Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jan Lepel

Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen - Prototyp einer Weboberfläche zur vereinfachten Inbetriebnahme individuell erstellter Entwicklungsumgebungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Lepel

Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen - Prototyp einer Weboberfläche zur vereinfachten Inbetriebnahme individuell erstellter Entwicklungsumgebungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: MSc Informatik Oliver Neumann

Eingereicht am: 1. Januar 2015

Jan Lepel

Thema der Arbeit
Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen -
Prototyp einer Weboberfläche zur vereinfachten Inbetriebnahme individuell erstellter Entwicklungsumgebungen

Stichworte
Ad hoc Umgebung, automatisierter Umgebungsaufbau und Provisionierung

Kurzzusammenfassung
Dieses Dokument …
Jan Lepel

Title of the paper
TODO

Keywords
Keywords, Keywords1

Abstract
This document …

Listings

Inhaltsverzeichnis

1 Einleitung

> "Es ist nicht zu wenig Zeit, die wir haben, sondern es ist zu viel Zeit, die wir nicht nutzen" - Lucius Annaeus Seneca, Seneca (2005)

Seneca formulierte 49 n. Chr. ein Gefühl das jeder kennt. Die Zeit die er hat, nicht richtig zu nutzen. Technische Neuerungen helfen uns unsere Zeit besser zu planen, mehr Zeit in andere Aktivitäten zu stecken und unsere Prioritäten zu überdenken. Diese Arbeit beschäftigt sich mit dem Teil-Aspekt der Informatik, der Virtualisierung von Servern im Entwicklungsumfeld.

...

1.1 Motivation

TODO [...] Sinn und Zweck dieser Ausarbeitung besteht darin, eine Software zu entwickeln, die durch vereinfachte Handhabung und minimaler Einarbeitungszeit, es dem Benutzer ermöglich eine ad-hoc Umgebung zu erstellen, ohne dass ein bürokratischer Aufwand erforderlich ist und ohne Grundwissen über die darunterliegende Anwendungsstruktur. Der normalerweise große zeitliche Aufwand für die Erstellung von virtuellen Maschinen soll möglichst minimiert werden und es Anwendern sowohl in Unternehmen als auch bei Projektarbeiten erleichtern, sich auf die vorhandenen Usecase zu fokussieren. Somit muss keine Zeit mehr in Aufbau, Installation und Problembehebung investieren werden. [...]

1.2 Zielsetzung

Das Ziel der vorliegenden Arbeit ist es, durch inkrementelles und interaktives Vorgehen eine Applikation zu modellieren, die den Anwendern der Applikation beim Aufbau von virtuellen Umgebungen unterstützt. Je nach Wunsch des Anwenders, wird nicht nur der Aufbau einer Umgebung vereinfacht, sondern auch die direkte Installation von Programmen veranlasst. Eine Weboberfläche soll die entsprechenden Optionen zur Verfügung stellen und dem Anwender

durch seine ausgewählte Funktion leiten. Große Konfigurationen oder komplizierte Einstellungen sollen dem Anwender abgenommen werde. Sie geschehen im Hintergrund. Damit auch ein Sichern oder ein Zurückspielen von vorhandenen virtuellen Maschinen möglich wird, sollten Im- und Exportfunktionen dies untzerstützen. Die Realisierung sollte auf einem zentralen Knotenpunkt stattfinden, um es mehreren Anwendern zu ermöglichen, ihre notwendige Maschine zu erstellen und zu verwalten. Kernaufgaben sollen Open-Source Anwendungen übernehmen, die in ihrem Einsatzbereich etabliert sind. Ebenfalls im Fokus steht die Leichtigkeit der Konfiguration der auszuwählenden Open-Source Anwendung. Bei der Erstellung der einzelnen Softwarekomponenten sind stehts die Prinzipien von hoher Kohäsion und loser Kopplung zu beachten. Darunter wird der Grad der Abhängigkeit zwischen mehreren Hard-/ und Softwarekomponenten verstanden, der Änderungen an einzelnen Komponenten erleichtert.**[TODO: SATZ KORRIGIEREN]** Um auch die Anwendungsoberfläche unkompliziert zu halten, soll der Anwender mit ein paar Klicks zu seinem Ziel geführt werden. Durch das Vermeiden sowohl von unnötigen verschachtelten Menüführungen als auch von einer Vielfalt an Optionen und Konfigurationen, soll der Anwender in der Applikation einen Helfer für seine Tätigkeit finden.

