten aktualisiert. Die Suche nach den Duplikaten und die Vereinigung der Datensätze kann dabei entweder durch den einfügenden Agenten oder durch das Datenbankverwaltungssystem erfolgen.

Vorteile:

- die Datenbank enthält keine Duplikate, d.h. pro System oder Teilsystem gibt es nur einen Datensatz
- die Datenbank enthält weniger Datensätze und ist daher insgesamt kleiner und dadurch schneller
- Abfragen können einfacher erstellt werden, da Duplikate nicht mehr beachtet werden müssen

Nachteile:

- Einfügeoperationen werden langsamer
- Löschung von Datensätzen wird sehr kompliziert (jeder Agent sollte nur Daten löschen, die er auch eingefügt hat. Bei kombinierten Datensätzen ist aber nicht mehr zu erkennen, welche Daten von welchem Agenten stammen)
- sofern ein Datensatz, welcher bereits aus mehreren Datensätzen kombiniert wurde, erneut mit einem neuen Datensatz kombiniert werden muss, kann es sehr schwierig sein zu erkennen, welche Daten die besten und neuesten sind und daher in den kombinierten Datensatz einfließen sollen
- es ist nicht mehr zu erkennen, aus welcher Quelle eine bestimmte Information stammt, wodurch die Fehlersuche deutlich erschwert wird

bei der Abfrage der Datensätze aus der Datenbank Hierbei werden doppelte Datensätze in der Datenbank zugelassen und erst bei der Abfrage der Daten aus der Datenbank werden die Datensätze entsprechend kumuliert. Auch hier kann die Kumulation entweder durch das abfragende System oder bereits durch das Datenbankverwaltungssystem erfolgen.

Vorteile:

- Einfügeoperationen bleiben schnell
- Löschung von Datensätzen ist einfach, da jeder Agent die von ihm erstellten Datensätze löschen kann ohne dabei auf andere Agenten Rücksicht zu nehmen
- Änderung von Datensätzen ist einfach, da dies durch Löschen und neu Einfügen implementiert werden kann und diese beiden Operationen einfach sind
- bei Bedarf kann nach wie vor auf die Rohdaten der einzelnen Agenten zugegriffen werden um Fehler zu finden oder falls diese aus irgend einem anderen Grund erforderlich sind

- die Quelle jeder Information in der Datenbank bleibt nachvollziehbar

Nachteile:

- die Datenbank enthält Duplikate wodurch mehr Speicher benötigt wird
- die Abfrage von Daten aus der Datenbank wird komplexer, da hierbei die Daten noch kombiniert werden müssen
- Abfragen werden langsamer, da mehr Daten zu durchsuchen sind und diese auch noch verarbeitet werden müssen

in regelmäßigen Zeitabständen durch einen eigenen Dienst Auch bei dieser Lösung werden Duplikate in der Datenbank zugelassen, allerdings werden die Datensätze nicht erst bei der Abfrage kumuliert, sondern dieser Aufgabe wird an ein eigenes Programm delegiert, welches die Datenbank regelmäßig nach neuen Datensätzen durchsucht und diese geeignet behandelt.

Vorteile:

- die Datenbank enthält immer nur wenige Duplikate und bleibt dadurch kleiner
- Einfügen von Datensätzen ist einfach, da hierbei nicht auf andere Datensätze geachtet werden muss
- Abfrage von Datensätzen ist einfach, da hierbei idealerweise nicht mehr auf Duplikate geachtet werden muss
- die Kumulation kann durchgeführt werden, wenn die Last auf die Datenbank gerade nicht sonderlich groß ist

Nachteile:

- Löschung und mehrfache Kumulation sind genauso aufwändig wie bei der ersten Lösung
- auch hier ist die Quelle einer bestimmten Information nicht mehr nachvollziehbar
- es verbleiben einige Duplikate, welche bei Abfrage geeignet behandelt werden müssen
- der zusätzlich Dienst verursacht eine zusätzliche Belastung des Servers

Auch für das Programm, welches die Kumulation der Daten durchführt, gibt es zumindest bei den ersten beiden Alternativen je zwei Möglichkeiten:

automatisch innerhalb des Datenbankverwaltungssystems Diese Lösung bietet den Vorteil, dass alle Logik der Datenkumulation nur einmal vorhanden sein muss. Da sich alleine aufgrund der Daten oft nicht sagen lässt, welche Daten zu bevorzugen sind, falls unterschiedliche Monitoringsysteme widersprüchliche Daten liefern, ist es unumgänglich, dass die Kumulationslogik gewisse Kenntnisse über alle einfügenden Agenten besitzt. Da diese Logik also unter Umständen für neue Agenten angepasst werden muss, ist es sehr wünschenswert, diese nur einmal an zentraler Stelle zu pflegen und nicht in eine Vielzahl von Agenten oder abfragenden Systemen zu replizieren. Das Datenbankverwaltungssystem ist hierfür ein sehr guter Platz.

außerhalb der Datenbank durch das einfügende beziehungsweise abfragende Programm Diese Lösung hat den Vorteil, dass der Datenbankserver weniger belastet wird und

aufwändige Berechnungen auf andere Systeme ausgelagert werden können. Sofern die Logik in die abfragenden Systeme integriert wird besteht überdies der Vorteil, dass die Kumulation genau auf die Rahmenbedingungen und die für den jeweiligen Einsatzzweck erforderlichen Daten abgestimmt werden kann. Auch stehen bei der Entwicklung der Auswertesysteme in der Regel leistungsfähigere Programmiersprachen zur Verfügung als der von den Datenbankverwaltungssystemen angebotene SQL-Dialekt.

Nach Abwägung der oben genannten Alternativen stellte sich die Kumulation der Daten bei der Abfrage aus der Datenbank und durch das Datenbankverwaltungssystem als einzige sinnvolle Lösung heraus, da insbesondere die Probleme beim Löschen und Aktualisieren der Daten, welche sich bei den anderen Lösungsalternativen ergeben, nur sehr aufwändig gelöst werden könnten und der Vorteil eines zentral zu pflegenden Kumulationscodes die kleineren Nachteile deutlich überwiegt. Da aber Abfragen oft sehr viele Datensätze betreffen während Einfügen beziehungsweise Änderungen meist nur wenige Datensätze betrifft und überdies Aktualisierungen praktisch immer durch Hintergrundprozesse stattfinden, Abfragen aber oft auch durch interaktive Prozesse durchgeführt werden, bei denen der Benutzer auf ein Ergebnis wartet, wurde entschieden, dass es nicht sinnvoll ist, die aufwändige Arbeit des Findens von Duplikaten erst bei der Abfrage durchzuführen. Das Kumulieren mehrerer Datensätze zu einem einzigen geht vergleichsweise schnell, allerdings stellt das Finden aller Duplikate einen nicht unerheblichen Aufwand dar und benötigt daher Zeit. Aus diesem Grund wurde ein Mittelweg aus der Kumulation beim Einfügen und der Kumulation beim Abfragen gewählt. Hierbei werden Duplikate bereits beim Einfügen gesucht und zu kombinierende Datensätze in einer zusätzlichen internen Tabelle vermerkt. Der Inhalt dieser Tabelle wird beim Einfügen, Löschen und Ändern von Datensätzen automatisch durch das Datenbankverwaltungssystem gepflegt und aktuell gehalten und ist daher für die einfügenden Agenten vollständig transparent. Diese Tabelle wird durch das Datenbankverwaltungssystem genutzt um die Datensätze bei Abfragen schneller kumulieren zu können. Bei praktischen Tests mit der Datenbank zeigte sich, dass so sehr gute Ergebnisse beim Abfragen erzielt werden können und es ausreicht, eine derartige Tabelle für alle Duplikate in der Tabelle ComputerSystem zu pflegen, da Duplikate in anderen Tabellen anhand von deren Beziehungen auch unter Nutzung dieser Tabelle schnell gefunden werden können. Die wenigen Tabellen, welche keine geeigneten Beziehungen zur Tabelle ComputerSystem haben, sind ohnehin recht klein und stellen daher kein Problem dar. Die kumulierten Datensätze werden für die meisten Tabellen in Form von Views zur Verfügung gestellt. Sofern die darin enthaltenen Informationen nicht ausreichen ist natürlich auch ein direkter Zugriff auf die Tabellen mit den zugrunde liegenden unkumulierten Daten weiterhin möglich.

4. Implementierung

In diesem Kapitel sollen einige Details der Implementierung vorgestellt und genauer erläutert werden. Aufgrund des Umfangs der zu dieser Arbeit gehörenden Programme ist es leider nicht möglich, die gesamte Implementierung im Detail zu besprechen weshalb hier nur besonders wichtige Aspekte hervorgehoben werden können. Einige Details wurden außerdem bereits im Kapitel 3 vorweg genommen. Dies liegt vor allem daran, dass es sich bei Entwurf und Implementierung solch eines Systems nicht um einen streng sequentiellen Ablauf handelt sondern um einen iterativen Prozess. So wurden während der Implementierung von Prototypen immer wieder Defizite im Entwurf der Datenbank oder dem Konzept der Abfrageagenten gefunden und behoben. Zwar handelte es sich nach dem ersten Entwurf meist nicht mehr um große Änderungen, aber eine Anpassung der gewählten Datentypen sowie teilweise auch der Attribute war mitunter erforderlich. Die Ursache dafür lag oft auch daran, dass die vom LRZmonitor erfassten Daten nicht oder nur ungenügend formal spezifiziert oder dokumentiert waren wodurch sich deren genaue Semantik teilweise erst bei der Implementierung des dazugehörigen Abfrageagenten ergab. So wurden zum Beispiel die Felder für MAC-Adressen ursprünglich mit einer Länge von sechs Bytes definiert, wie dies bei allen Ethernet Adaptern verwendet wird. Erst beim Import zeigte sich, dass es auch MAC-Adressen mit einer Länge von vier beziehungsweise zwanzig Bytes gibt, welche bei IrDA[1] und Infiniband Adaptern Verwendung finden. Dies war beim ursprünglichen Entwurf nicht aufgefallen, da die hierfür analysierten Datensätze des LRZmonitors keine solchen Daten enthielten.

4.1. Implementierung der Datenbank

4.1.1. Auswahl eines Datenbankverwaltungssystems

Die erste wichtige Entscheidung bei der Implementierung der Datenbank bestand in der Wahl des zu verwendenden Datenbankverwaltungssystems. Die Auswahl hierbei ist groß und die Systeme unterscheiden sich zum Teil erheblich in Leistungsumfang, Leistungsfähigkeit, Portabilität und Preis. Aufgrund der Natur des Systems war von Anfang an klar, dass nur ein Datenbanksystem, welches den gleichzeitigen Zugriff durch mehrere Anwendungen und über Netzwerk erlaubt, geeignet ist. Weiterhin muss das System in der Lage sein, große Datenmengen effizient zu verwalten und gut skalieren, damit es auch bei zukünftigem Wachstum und Erweiterungen noch einsatzfähig bleibt. Des Weiteren ist ein offenes und portables System wünschenswert um sich nicht an einen Hersteller oder eine Plattform zu binden. MySQL erfüllte alle diese Kriterien und ist überdies kostenlos erhältlich, wodurch die Installation auf mehreren Entwicklungssystemen deutlich erleichtert wird, da keine separaten Lizenzen erworben werden müssen. Leider hat MySQL auch einige Nachteile. So werden derzeit zum Beispiel noch keine Integritätsbedingungen für die eingegebenen Daten unterstützt[2], sodass

[1] Infrarot Kommunikationsanschluss gemäß den Standards der Infrared Data Association

[2] Stand MySQL 5.1, ausgenommen die automatische Fremdschlüsselüberprüfung, vgl. „The CHECK clause is parsed but ignored by all storage engines“ in [O+11a]

die Konsistenz der eingegebenen Daten zwingend vor dem Einfügen in die Datenbank sicher gestellt werden muss. Dadurch ist es denkbar, dass inkonsistente Daten in die Datenbank gelangen können, sofern auch nur ein Agent die Daten nicht richtig überprüft. Trotz diesem nicht unerheblichen Defizit von MySQL fiel die Entscheidung diese zu benutzen, da die Anzahl der Abfrageagenten überschaubar ist und die Vorteile von MySQL als wichtiger angesehen werden.

MySQL bietet als Datenbankverwaltungssystem außerdem die Wahl zwischen unterschiedlichen so genannten Storage Engines, also Datenbankformaten. Jede dieser Engines hat wiederum Vorteile aber auch Nachteile und der Funktionsumfang unterscheidet sich zum Teil sehr deutlich. Daher ist es auch hier sinnvoll, sich vor der Entscheidung für eine Engine einige Gedanken zu machen um die optimale Engine für den jeweiligen Einsatzzweck zu bestimmen. Einige Engines sind für recht spezielle Einsatzszenarios vorgesehen und können daher aus der Auswahl ausgeschlossen werden. Übrig bleiben die beiden Datenbankengines MyISAM und InnoDB. Beide Engines sind sehr weit verbreitet und gut unterstützt und getestet. Während MyISAM lesende Abfragen sehr schnell bearbeiten kann und deshalb im Umfeld von Web-Applikationen gerne eingesetzt wird bietet InnoDB bei einer großen Zahl von gleichzeitigen Schreibzugriffen auf die Datenbank die bessere Leistung, da Schreibsperren bei dieser Engine nicht die ganze Tabelle betreffen sondern feingranular pro Datensatz verwaltet werden. Des Weiteren unterstützt InnoDB im Gegensatz zu MyISAM Transaktionen und referenzielle Integrität von Fremdschlüsseln, wodurch es einfacher wird die Integrität der gespeicherten Daten sicher zu stellen[3]. Wird zum Beispiel ein neuer Computer zur Datenbank hinzugefügt, so geschieht dies durch Einfügen von Datensätzen in mehrere Tabellen. Durch die Nutzung von Transaktionen kann sichergestellt werden, dass entweder alle oder keine dieser Einfügeoperationen erfolgreich abgeschlossen werden. Somit ist es nicht möglich, dass unvollständige Daten in der Datenbank verbleiben, wenn ein Datensatz nicht richtig eingefügt werden kann.

Aufgrund der großen Zahl der zu erwartenden Schreibzugriffe und der Vorteile, die sich durch Transaktionen und referenzielle Integrität von Fremdschlüsseln ergeben, wurde entschieden, InnoDB als Storage Engine für alle Tabellen der Datenbank zu verwenden.

4.1.2. Auslagern von umfangreichen und selten benötigten Informationen

Nach der Wahl eines geeigneten Datenbankverwaltungssystems und einer Storage Engine müssen auch am Datenbankschema noch einige Optimierungen vorgenommen werden. Ein Beispiel hierfür ist das Auslagern von besonders großen Spalten aus den Tabellen ComputerSystem und OsSetup in eine eigene Tabelle LongText. Dies scheint zuerst nicht sinnvoll zu sein, da hierdurch nur zusätzliche Komplexität in Form einer zusätzlichen Tabelle und zusätzlicher Beziehungen geschaffen wird und die Informationen genauso gut direkt in den jeweiligen Tabellen stehen könnten. In der Praxis ist die für Abfragen auf eine Tabelle benötigte Zeit direkt abhängig von der Größe der Tabelle, weshalb es sinnvoll ist, häufig verwendete Tabellen möglichst klein und damit performant zu halten. Die in die Tabelle LongText ausgelagerten Daten enthalten sehr umfangreichen Text, welcher aber nur sehr selten abgefragt werden muss. Aus diesem Grund ist es sinnvoller, diese Daten auszulagern und damit einen Großteil der Anfragen zu beschleunigen.

[3] nach [O+11b]

4.1.3. Automatische Kumulation der Datensätze

Wie bereits in Kapitel 3.4 beschreiben wurde ist es sinnvoll, wenn unterschiedliche Datensätze, die ein einziges System beschreiben, bereits innerhalb der Datenbank zu nur einem Datensatz zusammengeführt werden und abfragenden Systemen, welche nicht daran interessiert sind, von welchem Monitoringsystem eine bestimmte Information genau stammt, die so kumulierten Daten einheitlich präsentiert werden. In diesem Kapitel soll nun genauer darauf eingegangen werden, wie diese automatische Kumulation innerhalb des Datenbankverwaltungssystems realisiert wurde. Als erstes ist es erforderlich, ein Kriterium festzulegen, wie erkannt werden soll, wann zwei Datensätze das gleiche System beschreiben. Dies wäre sehr einfach, wenn absolut eindeutige Informationen, wie zum Beispiel eine Inventarnummer oder Seriennummer, für jedes Gerät vorhanden und in jedem Monitoringsystem konsistent zugänglich wären. Leider ist dies jedoch nicht der Fall, weswegen auf andere Kriterien ausgewichen werden muss. Da alle in der Datenbank enthaltenen Komponenten mit dem Netzwerk verbunden sind, kann davon ausgegangen werden, dass jede dieser Komponenten mindestens eine Netzwerkschnittstelle besitzt. Jede Netzwerkschnittstelle verfügt über eine MAC-Adresse, welche für diesen Einsatzzweck als hinreichend zufällig angesehen werden kann und daher verwendet werden soll. Zwei Datensätze beschreiben also den gleichen Computer, wenn mindestens eine MAC-Adresse in diesen beiden Datensätze übereinstimmt. Gleichzeitig wird aber ein Monitoringsystem nie einen Computer mehrfach überwachen, als zusätzliche Bedingung gilt also, dass zwei Datensätze nur dann verschmolzen werden dürfen, wenn sie nicht vom gleichen Monitoringsystem stammen. Leider besitzen vereinzelt zwei unterschiedliche Computersysteme, zum Beispiel aufgrund von Konfigurationsfehlern, die gleiche MAC-Adresse. Damit es in diesem Fall zumindest nicht zu einer falschen Verschmelzung und damit zu einer Vermischung von Daten unterschiedlicher Systeme kommen kann, wurde eine dritte Bedingung definiert welche besagt, dass eine MAC-Adresse nicht für die Kumulation herangezogen werden darf, sofern eines der Monitoringsysteme erkannt hat, dass diese Adresse von mehr als einem Computersystem verwendet wird. Hierbei handelt es sich um eine reine Vorsichtsmaßnahme, welche aber schwer zu erkennende Fehler und damit langwierige Fehlersuchen ersparen kann.

Ebenfalls erwähnt wurde bereits, dass die Abfrage der kumulierten Datensätze durch Verwendung einer zusätzliche Tabelle, deren Inhalt automatisch erzeugt wird, deutlich beschleunigt werden kann. Diese Tabelle, welche den Namen *ComputerSystemDuplicateTracking* trägt, enthält zwar keine neuen Informationen, also nichts, was nicht bereits anderweitig in der Datenbank gespeichert ist, kann allerdings Informationen schnell bereitstellen, welche sonst nur durch eine aufwändige und langsame Abfrage zugänglich wären. Da die Kumulation der Daten anhand der MAC-Adressen der Systeme vorgenommen wird, welche in der Tabelle NicInfo gespeichert werden, muss bei jeder Änderung an dieser Tabelle überprüft werden, ob sich diese Änderung auch auf die Kumulation auswirkt, ob also auch der Inhalt der Tabelle ComputerSystemDuplicateTracking geändert werden muss. Dies kann innerhalb des Datenbankverwaltungssystems durch so genannte Trigger realisiert werden. Ein Trigger ist eine Funktion, welche mit einer Tabelle verknüpft ist und bei Änderungen an deren Inhalt automatisch ausgeführt wird. Der erste Trigger behandelt das Einfügen von neuen Datensätzen in die Datenbank und ist wie folgt definiert:

```
1 CREATE TRIGGER nicInfoAfterInsert AFTER INSERT ON NicInfo
2 FOR EACH ROW BEGIN
3   CALL _UpdateDuplicateTracking(NEW.macAddress);
4 END
```

Dieser Trigger wird also ausgeführt, nachdem neue Datensätze eingefügt wurden und ruft für jeden hinzugefügten Datensatz die Funktion *_UpdateDuplicateTracking* mit der MAC-Adresse des neu hinzugefügten Datensatzes als Parameter auf. Der Inhalt dieser Funktion wird später noch genauer betrachtet.

Datensätze können aber nicht nur hinzugefügt, sondern auch gelöscht werden. Für diesen Fall sind sogar zwei Trigger erforderlich, je einer wird aufgerufen, bevor beziehungsweise nachdem die Datensätze aus der Tabelle gelöscht wurden.

```
1 CREATE TRIGGER nicInfoBeforeDelete BEFORE DELETE ON NicInfo
2 FOR EACH ROW BEGIN
3   DELETE FROM ComputerSystemDuplicateTracking
4   WHERE
5     (identifiedByMacAddress = OLD.macAddress) AND
6     ((computerSystemId1 = OLD.computerSystemId) OR
7      (computerSystemId2 = OLD.computerSystemId)
8     );
9 END
10
11 CREATE TRIGGER nicInfoAfterDelete AFTER DELETE ON NicInfo
12 FOR EACH ROW BEGIN
13   CALL _UpdateDuplicateTracking(OLD.macAddress);
14 END
```

Der erste Trigger löscht die Datensätze aus der Tabelle ComputerSystemDuplicateTracking, welche unter Verwendung der zu löschenden Datensätze aus der Tabelle NicInfo erstellt wurden. Der zweite Trigger sorgt nach Löschung der Datensätze aus der Tabelle NicInfo dafür, dass vorher eventuell zu viel gelöscht Datensätze und dadurch verloren gegangene Beziehungen wiederhergestellt werden.

Die dritte mögliche Operation auf Datensätze in der Tabelle NicInfo ist die Änderung von bestehenden Datensätzen. Auch hierfür werden wieder zwei Trigger erstellt:

```
1 CREATE TRIGGER nicInfoBeforeUpdate BEFORE UPDATE ON NicInfo
2 FOR EACH ROW BEGIN
3   DELETE FROM ComputerSystemDuplicateTracking
4   WHERE
5     (identifiedByMacAddress = OLD.macAddress) AND
6     ((computerSystemId1 = OLD.computerSystemId) OR
7      (computerSystemId2 = OLD.computerSystemId)
8     );
9 END
10
11 CREATE TRIGGER nicInfoAfterUpdate AFTER UPDATE ON NicInfo
12 FOR EACH ROW BEGIN
13   CALL _UpdateDuplicateTracking(OLD.macAddress);
14   CALL _UpdateDuplicateTracking(NEW.macAddress);
15 END
```

Da eine Änderung der Daten eigentlich das gleiche ist, wie eine Löschung der alten und anschließendes Einfügen der neuen Daten, ist es nicht weiter verwunderlich, dass auch die

Trigger eigentlich nur eine Kombination aus den vorher definierten Triggern für Einfüge- und Löschoperationen darstellen.

Alle diese Trigger verwenden die bisher noch nicht gezeigte Funktion *_UpdateDuplicateTracking*. Diese führt die eigentlichen Änderungen an der Tabelle ComputerSystemDuplicateTracking durch und ist wie folgt definiert:

```
1  CREATE PROCEDURE _UpdateDuplicateTracking(IN mac VARBINARY(20))
2  BEGIN
3    -- bisherige Einträge, die diese MAC-Adresse betreffen, aus der
4    -- Tabelle ComputerSystemDuplicateTracking löschen. Sofern diese
5    -- Einträge noch gültig sind, werden sie sofort wieder hinzugefügt.
6    DELETE FROM ComputerSystemDuplicateTracking
7    WHERE (identifiedByMacAddress = mac);
8
9    -- Neue zu kumulierende Datensätze finden und zur Tabelle
10   -- ComputerSystemDuplicateTracking hinzufügen.
11   -- Die beiden inneren SELECT-Abfragen, deren Ergebniss als
12   -- x bzw. y bezeichnet wird, finden alle Computersysteme, die
13   -- diese MAC-Adresse verwenden und stellen sicher, dass nur
14   -- MAC-Adressen zur Kumulation verwendet werden, welche innerhalb
15   -- des jeweiligen Monitoring-Systems eindeutig sind. Die äußere
16   -- SELECT-Abfrage stellt sicher, dass kein Datensatz mit sich
17   -- selbst kumuliert wird.
18   INSERT INTO ComputerSystemDuplicateTracking (computerSystemId1,
19     computerSystemId2, identifiedByMacAddress)
20
21     SELECT x.computerSystemId AS id1, y.computerSystemId AS id2, mac
22     FROM (
23       SELECT computerSystemId FROM (
24         SELECT computerSystemId, COUNT(*) AS cnt
25         FROM NicInfo
26         WHERE (macKind = 'ethernet') AND (macAddress = mac)
27         GROUP BY datasource) AS a
28       WHERE cnt = 1) AS x, (
29       SELECT computerSystemId FROM (
30         SELECT computerSystemId, COUNT(*) AS cnt
31         FROM NicInfo
32         WHERE (macKind = 'ethernet') AND (macAddress = mac)
33         GROUP BY datasource) AS b
34       WHERE cnt = 1) AS y
35     WHERE x.computerSystemId <> y.computerSystemId;
36 END
```