1.3 Themenabgrenzung

Diese Arbeit greift bekannte und etablierte Softwareprodukte auf und nutzt diese in einem zusammenhängenden Kontext. Dabei werden die verwendeten Softwareprodukte nicht modifiziert, sondern für eine vereinfachte Benutzung durch eigene Implementierungen kombiniert. Sie werden mit einem Benutzerinterface versehen, welches die Abläufe visualisiert und dem Benutzer die Handhabung vereinfacht. Die vorzunehmenden Implementierungen greifen nicht in den Ablauf der jeweiligen Software ein, sondern vereinfachen das Zusammenspiel der einzelnen Anwendungen.

1.4 Struktur der Arbeit

[...]

2 Grundlagen

[...]

2.1 Grundlagen der Virtualisierung

Für den Begriff Virtualisierung existiert keine allgemeingültige Definition. In der Regel wird damit der parallele Einsatz mehrerer Betriebssysteme beschrieben. Detaillierter bedeutet das, dass Hardware-Ressourcen zu einer logischen Schicht **[TODO: LOGISCHE SCHICHT ERKLÄREN?!]** zusammengefasst werden und dadurch deren Auslastung optimiert wird. So kann die logische Schicht bei Aufforderung ihre Ressourcen automatisch zur Verfügung stellen. Das Prinzip dahinter ist die Verknüpfung von Servern, Speichern und Netzen zu einem virtuellen Gesamtsystem. Daraus können sich darüber liegende Anwendungen direkt und bedarfsgerecht ihre Ressourcen beziehen.
Grundsätzlich wird unterschieden zwischen

1. **Virtualisierung von Hardware**
   die sich mit der Verwaltung von Hardware-Ressourcen beschäftigt und

2. **Virtualisierung von Software**
   die sich mit der Verwaltung von Software-Ressourcen, wie z.B. Anwendungen und Betriebssystemen beschäftigt.

[Bengel (2008)]

2.1.1 Virtuelle Maschine

Eine virtuelle Maschine ist nach Robert P. Goldberg

> *'a hardware-software duplicate of a real existing computer system in which a statistically dominant subset of the virtual processor's instructions execute directly on the host processor in native mode'* [Siegert und Baumgarten (2006)]

Wörtlich übersetzt handelt es sich also bei einer virtuelle Maschine, um ein Hardware-/Software-Duplikats eines real existierenden Computersystems, in der eine statistisch dominante Untermenge an virtuellen Prozessoren, Befehle im Benutzer-Modus auf dem Host-Prozessor ausführen. Dieses Duplikat kann als Software-Container betrachtet werden, der aus einem vollständigen Satz an emulierten Hardwareressourcen, dem Gastbetriebssystem und entsprechenden Software-Anwendungen besteht. Durch die Containerstruktur wird eine Kapselung hervorgerufen, die es ermöglicht, mehrere virtuelle Maschinen komplett unabhängig auf einem Hostsystem zu installieren und laufen zu lassen. Ist eine virtuelle Maschine fehlerhaft oder nicht mehr erreichbar, betrifft dies nicht automatisch die restlichen parallel laufenden Maschinen und stellt damit einen der charakteristischen Vorteile von virtuellen Maschinen dar, die Verfügbarkeit. Backup-Systeme oder mehrere Instanzen einer Applikationen können so unabhängig auf einem Host ausgeführt werden, ohne sich gegenseitig zu beeinflussen. Durch den strukturellen Aufbau einer virtuellen Maschine, ist es ebenfalls möglich, eine Maschine nach den eigenen Wünschen zu erstellen und diese zu replizieren. Somit wird eine Redundanz geschaffen.
Virtuelle Maschinen können ohne größeren Aufwand verschoben, kopiert und zwischen Hostservern neu zugeteilt werden, um die Hardware-Ressourcen-Auslastung zu optimieren. Administratoren können auch die Vorteile von virtuellen Umgebungen für einfache Backups, Disaster Recovery, neue Deployments und grundlegenden Aufgaben der Systemadministration nutzen, da das Wiederherstellen aus Speicherpunkten oder gespeicherten Abzügen, innerhalb von Minuten zu realisieren ist.