Mit Hilfe dieser einfachen Funktion und weniger Trigger ist es also möglich, die Liste der zu kumulierenden Systeme automatisch und für die einfügenden Abfrageagenten vollständig transparent zu erzeugen und zu pflegen. Diese Liste wird anschließend verwendet, um Views, welche die bereits kumulierten Daten zur Verfügung stellen, zu erzeugen. Als Beispiel wird hier die Definition der View *VMergedComputerSystem*, welche die kumulierten Datensätze der Tabelle ComputerSystem enthält, vorgestellt. Diese View verwendet einige Hilfs-Views, welche hier ebenfalls mit angegeben sind.

```
1 CREATE VIEW VMergedComputerSystem AS
2   SELECT
3     -- mapTo ist die kleinste Id der Datensätze der Tabelle
4     -- ComputerSystem, aus denen dieser kumulierte Datensatz
5     -- besteht. Diese Id wird auch die primäre Id des kumulierten
6     -- Datensatzes genannt
7     mapTo AS primaryComputerSystemId,
8     -- computerSystemIds ist eine Komma getrennte Liste aller
9     -- Datensätze der Tabelle ComputerSystem, aus denen dieser
10    -- kumulierte Datensatz besteht
11    GROUP_CONCAT(idComputerSystem) AS computerSystemIds,
12    -- diese Felder sind je nach ursprünglichem Monitoringsystem
13    -- entweder 0 bzw. '' oder sollten die gleichen, richtigen
14    -- Werte enthalten. MAX nimmt den Wert eines der Systeme,
15    -- bei denen dieser nicht 0 bzw. '' ist
16    MAX(cpuCount) AS cpuCount,
17    MAX(cpuType) AS cpuType,
18    MAX(cpuMhz) AS cpuMhz,
19    MAX(memory) AS memory,
20    MAX(manufacturer) AS manufacturer,
21    MAX(model) AS model,
22    MAX(serialNumber) AS serialNumber,
23    MAX(inventory) AS inventory,
24    MAX(firstPwr) AS firstPwr,
25    MAX(location) AS location,
26    MAX(sysAdmin) AS sysAdmin,
27    MAX(asset) AS asset,
28    MAX(powerState) AS powerState,
29    MAX(commentId) AS commentId,
30    MAX(commentBackupId) AS commentBackupId,
31    -- Beschreibungen werden zusammengefasst, wobei jeweils ein
32    -- Zeilenumbruch zur Trennung verwendet wird
33    GROUP_CONCAT(description SEPARATOR '\n') AS description,
34    -- Zeitstempel und Datenquellen werden wie computerSystemIds
35    -- als Komma getrennte Liste ausgegeben
36    GROUP_CONCAT(timestamp) AS timestamps,
37    GROUP_CONCAT(datasource) AS datasources
38  FROM VInternalGetDuplicateIdMap, ComputerSystem
39  WHERE VInternalGetDuplicateIdMap.id = ComputerSystem.idComputerSystem
40  GROUP BY mapTo;
41
42 -- VInternalGetDuplicateIdMap liefert eine Liste aller Systeme
43 -- der Tabelle ComputerSystem sowie deren primärer Id. Sofern
44 -- ein Datensatz nicht kumuliert wird, da es zu diesem System nur
45 -- einen Datensatz gibt, sind idComputerSystem und die primäre Id
46 -- identisch
47 CREATE VIEW VInternalGetDuplicateIdMap AS
48   SELECT DISTINCT computerSystemId1 AS id, computerSystemId2 AS mapTo
49   FROM ComputerSystemDuplicateTracking
50   WHERE computerSystemId1 > computerSystemId2
51   UNION
52   SELECT idComputerSystem, idComputerSystem
53   FROM VUniqueComputerSystemIds;
```

```
54
55 -- Diese View enthält eine Liste aller Ids von zu kumulierenden
56 -- Datensätzen, die nicht zugleich primäre Ids dieser kumulierten
57 -- Datensätze sind
58 CREATE VIEW VInternalGetDuplicates AS
59   SELECT DISTINCT computerSystemId1 AS computerSystemId
60   FROM ComputerSystemDuplicateTracking
61   WHERE computerSystemId1 > computerSystemId2;
62
63 -- Diese View liefert eine Liste aller Ids von zu kumulierenden
64 -- Datensätzen, die zugleich primäre Ids dieser kumulierten
65 -- Datensätze sind
66 CREATE VIEW VUniqueComputerSystemIds AS
67   SELECT idComputerSystem
68   FROM
69     ComputerSystem LEFT JOIN VInternalGetDuplicates
70     ON (ComputerSystem.idComputerSystem =
71         VInternalGetDuplicates.computerSystemId)
72   WHERE VInternalGetDuplicates.computerSystemId IS NULL;
```

Für jedes Feld der kumulierten Tabelle ist also eine jeweils geeignete Aggregatfunktion zu finden, welche aus den zu kumulierenden Datensätzen den besten Wert auswählt oder die Feldinhalte sinnvoll kombiniert. Das Feld des Primärschlüssels der unkumulierten Daten enthält im kumulierten Datensatz eine Liste aller Datensätze, aus denen dieser kumulierte Datensatz erstellt wurde. Des Weiteren wird die primäre Id des kumulierten Datensatzes als neues Attribut mit angegeben, um eindeutig auf den kumulierten Datensatz Bezug nehmen zu können. Eine derartige View, deren Namen aus VMerged und dem angehängten Namen der ursprünglichen Tabelle besteht, ist für die meisten Tabellen implementiert worden.

4.2. Implementierung der Abfrageagenten

Beide derzeit implementierten Abfrageagenten (für LRZmonitor und VMware vSphere) sind größtenteils in PHP geschrieben und verwenden einen Teil des Programmcodes, insbesondere im Bereich der Datenbankzugriffe gemeinsam. Dadurch ist es möglich diese kritischen Bereiche nur einmal zu entwickeln und zu testen. Allerdings ist der bei weitem größte Teil der Agenten spezifisch für die jeweilige Datenquelle und befasst sich mit dem Parsen, Überprüfen und Konvertieren der Quelldaten in das Format der Datenbank. Auf diese Details der Implementierung soll im Folgenden genauer eingegangen werden.

4.2.1. Implementierung des Abfrageagenten für den LRZmonitor

Der LRZmonitor speichert die erfassten Daten jedes Computers in einer Reihe von Textdateien. Die Dateien aller erfassten Computer werden dabei in einem einzigen Ordner abgelegt. Alle Dateinamen setzten sich aus dem Computernamen, dessen Domänennamen und einer vom jeweiligen Dateiinhalt abhängigen Dateinamenserweiterung zusammen, welche jeweils durch einen Punkt getrennt werden. Der Abfrageagent für die Daten des LRZmonitors kann auf die Datensätze direkt zugreifen. Dies ist möglich, da der gesamte Datenbestand des LRZmonitors automatisch auf den Computer, auf dem der Abfrageagent ausgeführt wird, repliziert wird. Für jeden Rechner muss eine Reihe von Dateien existieren, andere sind op-

tional und existieren nur bei manchen Computern. Für jeden erfassten Computer existiert eine Datei mit der Erweiterung *chk*, in der wichtige Konfigurationen für den LRZmonitor bezüglich des jeweiligen Rechners und manuell eingetragene statische Informationen erfasst sind. Für einen Webserver mit Namen *www* und der Domäne *lrz.de* würde diese Datei also *www.lrz.de.chk* heißen. Diese Datei setzt sich aus einer Reihe von Zeilen zusammen, die alle die Syntax <Schlüssel>=<Wert> haben. Die wichtigsten Schlüssel, Beispiele für die Werte sowie deren Bedeutung sind der Tabelle in Abbildung 4.1 dargestellt.

Schlüssel	**Beispiel**	**Bedeutung**
NODENAME	www.lrz.de	voll qualifizierter Hostname
CLASS	web	Klasse, zu der dieser Rechner gehört
DESCR	Webserver des LRZ	Beschreibung
LOCATION	Raum 1234	Standort
SYSADMIN	Webmaster	Systemadministrator
CHK_RPM	yes	automatisch Updates installieren?
CHK_PERM	no	Dateiberechtigungen prüfen?
CHK_TSM	yes	Backup Status überprüfen?
MACADDR	00:0C:FC:00:1A:FF	MAC-Adresse dieses Systems (Sollzustand)

Abbildung 4.1.: Inhalt der .chk-Dateien des LRZmonitors (Auszug)

Neben dieser Datei existieren eine Reihe weitere Dateien, welche oft nur aus einer einzigen Zeile bestehen, in der die Informationen als kurze Zeichenkette gespeichert sind, zum Beispiel die in Tabelle 4.2 dargestellten.

Erweiterung	**Beispiel**	**Inhalt**
cpu	Intel(R) Xeon(TM) CPU 2.80GHz 2X-SMP @ 2791.308 MHz	Anzahl und Art der Prozessoren
ram	4036628 kB	Größe des verbauten Arbeitsspeichers
ins	10.01.08 16:55:09 CET	Installationszeitpunkt
kbd	DE	Tastaturbelegung
kernel	2.6.32.24-0.2-default	Kernelversion
os	SuSE Linux ES10.3ia64	Installiertes Betriebssystem, Version und Architektur
time	29.11.10 11:49:22	Zeitpunk der letzten Aktualisierung der Daten

Abbildung 4.2.: Liste diverser Dateien des LRZmonitors (Auszug)

Einige Dateien sind auch deutlich komplizierter aufgebaut, so zum Beispiel die Dateien mit den Erweiterungen *dfs* und *net*, welche die Informationen zu den eingebundenen Dateisystemen und deren Belegung beziehungsweise zu allen Netzwerkkarten und deren Konfiguration enthalten. Im Folgenden ist ein Beispiel für den Inhalt einer *net*-Datei für einen Computer mit zwei Netzwerkkarten dargestellt:

```
1 0000:03:07.0|Intel 82546EB Gigabit Ethernet Controller|e1000|00:0E:0C↩
    :23:FA:01|eth0|1000Mb/s,FullDuplex,LinkOK|192.168.15.105/24|www.lrz↩
```

```
    .de|fe80::fe12/64
2 0000:03:07.1|Intel 82546EB Gigabit Ethernet Controller|e1000|00:0E:0C↩
    :23:FA:02|eth1|100Mb/s,FullDuplex,LinkOK|192.168.10.1/24|-|fe80::↩
    fe10/64
```

Diese Datei enthält also eine Zeile für jede Netzwerkkarte. Jede Zeile besteht aus einer Reihe von Feldern, welche durch einen senkrechten Strich getrennt sind. Außerdem existieren noch Dateien, deren Inhalt nicht automatisch ausgewertet werden muss, zum Beispiel enthalten Dateien mit der Endung *rpm* eine Liste aller auf dem Computer installierter Softwarepackete und Dateien mit der Endung *mail* enthalten die letzte durch den LRZmonitor an den Administrator verschickte Warn-eMail. Abschießend existiert eine Reihe von Dateien, deren Erweiterung auf *led* endet, zum Beispiel *dfsled*, *perled* oder *rpmled*. Diese Dateien sind alle gleich aufgebaut und enthalten einen Statuswert einer bestimmten durch den LRZmonitor durchgeführten Prüfung. Sie bestehen alle nur aus einer Zeile, die nur ein Wort enthält. Mögliche Werte sind zum Beispiel *red* für einen Fehler, *green* für einen erfolgreichen Test oder *testgreen* für einen erfolgreichen Test im Simulationsmodus. Die möglichen Werte und deren Bedeutung sind dabei abhängig von der jeweiligen Prüfung. Die Datei *dfsled* enthält das Ergebnis der Prüfung der Dateisystemkontingente, *perled* das Ergebnis der Prüfung der Dateiberechtigungen und *rpmled* das Ergebnis des letzten automatischen Systemupdates.

Zum Einfügen dieser Datensätze in die Datenbank wird zuerst eine Liste aller Computernamen, für die Daten vorliegen, erstellt. Anschließend wird über diese Liste iteriert und für jeden Computer werden alle verfügbaren Daten ermittelt. Dies geschieht in einer eigenen Funktion, welche die Daten in Form einer einzigen großen Datenstruktur an das Hauptprogramm zurück liefert. Die Funktion zum Lesen der Daten ist sehr umfangreich, was primär darin begründet ist, dass die Daten in vielen unterschiedlichen Formaten vorliegen, welche alle unterstützt werden müssen. So kann alleine die Angabe zum Zeitpunkt des letzten Systemneustarts in vier verschiedenen Formaten angegeben werden, die in der folgenden Tabelle zusammengefasst sind.

06.12.10 11:39:17 CET, 11 days 9:42	Startzeitpunkt und Laufzeit
56 days 0:07	nur Laufzeit in Tagen, Minuten und Sekunden
1 day, 5:36	nur Laufzeit mit einem zusätzlichen Komma
0:01	nur Laufzeit in Minuten und Sekunden

Diese vielen möglichen Darstellungsformen lassen sich primär dadurch erklären, dass der LRZmonitor oft direkt die Ausgabe von Linux Kommandozeilenwerkzeugen, wie zum Beispiel dem Befehl `uptime`, übernimmt. Wie bei allen unixoiden Systemen sind aber bisher alle Versuche, die Ausgabe oder Parameter dieser Standardwerkzeuge zu vereinheitlichen, gescheitert und die Unterschiede sich je nach Hersteller und Version zum Teil recht deutlich. Daraus folgt, dass der Programmcode des Abfrageagenten, der diese Daten lesen und interpretieren muss, sehr umfangreich wird, da die jeweils verwendete Notation erkannt und behandelt werden muss. Außerdem werden die Daten beim Lesen einer genauen Konsistenzprüfung unterzogen um mögliche Inkonsistenzen zu erkennen und zu korrigieren bevor diese Daten in die Datenbank gelangen können. Dabei wird bei allen Daten geprüft, ob diese sich in einer der unterstützten Darstellungsformen befinden, sich sofern anwendbar in einem jeweils sinnvollen Wertebereich befinden, Datumsfelder ein gültiges Datum enthalten, ob zusammengehörige Daten zueinander Konsistent sind und ob zwingend erforderliche Daten auch wirklich vorhanden sind. Bei Fehlern wird eine Warnmeldung an den Administrator geschickt und die Daten wenn möglich korrigiert oder durch Standardwerte ersetzt. Im folgenden Quelltext-Beispiel ist die Prüfung der Arbeitsspeichergröße sowie des Installations-

zeitpunkts dargestellt:

```
// Prüfung der Arbeitsspeichergröße
// Format ist: "XXX kB"
// Beispiel: "512412 kB"
$s = substr($ram, 0, strpos($ram, ' '));
if ( (! is_numeric($s)) ||
     (substr($ram, strpos($ram, ' ') + 1) != 'kB') ) {
  echo('memory info is inconsistant and might be wrong: ' . $ram);
  if (! is_numeric($s)) { $s = 0; }
}
ram = intval($s) * 1024;

// Prüfung des Installationszeitpunkts
// Format ist: "dd.mm.yy HH.MM.SS Zeitzone"
// Beispiel: "27.02.08 15:54:18 CET"
$ins = date_create_from_format('d.m.y H:i:s T', $installTime);
if ($ins) {
  $ins = date_format($s, 'U');
  $installTime = intval($ins);
} else {
  echo('could not parse timestamp: ' . $installTime);
  $installTime = 0;
}
```

Nach dem Einlesen der Daten können diese in die Datenbank eingefügt werden. Sofern in der Datenbank bereits ein Datensatz für diesen Computer existiert wird dieser gelöscht. Anschließend werden die ermittelten Daten als neuer Datensatz in die Datenbank eingefügt. Löschen und Einfügen werden dabei zu einer einzigen atomaren Transaktion zusammengefasst, sodass zu jedem Zeitpunk einer der beiden Datensätze in der Datenbank enthalten ist und im Falle eines Fehlers der alte Datensatz vollständig wiederhergestellt werden kann. Der grobe Programmablauf ist in Abbildung 4.3 dargestellt. Der Programmcode fügt also einen Computer nach dem anderen in die Datenbank ein beziehungsweise aktualisiert die Daten. Dies bietet den Vorteil, dass der Abfrageagent später dahingehend erweitert werden kann, dass er nicht mehr in regelmäßigen Zeitabständen ausgeführt wird und dabei alle Daten verarbeitet, sondern direkt durch einen der Computer, welche Daten an das System liefern, aufgerufen wird und die Daten nur dieses Computers verarbeitet. Hierdurch entfallen unnötige Verzögerungen beim Einfügen der Daten und langfristig könnten die vielen einzelnen Dateien, welche zentral gespeichert werden müssen, sogar komplett entfallen. Für diese Änderungen muss der Abfrageagent nur mehr leicht verändert werden, da der gesamte Code zum Lesen der Daten und Einfügen in die Datenbank nicht mehr modifiziert werden muss, sondern nur die äußere Schleife entfällt und durch Code zur Übergabe der Daten vom zu erfassenden Computersystem ersetzt wird. Leider war es bisher noch nicht möglich diese Änderungen durchzuführen, da hierzu auch größere Änderungen am LRZmonitor erforderlich sind, deren Entwicklung und Test auf allen betroffenen Systemen einige Zeit benötigt.

4.2.2. Implementierung des Abfrageagenten für VMware vSphere

Beim VMware vSphere Abfrageagenten ist der Ablauf deutlich anders. Hier werden zuerst alle Daten von einem Microsoft Windows PowerShell Skript vom VMware Server abgefragt und in eine Reihe temporärer Dateien gespeichert. Die PowerShell arbeitet dabei durchge-

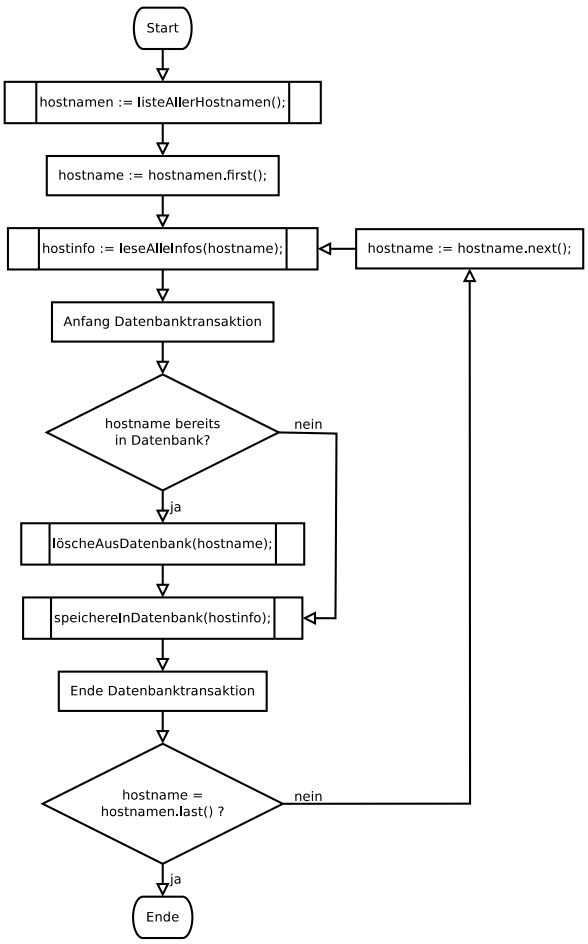


Abbildung 4.3.: Ablaufplan des Abfrageagenten für LRZmonitor-Datensätze

hend objektorientiert und VMware stellt die Daten in einer komplexen Objekthierarchie zur Verfügung. Die Abfrage der Daten erfolgt, indem benötigten Daten zu einem neuen Objekt zusammengestellt werden und dieses Objekt dann mittels des `Export-CSV` Befehls in eine Textdatei geschrieben wird. Innerhalb des PowerShell Skripts werden dabei nur sehr rudimentäre Konsistenzprüfungen durchgeführt, die meisten Prüfungen finden erst bei der Weiterverarbeitung der Daten durch ein PHP-Skript statt. Das folgende Quelltextfragment zeigt die Abfrage aller virtuellen Maschinen aus VMware und den Export der benötigten Daten in eine Textdatei.

```
1 ## Dieser Funktion wird ein Objekte übergeben, welches eine
2 ## virtuelle Maschine repräsentieren. Die Funktion extrahiert
3 ## daraus die benötigten Daten und gibt diese in Form eines
4 ## neuen Objekts zurück.
5 filter Extract-VMData() {
6     ## Erstellt ein neues Objekt für die zurückzuliefernden Daten
7     $result = "" | Select-Object ↩
          Id,Name,Description,NumCPU,MemoryMB,BootTime
8
9     ## $_ ist das übergebene Objekt, daraus die benötigten
```

```
10      ## Daten kopieren
11      $result.Id = $_.Id
12      $result.Name = $_.Name
13      $result.Description = $_.Description
14      $result.NumCPU = $_.NumCPU
15      $result.MemoryMB = $_.MemoryMB
16
17      ## Der Boot-Zeitpunkt ist in einer eigenen Unteransicht versteckt,
18      ## diese Ansicht erstellen
19      $Vm = Get-View $_
20
21      ## Boot-Zeitpunkt ermitteln
22      $time = $Vm.Summary.Runtime.BootTime
23
24      ## Der Zeitstempel ist nicht immer auf einen gültigen Wert
25      ## gesetzt, weshalb dies hier überprüft wird
26      if ( $time ) {
27          ## Boot-Zeitpunkt ist gültig, in UTC umrechnen und
28          ## für die Datenbank formatieren
29          $result.BootTime = $time.ToUniversalTime() | Get-Date -UFormat ↩
                "%Y-%m-%d␣%H:%M:%S"
30      } else {
31          ## Boot-Zeitpunkt ist ungültig, auf null setzen
32          $result.BootTime = "0000-00-00␣00:00:00"
33      }
34
35      ## Ergebnis zurückliefern
36      return $result
37  }
38
39
40  ##### Anfang des Hauptprogramms
41
42  ## Verbindung zum VIServer herstellen
43  $VIServer = Connect-VIServer -Server 'vmware-cluster.lrz.de' -User ↩
        'administrator' -Password 'geheim'
44
45  ## alle virtuellen Maschinen ermitteln, benötigte Daten extrahieren
46  ## und in eine csv-Datei speichern. Get-VM liefert nicht nur eine,
47  ## sondern alle virtuellen Maschinen auf einmal. PowerShell ruft
48  ## anschließend die Funktion Extract-VMData für jedes Objekt einmal
49  ## auf und sammelt die zurückgegeben Objekte. Nun wird Export-CSV
50  ## mit allen Objekten auf einmal aufgerufen und speichert diese
51  ## in eine einzige Datei.
52  Get-VM | Extract-VMData | Export-CSV -Delimiter ";" -NoTypeInformation ↩
        -Encoding UTF8 -Path vm.csv
53
54  ## Verbindung zum VIServer trennen
55  Disconnect-VIServer -Server $VIServer -Confirm:$false
```

Die erstellte Textdatei enthält alle ermittelten Werte getrennt durch Strichpunkte und sieht zum Beispiel wie folgt aus:

```
1  "Id";"Name";"Description";"NumCPU";"MemoryMB";"BootTime"
```

```
2 "vm-4851";"www";"Webserver";"2";"1024";"2010-09-12 12:26:01"
3 "vm-8649";"mail";"Mailserver";"1";"512";"2011-01-29 04:35:28"
4 "vm-9367";"dns";"DNS Server";"1";"1024";"2010-12-30 22:01:56"
5 "vm-9372";"ldap";"LDAP Server";"4";"4096";"2011-03-03 00:38:43"
```

Anschließend werden diese Dateien zur besseren Weiterverarbeitung in eine temporäre Datenbank geladen (diese Datenbank hat mit der endgültigen Datenbank zur Speicherung der Daten nichts zu tun, verwendet nicht das gleiche Schema und dient nur der besseren Weiterverarbeitung der Daten). Die Daten werden nun durch ein PHP Skript einer Reihe von Konsistenzprüfungen unterworfen. Während syntaktische Inkonsistenzen der Daten bei der Abfrage aus VMware eher selten sind, da die Daten ja aus einem System stammen, welches bereits Konsistenzprüfungen durchführt und ein definiertes Datenformat für die einzelnen Felder verwendet, können sich trotzdem Inkonsistenzen an den Daten ergeben, wenn sich die Daten während der Abfrage ändern. Da es keine Möglichkeit gibt einen konsistenten Schnappschuss aller Daten von VMware abzufragen und die Abfrage aller Daten einige Zeit benötigt kann es zu Race-Conditions kommen, welche zu Inkonsistenzen in den erfassten Daten führen. Wird zum Beispiel eine neue virtuelle Maschine erstellt nachdem die Liste aller virtuellen Maschinen, jedoch bevor die Liste aller virtuellen Laufwerke abgefragt wurde, führt dies dazu, dass die Liste virtueller Laufwerke Datensätze der neu erstellten Maschine enthält, welche sich keiner virtuellen Maschine zuordnen lassen. Derartige Inkonsistenzen sollten aber nicht in die Datenbank übernommen werden, weshalb sie vor der weiteren Verarbeitung der Daten gesucht und behoben werden müssen.

Dabei ist es sehr hilfreich, dass sich alle Daten von VMware als große Baumstruktur betrachten lassen. Die Wurzel bildet dabei ein Datencenter, dieses enthält Hostsysteme, denen untergeordnet sind die virtuellen Maschinen und unterhalb der virtuellen Maschinen können die darin enthaltenen virtuellen Laufwerke und Netzwerkkarten platziert werden. Jedes Objekt in VMware hat eine feste Position in dieser Baumstruktur und mit Hilfe dieser Struktur können viele Konsistenzprüfungen durchgeführt werden. Eine wichtige Prüfung ist zum Beispiel, ob wirklich alle Daten in der temporären Datenbank sich in einer einzigen Baumstruktur befinden, ob also nur eine Wurzel existiert und keine Knoten vom Baum getrennt sind. Diese Prüfung verhindert Fehler, wie sie durch die oben angesprochenen Race-Conditions entstehen können. Eine weitere Funktion prüft, ob es sich bei dem Graphen tatsächlich um einen Baum handelt, ob der Graph also zyklenfrei ist. Zyklen könnten ebenfalls durch Race-Conditions entstehen, wenn sich die Position eines Elements während der Abfrage ändert und würden bei der späteren Arbeit zu Fehlern und nicht terminierenden Funktionen führen. Abschließend werden noch einige semantische Prüfungen der Daten durchgeführt, zum Beispiel ob deren Werte sich in sinnvollen Grenzen bewegen.

Nach Abschluss der Konsistenzprüfung werden alle Daten in mehrere große Datenstrukturen im Arbeitsspeicher geladen. Dies ist möglich, da die zu erwartende Gesamtdatenmenge nur wenige Megabytes groß ist. Anschließend werden diese Daten in die Datenbank eingefügt. Im Gegensatz zur Vorgehensweise beim Agenten des LRZmonitors wird aber nicht jeder Datensatz einzeln eingefügt, sondern alle Datensätze werden in nur einer einzige großen Transaktion in die Datenbank geladen. Dadurch ist es einerseits möglich, die Leistung deutlich zu verbessern, andererseits entspricht diese Vorgehensweise viel eher der Repräsentation der Daten in den Speicherstrukturen. Vor dem Einfügen der Daten in die Datenbank werden alle alten Datensätze, die von diesem Abfrageagenten stammen, aus der Datenbank entfernt. Dies ist einfacher zu implementieren als für jeden Datensatz zu prüfen, ob dieser eingefügt, aktualisiert oder gelöscht werden muss.

4.3. Implementierung eines prototypischen Webinterfaces zum Anzeigen der Daten

Als kleine Demonstration der Möglichkeiten der zentralen Datenbank wurde ebenfalls ein Prototyp zur Abfrage und Darstellung der erfassten Daten in einer Weboberfläche erstellt. Dieser dient der Demonstration einiger der Möglichkeiten des neuen Systems und insbesondere der automatischen Kumulation der Datensätze, ist aber nicht als vollwertiges Anzeigesystem zu verstehen. Leider war es aufgrund der beschränkten Zeit nicht möglich eine vollständige Weboberfläche zu entwickeln, so dass viele Daten noch nicht dargestellt werden können. Der Prototyp basiert auf der Weboberfläche des LRZmonitors, welche von Winfried Raab entwickelt und zur Verfügung gestellt wurde. Die Abbildungen 4.4, 4.5 und 4.6 zeigen Screenshots der Übersichtslisten über alle Systeme mit jeweils unterschiedlichen angezeigten Informationen. Diese Liste lässt sich durch den Benutzer beliebig sortieren, außerdem können Filter auf die angezeigten Daten angewendet werden um nur relevante Systeme anzuzeigen. Der Screenshot in Abbildung 4.7 zeigt die Datailansicht eines Computersystems, in der alle

Abbildung 4.4.: Screenshot der Übersichtsliste aller Systeme (mit Filter)

zu diesem System bekannten Daten eingesehen werden können. Bei der Entwicklung des Prototypen wurde auf Modularisierung und Abstraktion großer Wert gelegt, sodass es einfach möglich ist, neue Ansichten zu erstellen oder den bestehenden Ansichten neue Spalten hinzuzufügen. Außerdem bleibt der Programmquelltext auf diese Weise übersichtlicher.

Der Quelltext, um eine Tabelle mit einer Übersicht über alle Systeme, deren Hostnamen, Domänennamen, Beschreibung, Aufstellungsort und Inventarnummer zu erstellen, umfasst dabei nur wenige Zeilen und sieht wie folgt aus:

```
1 <?php
2 // Infos über alle Computersysteme aus der Datenbank abfragen
```

Abbildung 4.5.: Screenshot der Übersichtsliste aller Systeme (Status des LRZmonitors)

```
3  $computerSystems = getAllHostOverview();
4
5  // Tabelle erstellen
6  $table = new Table();
7
8  // Einstellungen für das Aussehen der Tabelle
9  $table->addGlobalAttrib('table', 'border', 1);
10
11 // Tabellenspalten hinzufügen
12 // Jeder Aufruf fügt eine Spalte zur Tabelle hinzu,
13 // Parameter sind die anzuzeigenden Daten, die Spaltenüberschrift
14 // und der Name dieser Spalte in der Datenbank
15 $table->addCol($computerSystems, 'Hostname', 'hostname');
16 $table->addCol($computerSystems, 'Domänenname', 'domainname');
17 $table->addCol($computerSystems, 'Beschreibung', 'desription');
18 $table->addCol($computerSystems, 'Aufstellungsort', 'location');
19 $table->addCol($computerSystems, 'Inventarnummer', 'inventory');
20
21 // Die Inventarnummern rechtsbündig ausrichten
22 $table->addColAttrib(5, 'align', 'right');
23
24 // Die fertige Tabelle ausgeben und anzeigen
25 $table->draw();
26 ?>
```

Die Klasse *Table* wurde für die Darstellung der Daten im LRZmonitor erstellt und unterstützt neben den gezeigten noch eine Reihe weiterer Funktionen, welche im Prototypen genutzt, hier aber nicht genauer beschrieben werden können.

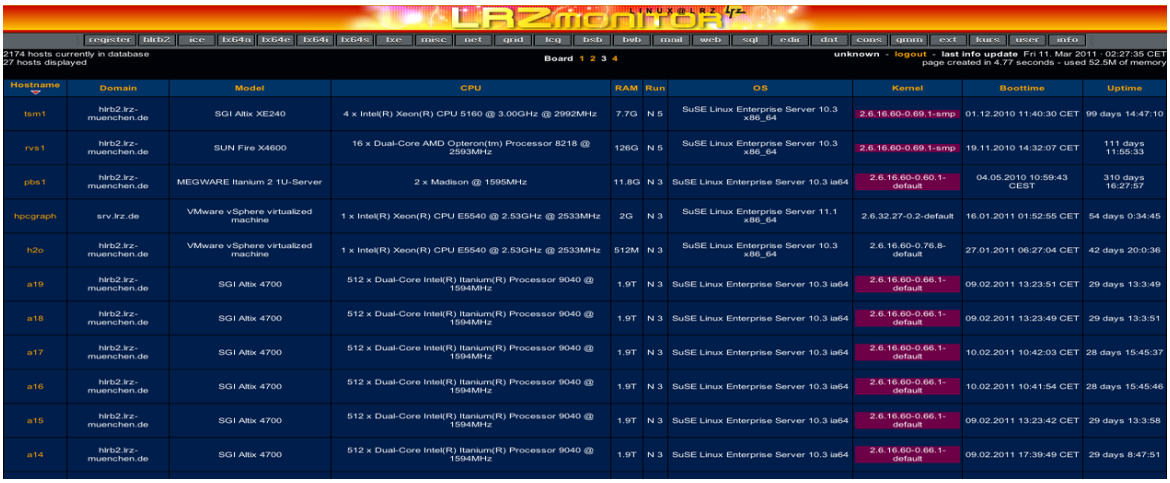


Abbildung 4.6.: Screenshot der Übersichtsliste aller Systeme (Rechnerkonfiguration)

Abbildung 4.7.: Screenshot der Detailansicht eines Computersystems

5. Ergebnisse, Erweiterungsmöglichkeiten und Ausblick

Durch diese Arbeit wurde die Grundlage für ein großes, zentrales Monitoringsystem am LRZ geschaffen. Es wurden ein Datenmodell und ein Datenbankschema entworfen und implementiert um die wichtigsten Daten von zwei der bisher eingesetzten Monitoringsysteme zu sammeln und zentral zu speichern. Hierbei wurde Wert darauf gelegt, dass eine zukünftige Erweiterung einfach möglich ist um die Daten von weiteren Systemen ebenfalls in diese Datenbank einfließen zu lassen und damit den Nutzen noch zu vergrößern. Ebenfalls wurde ein grundlegendes Schema definiert, nach welchem die Datenübernahme aus den bestehenden Systemen ablaufen soll. Die vollständige Übername der Daten des LRZmonitors sowie die Übername ausgewählter Informationen aus dem VMware Monitoring System wurde implementiert und ausführlich getestet und kann nun in der Praxis eingesetzt werden. Dabei wurde großer Wert auf die Prüfung der Daten gelegt um nur konsistente Datensätze in der Datenbank zu speichern und damit die Qualität der gespeicherten Informationen zu garantieren. Des Weiteren wurde eine automatische Kombination der durch die unterschiedlichen Monitoringsysteme erfassten Informationen implementiert, welche den auswertenden Systemen alle zu einem System vorliegenden Daten einheitlich zur Verfügung stellt und damit die Implementierung von Auswerte- und Anzeigesystemen vereinfacht. Als Beispiel für ein solches System und die sich darin ergebenden Möglichkeiten wurde ein Prototyp einer Weboberfläche zur Visualisierung der Daten auf Basis des bisher durch den LRZmonitor genutzten Systems geschaffen, in welchem der Funktionsumfang und Bedienungskomfort bereits jetzt zum Teil deutlich über dem des bisherigen Systems liegt (so werden die zusätzlich aus dem VMware Monitoring System gewonnenen Informationen angezeigt und die Filterung und Sortierung von Daten ermöglicht). Durch die Erstellung dieses Prototypen wurde ein Rahmen geschaffen, innerhalb dessen leicht neue Funktionen hinzugefügt werden können.

Das bisher geschaffene System stellt eine Grundlage dar, bietet aber noch viele Erweiterungsmöglichkeiten. So wäre es sehr erstrebenswert, weitere Monitoringsysteme, wie zum Beispiel das Monitoring der Windows basierten Rechner oder der Netzwerkkomponenten, mit in das neue Monitoringsystem zu integrieren. Auch konnten noch keine weitergehenden Auswertesysteme, wie zum Beispiel die automatische Suche nach Konfigurationsfehlern auf Basis der gespeicherten Informationen, erstellt werden. Erste Tests während der Implementierung zeigen jedoch, dass es prinzipiell möglich ist, einige Fehler, zum Beispiel bezüglich der Netzwerkkonfiguration, automatisch zu finden und den zuständigen Administrator darüber zu informieren. Fehler, welche zu inkonsistenten Datensätzen führen würden, werden auch bereits beim Einfügen dieser Datensätze erkannt und müssten nur noch geeignet weiterverarbeitet werden.

Ebenfalls sehr interessant und ein lohnenswertes Thema für zukünftige Arbeiten könnte es sein, auch die Dienste und deren Abhängigkeiten von einander mit in die Datenbank zu übernehmen um die Auswirkungen des Ausfalls eines dieser Dienste besser vorhersagen zu können. Hierfür könnte die Datenbank leicht erweitert werden, allerdings gibt es derzeit am

LRZ noch kein System, welches diese Informationen automatisch erstellen könnte, so dass es mit einer einfachen Übernahme der Daten in die Datenbank in diesem Fall leider nicht getan ist. Auch logistische Aspekte, welche bisher noch kaum behandelt werden, sollten in Zukunft genauer betrachtet werden. Dazu zählen zum Beispiel die Kapazitätsplanung der Computersysteme und Netzwerke, die Erkennung und Vorhersage von Kapazitätsengpässen sowie der Abgleich des Ist-Zustands aus dieser Datenbank mit dem Soll-Zustand aus einer externen CMDB und die Erkennung von relevanten Abweichungen.

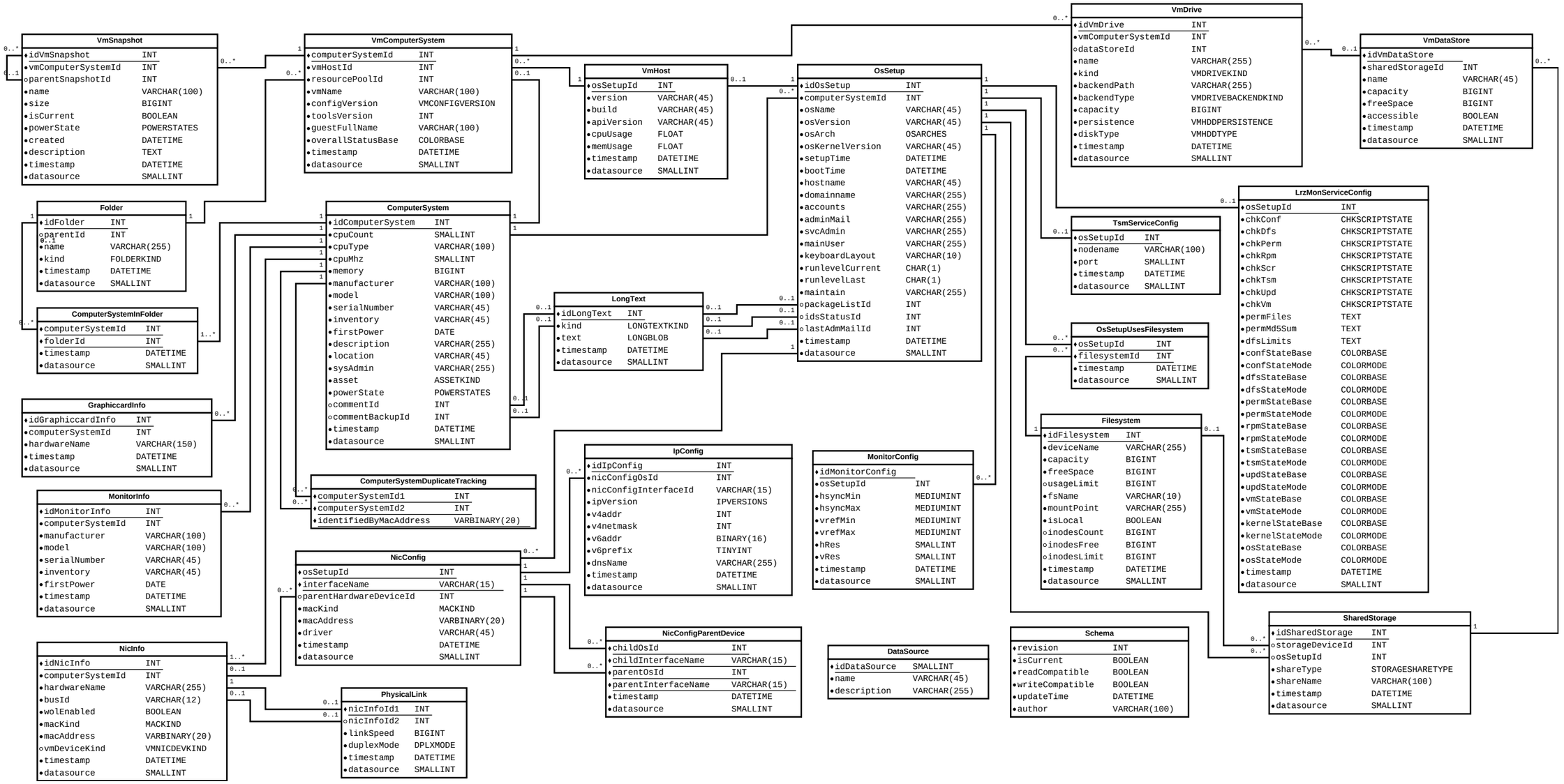
Ein weiterer wichtiger Aspekt für zukünftige Arbeiten liegt in der automatischen Erkennung von Fehlern, welche die gleiche Ursache haben. So kann zum Beispiel der Ausfall eines Netzwerkspeichers sich auf eine große Anzahl von darauf gespeicherten virtuellen Maschinen auswirken. In diesem Fall wäre es für die Administratoren sehr hilfreich, wenn die Fehlermeldungen automatisch zusammengefasst würden und nur die grundlegende Ursache, nämlich der Ausfall eines Servers, und die daraus resultierenden sekundären Ausfälle und ihre Auswirkungen auf das Netzwerk, zum Beispiel in Form von nicht mehr verfügbaren Diensten, übersichtlich angezeigt würden. Es wäre auch denkbar, alle bereits einmal aufgetretenen Fehler, deren Symptome, diagnostizierte Ursachen und unternommene Schritte zur Fehlerbehebung zu erfassen. Dadurch könnte ein Expertensystem beim wiederholten auftreten eines bereits früher einmal behobenen Fehlers die Ursache automatisch erkennen und Hinweise zur Fehlerbehebung geben.

Insgesamt lässt sich sagen, dass während dieser Arbeit ein funktionierender Prototyp eines zentralen Monitoringsystems geschaffen wurde, welcher zwar noch vergleichsweise klein und in seiner Funktion beschränkt sein mag, jedoch bereits jetzt einen Ausblick, auf die zukünftigen Möglichkeiten und Anwendungsgebiete bietet, die sich durch die Anbindung weiterer Datenquellen und Entwicklung besserer Auswertesysteme eröffnen können. Durch Erweiterung dieses Systems um neue Datenquellen und Auswertesysteme könnte es möglich werden, eine Vielzahl von oft schwer zu diagnostizierenden Fehlern, zum Beispiel aufgrund von falschen Konfigurationen, zu finden und zu beheben noch bevor diese sich tatsächlich auf das Netzwerk auswirken. Des Weiteren kann die Wartung von Systemen vereinfacht werden, da abhängige Systeme, die von einem Ausfall ebenfalls betroffen sind, jederzeit schnell aufgelistet werden können. Aus diesem Grund wäre es auf jeden Fall sinnvoll, in Zukunft weiter an diesem System zu arbeiten und die bereits erwähnten möglichen Erweiterungen voll umzusetzen.

A. Datenmodelle und Diagramme

FOLDERKIND	ENUM('simpleFolder', 'resourcePool', 'datacenter', 'cluster')
COLORBASE	ENUM('unknown', 'red', 'orange', 'yellow', 'green', 'blue', 'gray')
COLORMODE	ENUM('', 'test', 'off', 'offtest', 'old', 'oldtest')
IPVERSIONS	ENUM('IPv4', 'IPv6')
DPLXMODE	ENUM('unknown', 'half', 'full')
LONGTEXTKIND	ENUM('idsStatus', 'admMail', 'packageList', 'comment', 'commentBackup')
CHKSCRIPTSTATE	ENUM('enabled', 'disabled', 'testing')
OSARCHES	ENUM('unknown', 'x86', 'x86_64', 'ia64', 'mips', 'alpha', 'arm', 'powerpc', 'sparc')
ASSETKIND	ENUM('unknown', 'intern', 'attendedHousing', 'unattended-Housing', 'attendedHosting', 'unattendedHosting')
POWERSTATES	ENUM('unknown', 'off', 'on', 'suspended')
VMDRIVEBACKKIND	ENUM('none', 'imageFile', 'hostDevice', 'remoteDevice')
VMDRIVEKIND	ENUM('harddisk', 'cdrom', 'floppy')
VMHDDPERSISTENCE	ENUM('unknown', 'persistent', 'nonPersistent', 'undoable', 'independentPersistent', 'independentNonPersistent')
VMHDDTYPE	ENUM('unknown', 'rawVirtual', 'rawPhysical', 'flat')
VMNICDEVKIND	ENUM('unknown', 'e1000', 'flexible', 'vmxnet', 'enhancedv-mxnet', 'vmxnet3')
STORAGESHARETYPE	ENUM('nfs', 'vmfs', 'cifs', 'iscsi')
VMCONFIGVERSION	ENUM('unknown', 'v4', 'v7')
MACKIND	ENUM('unknown', 'ethernet', 'infiniband', 'irda')

Abbildung A.1.: Benutzerdefinierte Datentypen des Datenbankschemas



(angegebene Datentypen sind entweder MySQL Standardtypen oder der Tabellen in Abbildung A.1 zu entnehmen)
(alle Beziehungen zwischen den datasource-Attributen und der DataSource-Tabelle wurden zur besseren Übersichtlichkeit nicht eingezeichnet)

Abbildung A.2.: relationales Datenschema

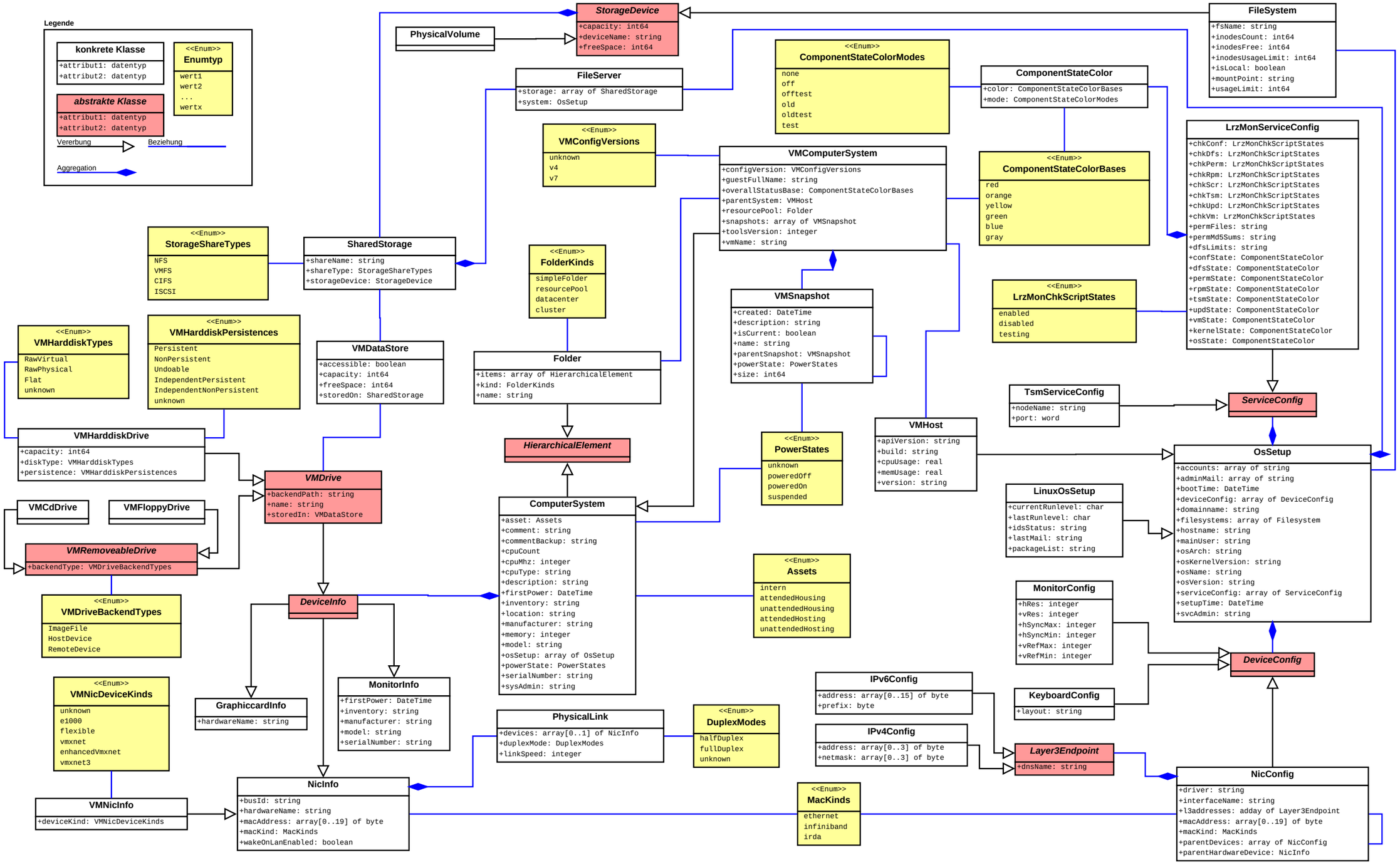


Abbildung A.3.: UML-Diagramm des verwendeten Datenschemas

B. Tabellendefinitionen und Beispieldatensätze

Im Folgenden sind alle Tabellen der Datenbank, deren Verwendungszweck, die Attribute samt Datentypen, Beispieldatensätze sowie die genaue Bedeutung der einzelnen Attribute aufgelistet. Primärschlüssel sind unterstrichen, Fremdschlüssel kursiv gesetzt. Die angegebenen Datentypen sind entweder grundlegende Typen des MySQL Datenbankverwaltungssystems oder in der Tabelle A.1 auf Seite 45 aufgelistet. Die Spalte Null gibt an, ob für dieses Attribut Null-Werte zulässig sind. Enthält die Spalte Beispiel einen Wert in spitzen Klammern so handelt es sich um einen Verweis auf einen Datensatz der in den Klammern angegebenen Tabelle, <ComputerSystem> stellt also zum Beispiel einen Verweis auf einen Datensatz der Tabelle ComputerSystem dar.

B.1. ComputerSystemDuplicateTracking

Diese Tabelle wird intern zur Nachverfolgung von Datensätzen verwendet, welche das gleiche System beschreiben. Siehe hierzu auch Kapitel 3.4.

Feld	Typ	Null	Beispiel	Beschreibung
computerSystemId1	INT	Nein	<ComputerSystem>	Verweis auf ein Computersystem
computerSystemId2	INT	Nein	<ComputerSystem>	Verweis auf dazugehöriges Computersystem
identifiedByMacAddress	VARBINARY(20)	Nein	005056AABB00	MAC-Adresse, anhand derer das Duplikat erkannt wurde

B.2. ComputerSystemInFolder

Diese Tabelle realisiert die n:m-Beziehung zwischen Computersystemen und Ordnern. Dadurch ist es möglich, dass ein Computersystem in mehreren Ordnern aufgelistet wird.

Feld	Typ	Null	Beispiel	Beschreibung
computerSystemId	INT	Nein	<ComputerSystem>	Verweis auf Computersystem
folderId	INT	Nein	<Folder>	Verweis auf Ordner
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.3. ComputerSystem

Die Tabelle ComputerSystem speichert Informationen zur Hardware von Systemen. Jeder Eintrag steht für ein physisches (oder virtualisiertes) Computersystem. Auch andere Komponenten, wie zum Beispiel Netzwerkswitches, können hier erfasst werden.

Feld	Typ	Null	Beispiel	Beschreibung
idComputerSystem	INT	Nein	42	eindeutiger Schlüssel
cpuCount	SMALLINT	Nein	1	Anzahl der Prozessoren
cpuType	VARCHAR(100)	Nein	Intel(R) Xeon(R) CPU E5540	Art des Prozessors
cpuMhz	SMALLINT	Nein	2533	max. Prozessortakt in MHz
memory	BIGINT	Nein	515063808	Arbeitsspeicher in Bytes
manufacturer	VARCHAR(100)	Nein	HP	Systemhersteller
model	VARCHAR(100)	Nein	ProLiant BL490c G6	Modellbezeichnung
serialNumber	VARCHAR(45)	Nein	TES07401	Seriennummer
inventory	VARCHAR(45)	Nein	1.234	Inventarnummer
firstPower	DATE	Nein	01.11.2010	Datum der Inbetriebnahme
description	VARCHAR(255)	Nein	DNS Nameserver	Beschreibung, Funktion
location	VARCHAR(45)	Nein	NSR/4J/13-14	Aufstellungsort, Raumnummer, etc.
sysAdmin	VARCHAR(255)	Nein	Grid-Admin	Systemadministrator oder Gruppe
asset	ASSETKIND	Nein	unattendedHosting	Art der Hosting/Housing Vereinbarung
powerState	POWERSTATES	Nein	on	Betriebszustand
commentId	INT	Ja	<LongText>	Verweis auf Kommentarfeld
commentBackupId	INT	Ja	<LongText>	Verweis auf Backup Kommentarfeld
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.4. GraphiccardInfo

Diese Tabelle enthält Informationen zu allen verbauten Graphikkarten.

Feld	Typ	Null	Beispiel	Beschreibung
idGraphiccardInfo	INT	Nein	42	eindeutiger Schlüssel
computerSystemId	INT	Nein	<ComputerSystem>	Computer, der dieses Gerät enthält
hardwareName	VARCHAR(150)	Nein	ATI Technologies Inc ES1000	Hardwarebezeichnung
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.5. Filesystem

Diese Tabelle enthält Informationen zu allen Dateisystemen. Dazu zählen die Art des Dateisystems, Kapazität, Belegung und der Einhängepunkt beziehungsweise Laufwerksbuchstabe.

Feld	Typ	Null	Beispiel	Beschreibung
idFilesystem	INT	Nein	42	eindeutiger Schlüssel
deviceName	VARCHAR(255)	Nein	/dev/sda3	Gerätename (abhängig vom Betriebssystem)
capacity	BIGINT	Nein	8584888320	Speicherkapazität in Bytes
freeSpace	BIGINT	Nein	4994936832	freier Speicher in Bytes
usageLimit	BIGINT	Ja	8499039437	Speicherplatzlimit in Bytes, Warnung bei Überschreitung
fsName	VARCHAR(10)	Nein	ext4	Name des Dateisystems
mountPoint	VARCHAR(255)	Nein	/home	Einhängepunkt oder Laufwerksbuchstabe
isLocal	BOOLEAN	Nein	1	1 für lokalen Speicher, 0 für Netzwerkspeicher
inodesCount	BIGINT	Ja	8393920	Anzahl der Inodes, sofern diese nicht dynamisch vergeben werden
inodesFree	BIGINT	Ja	8217201	Anzahl der freien Inodes
inodesLimit	BIGINT	Ja	8309981	Warnstufe für Inodes, Warnung bei Überschreitung
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.6. IpConfig

Diese Tabelle speichert die für jeden Rechner vergebenen IP-Adressen. Jeder Rechner kann beliebig viele IPv4 und IPv6 Adressen besitzen.

Feld	Typ	Null	Beispiel	Beschreibung
idIpConfig	INT	Nein	42	eindeutiger Schlüssel
nicConfigOsId	INT	Nein	<NicConfig>	Verweis auf zugehöriges Netzwerkinterface
nicConfigInterfaceId	VARCHAR(15)	Nein	<NicConfig>	Verweis auf zugehöriges Netzwerkinterface
ipVersion	IPVERSIONS	Nein	IPv4	IP Version (IPv4 oder IPv6)
v4addr	INT	Nein	3232235521	IPv4 Adresse als 32 Bit Integer
v4netmask	INT	Nein	4294967040	IPv4 Subnetmaske als 32 Bit Integer
v6addr	BIGINT	Nein	0	IPv6 Adresse
v6prefix	TINYINT	Nein	64	IPv6 Prefixlänge
dnsName	VARCHAR(255)	Nein	www.example.com	Reverse-DNS Name dieser IP-Adresse
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.7. LrzMonServiceConfig

Diese Tabelle speichert die Konfiguration für die Skripte des LRZmonitors. Jeder Rechner, der durch den LRZmonitor überwacht wird, hat einen Eintrag in dieser Tabelle.

Feld	Typ	Null	Beispiel	Beschreibung
osSetupId	INT	Nein	<OsSetup>	Verweis auf die Betriebssysteminstallation
chkConf	CHKSCRIPTSTATE	Nein	enabled	chk-conf.sh ausführen
chkDfs	CHKSCRIPTSTATE	Nein	disabled	chk-dfs.sh ausführen
chkPerm	CHKSCRIPTSTATE	Nein	testing	chk-perm.sh ausführen
chkRpm	CHKSCRIPTSTATE	Nein	enabled	chk-rpm.sh ausführen
chkScr	CHKSCRIPTSTATE	Nein	disabled	chk-scr.sh ausführen
chkTsm	CHKSCRIPTSTATE	Nein	testing	chk-tsm.sh ausführen
chkUpd	CHKSCRIPTSTATE	Nein	enabled	chk-upd.sh ausführen
chkVm	CHKSCRIPTSTATE	Nein	disabled	chk-vm.sh ausführen
permFiles	TEXT	Nein	/etc/ permissions	Dateien mit Berechtigungsinformationen (getrennt durch Doppelpunkt)
permMd5Sum	TEXT	Nein	3b7ee3b7 4637f17f 27d26b67 3aa7bec5	Prüfsummen der Berechtigungsdateien (getrennt duch Doppelpunkt)
dfsLimits	TEXT	Nein	L:99.0%:B	Konfiguration für Speicherplatzlimits
confStateBase	COLORBASE	Nein	green	Status Konfigurationsprüfung
confStateMode	COLORMODE	Nein	off	Status Konfigurationsprüfung
dfsStateBase	COLORBASE	Nein	green	Status Dateisystemprüfung
dfsStateMode	COLORMODE	Nein	off	Status Dateisystemprüfung
permStateBase	COLORBASE	Nein	green	Status Berechtigungsprüfung
permStateMode	COLORMODE	Nein	off	Status Berechtigungsprüfung
rpmStateBase	COLORBASE	Nein	green	Status Updateprüfung und Installation
rpmStateMode	COLORMODE	Nein	off	Status Updateprüfung und Installation
tsmStateBase	COLORBASE	Nein	green	Status Tsmprüfung
tsmStateMode	COLORMODE	Nein	off	Status Tsmprüfung
updStateBase	COLORBASE	Nein	green	Status Updateprüfung Distribution und Kernel
updStateMode	COLORMODE	Nein	off	Status Updateprüfung Distribution und Kernel
vmStateBase	COLORBASE	Nein	green	Status VMware Prüfung
vmStateMode	COLORMODE	Nein	off	Status VMware Prüfung
kernelStateBase	COLORBASE	Nein	green	Status Kernel Aktualitätsprüfung
kernelStateMode	COLORMODE	Nein	off	Status Kernel Aktualitätsprüfung
osStateBase	COLORBASE	Nein	green	Status Distributions Aktualitätsprüfung
osStateMode	COLORMODE	Nein	off	Status Distributions Aktualitätsprüfung
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<Data-Source>	Verweis auf Datenquelle

B.8. MonitorInfo

Diese Tabelle speichert statische Daten zu den an die Computer angeschlossenen Monitoren. Gespeichert werden Informationen zur Hardware wie Hersteller und Inventarnummern.

Feld	Typ	Null	Beispiel	Beschreibung
idMonitorInfo	INT	Nein	42	eindeutiger Schlüssel
computerSystemId	INT	Nein	<ComputerSystem>	Computer, an den dieses Gerät angeschlossen ist
manufacturer	VARCHAR(100)	Nein	DELL	Hersteller
model	VARCHAR(100)	Nein	19er DELL FlatPanel 1907FP	Modellbezeichnung
serialNumber	VARCHAR(45)	Nein	UY54669RBABC	Seriennummer
inventory	VARCHAR(45)	Nein	1234	Inventarnummer
firstPower	DATE	Nein	40483	Datum der Inbetriebnahme
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.9. NicConfig

Hier werden Informationen zur Konfiguration der Netzwerkkarten erfasst. Dazu gehören zum Beispiel Gerätenamen und der verwendete Treiber.

Feld	Typ	Null	Beispiel	Beschreibung
osSetupId	INT	Nein	<OsSetup>	Verweis auf die Betriebssysteminstallation
interfaceName	VARCHAR(15)	Nein	eth0	Name der Schnittstelle (betriebssystemabhängig)
parentHardwareDeviceId	INT	Ja	<NicInfo>	Verweis auf Hardware
macKind	MACKIND	Nein	ethernet	Art des physischen Netzzugriffs
macAddress	VARBINARY(20)	Nein	003048AABB00	MAC-Adresse
driver	VARCHAR(45)	Nein	e1000	Verwendeter Treiber (betriebssystemabhängig)
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.10. OsSetup

Jedes Computersystem kann eine Reihe von Betriebssysteminstallationen enthalten (die meisten enthalten genau eine). Diese Tabelle erfasst die Betriebssysteminstallationen sowie die Konfiguration der installierten Betriebssysteme.

Feld	Typ	Null	Beispiel	Beschreibung
idOsSetup	INT	Nein	42	eindeutiger Schlüssel
computerSystemId	INT	Nein	<ComputerSystem>	Computer, auf dem dieses Betriebssystem installiert ist
osName	VARCHAR(45)	Nein	SuSE Linux Enterprise Server	Name des Betriebssystems
osVersion	VARCHAR(45)	Nein	11.0	Version des Betriebssystems
osArch	OSARCHES	Nein	x86_64	Systemarchitektur des Betriebssystems
osKernelVersion	VARCHAR(45)	Nein	2.6.32.27-0.2-default	Kernelversion des Betriebssystems
setupTime	DATETIME	Nein	13.05.2009 09:10	Installationszeitpunkt des Betriebssystems
bootTime	DATETIME	Nein	18.01.2011 02:04	Zeitpunkt des letzten Betriebssystemneustarts
hostname	VARCHAR(45)	Nein	app01	Hostname
domainname	VARCHAR(255)	Nein	lrz.de	DNS Domänenname
accounts	VARCHAR(255)	Nein	lu64abc	Liste spezieller Accounts (Komma getrennt)
adminMail	VARCHAR(255)	Nein	root@lrz.de	Liste von eMail Adressen der Administratoren (Komma getrennt)
svcAdmin	VARCHAR(255)	Nein	Webmaster	verantwortlicher Dienstadministrator oder Gruppe
mainUser	VARCHAR(255)	Nein	Ralf Glauberman	Name des Hauptbenutzers
keyboardLayout	VARCHAR(10)	Nein	DE	Tastaturlayout
runlevelCurrent	CHAR(1)	Nein	3	aktuelles Runlevel
runlevelLast	CHAR(1)	Nein	N	vorheriges Runlevel
maintain	VARCHAR(255)	Nein	jeder Di 06:00 Uhr	vereinbartes Wartungsfenster
packageListId	INT	Ja	<LongText>	Verweis auf Packetliste
idsStatusId	INT	Ja	<LongText>	Verweis auf den IDS-Status
lastAdmMailId	INT	Ja	<LongText>	Verweis auf die letzte Administrator-eMail
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.11. NicConfigParentDevice

Hier werden Informationen zu Abhängigkeiten von logischen Netzwerkschnittstellen gespeichert. Zum Beispiel sind die virtuellen Netzwerkschnittstellen einer VPN-Verbindung oder einer Bonding-Konfiguration Kinder der von ihnen verwendeten physischen Netzwerkschnittstellen.

Feld	Typ	Null	Beispiel	Beschreibung
childOsId	INT	Nein	<NicConfig>	Verweis auf zugehöriges Netzwerkinterface
childInterfaceName	VARCHAR(15)	Nein	<NicConfig>	Verweis auf zugehöriges Netzwerkinterface
parentOsId	INT	Nein	<NicConfig>	Verweis auf zugehöriges Netzwerkinterface
parentInterfaceName	VARCHAR(15)	Nein	<NicConfig>	Verweis auf zugehöriges Netzwerkinterface
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.12. Folder

Die Tabelle Folder speichert eine Ordnerhierarchie, in der die Computersysteme einsortiert werden können. Eine genaue Beschreibung findet sich im Kapitel 3.2.3.

Feld	Typ	Null	Beispiel	Beschreibung
idFolder	INT	Nein	42	eindeutiger Schlüssel
parentId	INT	Ja	<Folder>	Verweis auf übergeordneten Ordner
name	VARCHAR(255)	Nein	Ordner	Ordnername
kind	FOLDERKIND	Nein	simpleFolder	Art des Ordners
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.13. LongText

Diese Tabelle dient der effizienten Speicherung besonders langer Textfelder. Der Hintergrund, warum diese Felder in eine eigene Tabelle ausgelagert wurden, ist in Kapitel 4.1.2 beschrieben.

Feld	Typ	Null	Beispiel	Beschreibung
idLongText	INT	Nein	42	eindeutiger Schlüssel
kind	LONGTEXTKIND	Nein	comment	Art des Textes
text	LONGBLOB	Nein	sehr langer Text	Langer Text, Bedeutung abhängig von kind
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.14. MonitorConfig

Diese Tabelle enthält Informationen zur Konfiguration der an die Computer angeschlossenen Monitore. Insbesondere werden hier die verwendete Auflösung sowie die Bildwiederholfrequenzen gespeichert.

Feld	Typ	Null	Beispiel	Beschreibung
idMonitorConfig	INT	Nein	42	eindeutiger Schlüssel
osSetupId	INT	Nein	<OsSetup>	Verweis auf die Betriebssysteminstallation
hsyncMin	MEDIUMINT	Nein	30000	minimaler HSync-Wert in Hz
hsyncMax	MEDIUMINT	Nein	81000	maximaler HSync-Wert in Hz
vrefMin	MEDIUMINT	Nein	56	minimaler VRef-Wert in Hz
vrefMax	MEDIUMINT	Nein	76	maximaler VRef-Wert in Hz
hRes	SMALLINT	Nein	1280	horizontale Auflösung
vRes	SMALLINT	Nein	1024	vertikale Auflösung
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.15. NicInfo

Diese Tabelle erfasst die in den Computern physisch verbauten Netzwerkadapter.

Feld	Typ	Null	Beispiel	Beschreibung
idNicInfo	INT	Nein	42	eindeutiger Schlüssel
computerSystemId	INT	Nein	<ComputerSystem>	Computer, der dieses Gerät enthält
hardwareName	VARCHAR(255)	Nein	Intel Corporation 82573L Gigabit Ethernet	Hardwarebezeichnung
busId	VARCHAR(12)	Nein	0e:00.0	PCI Busadresse dieses Gerätes
wolEnabled	BOOLEAN	Nein	0	ist auf diesem Gerät Wake-On-Lan aktiviert
macKind	MACKIND	Nein	ethernet	Art des physischen Netzzugriffs
macAddress	VARBINARY(20)	Nein	003048AABB00	MAC-Adresse
vmDeviceKind	VMNICDEVKIND	Ja	vmxnet3	Art der emulierten virtuellen Netzwerkkarte
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.16. OsSetupUsesFilesystem

Hierdurch wird die n:m-Beziehung zwischen Betriebssystemen und Dateisystemen realisiert. In der Regel verwendet eine Betriebssysteminstallation mehrere Dateisysteme, aber im Falle von Clustering-Dateisystemen kann auch ein Dateisystem durch mehrere Betriebssysteme verwendet werden.

Feld	Typ	Null	Beispiel	Beschreibung
osSetupId	INT	Nein	<OsSetup>	Verweis auf die Betriebssysteminstallation
filesystemId	INT	Nein	<FileSystem>	Verweis auf ein Dateisystem
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.17. PhysicalLink

Die Datensätze dieser Tabelle erfassen physische Netzwerkverbindungen zwischen Komponenten. Gespeichert werden die verbundenen Komponenten sowie die physischen Verbindungsparameter.

Feld	Typ	Null	Beispiel	Beschreibung
nicInfoId1	INT	Nein	<NicInfo>	Verweis auf eine physische Netzwerkkarte
nicInfoId2	INT	Ja	<NicInfo>	Verweis auf eine physische Netzwerkkarte
linkSpeed	BIGINT	Nein	1000000000	Verbindungsgeschwindigkeit in Bits pro Sekunde
duplexMode	DPLXMODE	Nein	full	aktueller Duplex-Modus
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.18. Schema

Diese Tabelle, welche Teil des Metaschemas ist, erlaubt eine sichere Revisionierung der Datenbank. Die genaue Funktionsweise ist im Kapitel 3.2.4 beschrieben.

Feld	Typ	Null	Beispiel	Beschreibung
revision	INT	Nein	1	eindeutige Revisionsnummer
isCurrent	BOOLEAN	Nein	1	1 wenn dies die aktuelle Schemaversion ist, sonst 0
readCompatible	BOOLEAN	Nein	1	1 wenn Lesen unterstützt wird, sonst 0
writeCompatible	BOOLEAN	Nein	1	1 wenn Schreiben unterstützt wird, sonst 0
updateTime	DATETIME	Nein	29.01.2011 23:49	Zeitpunkt der Aktualisierung auf diese Revision
author	VARCHAR(100)	Nein	Ralf Glauberman	Urheber dieser Änderung

B.19. SharedStorage

Diese Tabelle erfasst im Netzwerk freigegebenen Speicherplatz. Dabei kann es sich sowohl um Dateifreigaben wie NFS oder SMB/CIFS Freigaben handeln, wie auch um freigegebene Block-Devices, zum Beispiel mittels ISCSI.

Feld	Typ	Null	Beispiel	Beschreibung
idSharedStorage	INT	Nein	42	eindeutiger Schlüssel
storageDeviceId	INT	Ja	<StorageDevice>	Verweis auf Speichergerät
osSetupId	INT	Ja	<OsSetup>	Verweis auf die Betriebssysteminstallation
shareType	STORAGESHARE-TYPE	Nein	nfs	Art der Netzwerkfreigabe
shareName	VARCHAR(100)	Nein	/nfs	Name der Netzwerkfreigabe
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.20. TsmServiceConfig

Hier wird die Konfiguration des Tivoli Storage Manager (TSM) Clients erfasst. Diese besteht primär aus der Netzwerkadresse des verwendeten TSM Servers.

Feld	Typ	Null	Beispiel	Beschreibung
osSetupId	INT	Nein	<OsSetup>	Verweis auf die Betriebssysteminstallation
nodename	VARCHAR(100)	Nein	s15	Name des verwendeten TSM-Servers
port	SMALLINT	Nein	1650	Port-Nummer des TSM-Dienstes an diesem Server
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.21. VmComputerSystem

Diese Tabelle speichert die spezifischen Eigenschaften von VMware virtualisierten Maschinen.

Feld	Typ	Null	Beispiel	Beschreibung
computerSystemId	INT	Nein	<ComputerSystem>	Identifikationsnummer dieses Betriebssystems
vmHostId	INT	Nein	<VmHost>	Verweis auf das Hostsystem
resourcePoolId	INT	Nein	<Folder>	Verweis auf den Resource-Pool dieses Systems
vmName	VARCHAR(100)	Nein	app01	Name der virtuellen Maschine
configVersion	VMCONFIG-VERSION	Nein	v7	Version der Konfiguration
toolsVersion	INT	Nein	8290	Version der VMware Tools
guestFullName	VARCHAR(100)	Nein	Suse Linux Enterprise 11 (64-Bit)	Bezeichnung des Gastbetriebssystems
overallStatusBase	COLORBASE	Nein	green	Gesamtzustand dieser VM
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.22. VmDataStore

VMware verwendet so genannte Datastores zur Speicherung der Medienabbilder von virtuellen Laufwerken. Ein Datastore kann dabei beliebig viele Abbilder speichern und greift seinerseits auf einen Netzwerkspeicher zu, wo diese Abbilder abgelegt werden. Diese Tabelle speichert die in VMware konfigurierten Datastores.

Feld	Typ	Null	Beispiel	Beschreibung
idVmDataStore	INT	Nein	42	eindeutiger Schlüssel
sharedStorageId	INT	Nein	<SharedStorage>	Verweis auf Netzwerkfreigabe
name	VARCHAR(45)	Nein	datastore1	Name des DataStores
capacity	BIGINT	Nein	141197049856	Speicherkapazität in Bytes
freeSpace	BIGINT	Nein	140608798720	freier Speicher in Bytes
accessible	BOOLEAN	Nein	1	1 wenn Zugriff möglich ist, 0 sonst
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.23. VmDrive

Diese Tabelle enthält Informationen zu virtuellen Laufwerken von VMware virtuellen Maschinen. Gespeichert werden alle Arten von Laufwerken sowie der Speicherort des Medienabbilds.

Feld	Typ	Null	Beispiel	Beschreibung
idVmDrive	INT	Nein	42	eindeutiger Schlüssel
vmComputerSystemId	INT	Nein	<VmComputer-System>	Computer, der dieses Laufwerk enthält
dataStoreId	INT	Ja	<DataStore>	Datenspeicher, auf dem dieses virtuelle Laufwerk gespeichert ist
name	VARCHAR(255)	Nein	CD-/DVD-Laufwerk 1	Name des Laufwerks
kind	VMDRIVEKIND	Nein	cdrom	Art des Laufwerks
backendPath	VARCHAR(255)	Nein	/vmware/linux.iso	Pfad zum Speicherbackend dieses Laufwerks
backendType	VMDRIVE-BACKKIND	Nein	remoteDevice	Art des Speicher-Backends
capacity	BIGINT	Nein	352845230	Kapazität in Bytes
persistence	VMHDDPER-SISTENCE	Nein	persistent	Behandlung dieses Laufwerks im Bezug auf Snapshots
diskType	VMHDDTYPE	Nein	flat	Art der Speicherung des Medienabbilds
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.24. VmHost

In dieser Tabelle sind die spezifischen Eigenschaften von VMware Hostsystemen und deren Hypervisor erfasst.

Feld	Typ	Null	Beispiel	Beschreibung
osSetupId	INT	Nein	<OsSetup>	Verweis auf die Betriebssysteminstallation
version	VARCHAR(45)	Nein	4.1.0	Versionsnummer des Hypervisors
build	VARCHAR(45)	Nein	320137	Buildnummer des Hypervisors
apiVersion	VARCHAR(45)	Nein	4.1	API-Version des Hypervisors
cpuUsage	FLOAT	Nein	0.1932	Prozessorauslastung (0-1)
memUsage	FLOAT	Nein	0.7545	Arbeitsspeicherauslastung (0-1)
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.25. VmSnapshots

Hier werden Snapshots von VMware virtualisierten Maschinen gespeichert.

Feld	Typ	Null	Beispiel	Beschreibung
idVmSnapshots	INT	Nein	42	eindeutiger Schlüssel
vmComputerSystemId	INT	Nein	<ComputerSystem>	Computer, von dem dieser Snapshot stammt
parentSnapshotId	INT	Ja	<VmSnapshot>	übergeordneter Snapshot
name	VARCHAR(100)	Nein	working sp3	Name des Snapshots
size	BIGINT	Nein	32529875	benötigter Speicherplatz in Bytes
isCurrent	BOOLEAN	Nein	0	1 wenn der Snapshot aktuell ist, sonst 0
powerState	POWERSTATES	Nein	off	Betriebszustand zur Zeit der Snapshoterzeugung
created	DATETIME	Nein	21.07.2010 05:41	Zeitpunk der Snapshoterzeugung
description	TEXT	Nein	neu installiertes System	Beschreibung dieses Snapshots
timestamp	DATETIME	Nein	15.01.2011 17:22	Zeitstempel
datasource	SMALLINT	Nein	<DataSource>	Verweis auf Datenquelle

B.26. DataSource

Diese Tabelle verwaltet eine Liste aller Datenquellen (Abfrageagenten), welche Daten in die Datenbank einfügen. Siehe hierzu auch Kapitel 3.2.4.

Feld	Typ	Null	Beispiel	Beschreibung
idDataSource	SMALLINT	Nein	42	eindeutiger Schlüssel
name	VARCHAR(45)	Nein	vmware	eindeutiger Name (ein Wort)
description	VARCHAR(255)	Nein	VMware vSphere cluster monitoring	kurze Beschreibung (ein Satz)

Abbildungsverzeichnis

Literaturverzeichnis

[Dis10a] Distributed Management Task Force, Inc.: *Common Information Model*, 2010. http://www.dmtf.org/standards/cim.

[Dis10b] Distributed Management Task Force, Inc.: *Web-Based Enterprise Management*, 2010. http://www.dmtf.org/standards/wbem.

[GS04] Gumm, Heinz-Peter und Manfred Sommer: *Einführung in die Informatik*. Oldenbourg, 6. Auflage, 2004.

[Hor07] Horn, Torsten: *Vererbung und Polymorphie mit relationalen Datenbanken*, 2007. http://www.torsten-horn.de/techdocs/sql-vererbung.htm.

[LR06] Lahres, Bernhard und Gregor Rayman: *Praxisbuch Objektorientierung – Von den Grundlagen zur Umsetzung*. Galileo Computing, 2006.

[Mic08a] Microsoft Corporation: *Monitoring Your Windows Server 2008 Infrastructure*, 2008. http://download.microsoft.com/download/4/a/9/4a9a4f66-9873-4612-8615-b71f5c13069d/SCOM2007_WS08_datasheet.pdf.

[Mic08b] Microsoft Corporation: *System Center Configuration Manager Asset Intelligence*, März 2008. http://download.microsoft.com/download/d/0/f/d0f027ac-501c-415a-ba28-85c051d57da4/ConfigMgr%20Asset%20Intelligence%20Whitepaper%20FINAL%20v4.pdf.

[O+11a] Oracle Corporation et al.: *MySQL 5.1 Reference Manual – Chapter 12.1.17. CREATE TABLE Syntax*, 2011. http://dev.mysql.com/doc/refman/5.1/en/create-table.html.

[O+11b] Oracle Corporation et al.: *MySQL 5.1 Reference Manual – Chapter 13. Storage Engines*, 2011. http://dev.mysql.com/doc/refman/5.1/en/storage-engines.html.

[Sch11] Schwabe, Christian: *System- und anwendungsübergreifendes Monitoring am LRZ*, 2011. http://www.nm.ifi.lmu.de/pub/Fopras/#2011.

[VMw09] VMware Inc.: *VMware vCenter Server 4 – Product Datasheet*, 2009. http://www.vmware.com/files/pdf/vcenter_server_datasheet.pdf.