2.1.2 Gastbetriebssystem

Ein übliches Betriebssystem wird im privilegierten Prozessor-Modus ausgeführt. Dies wird auch Kernel-Mode genannt. Dies befähigt es, die absolute Kontrolle über die vorhandenen Ressourcen zu gewinnen. Alle Anwendungen, die auf dem Betriebssystem laufen, werden im sogenannten Benutzer-Modus ausgeführt. Die Privilegien im Benutzer-Modus sind allerdings relativ eingeschränkt. Ein direkter Zugriff wird nur sehr selten und unter genau kontrollierten Bedingungen gestattet. Dies hat den Vorteil, dass kein Programm z. B. durch einen Fehler das System zum Absturz bringen kann. (Reuther (2013))
Eine virtuelle Maschine (siehe 2.1.1) läuft als normaler Benutzer-Prozess im Benutzer-Modus. Dies hat zur Folge, dass ein auf der virtuellen Maschine installierte Betriebssystem, das Gastbetriebssystem, folglich nicht den privilegierten Prozessor-Modus nutzen kann, wie es ein nicht virtualisiertes Betriebssystem könnte. Da weder die Anwendungen noch das entsprechende

Gastbetriebssystem Kenntnis darüber haben, dass sie in einer virtuellen Maschine laufen, müssen ausgeführte Instruktionen, die den Prozessor-Modus erfordern, entsprechend anders gehandhabt werden. Dies ist unter anderem die Aufgabe des Hypervisors (siehe 2.1.3).

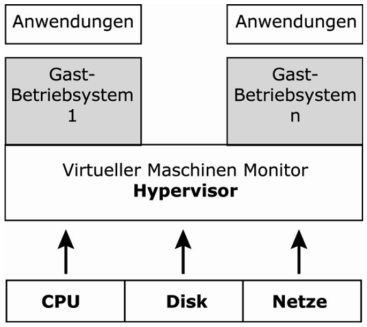


Abbildung 2.1: Betriebssystemvirtualisierung [Siegert und Baumgarten (2006)]

2.1.3 Hypervisor

Der Name des Hypervisors kann von Hersteller zu Hersteller variieren. Bei Microsoft z.B. wird er Hyper-V genannt und bei VMware als ESXi bezeichnet. Der Hypervisor, oder in der Literatur auch VMM (Virtual Machine Monitor) genannt, ist die sogenannte abstrahierende Schicht zwischen der tatsächlich vorhanden Hardware und den ggf. mehrfach existierenden Betriebssystemen. Siehe 2.1. Seine primäre Aufgabe ist die Verwaltung der Host-Ressourcen und deren Zuteilung bei Anfragen der Gastsysteme. Lösen Instruktionen oder Anfagen eines Gastbetriebssystems eine CPU-Exception aus, weil diese im Benutzer-Modus ausgeführt werden, fängt der Hypervisor diese auf und emuliert die Ausführung der Instruktionen (trap and emulate). Die Ressourcen des Hostsystems werden derart verwaltet, dass diese bedarfsgerecht zur Verfügung stehen, egal ob ein oder mehrere Gastsysteme laufen. Zudem zählt unter anderem E/A-Zugriffe (insbesondere Hardwarezugriffe), Speichermanagement, Prozesswechsel und System-Aufrufe.
Der Hypervisor kann in zwei verschiedene Typen kategorisiert werden:

Typ 1 Hypervisor

arbeitet direkt auf der Hardware und benötigt somit kein Betriebssystem, welches zwischen ihm und der Hardware liegt. Alle darüber liegenden virtuelle Maschinen laufen in sogenannten Domains. Weder der Hypervisor noch die anderen Domains sind für die jeweilige Domain sichtbar. Die Verwaltung der laufenden Domains wird durch eine priviligierte Domain geregelt, die in der Dom0 läuft. Dadurch hat die privilegierte Domain die Möglichkeit andere Domains zu starten, stoppen und zu verwalten.

Der Hypervisor Type-1 verfügt selbst über die nötigen Gerätetreiber, um den virtuellen Maschinen sowohl CPU, Speicher als auch I/O zur Verfügung zu stellen. Durch die wegfallende Schicht, nämlich dem nicht benötigten Betriebssystems, gewinnt der Hypervisor Typ 1 an Performance und reduziert den Ressourcenverbrauch. Siehe Abbildung [2.1.3.a].

Typ 2 Hypervisor

lässt durch seine Bezeichnung als 'Hosted' bereits erahnen, dass der Unterschied zu Typ 1 darin besteht, dass er auf einem Hostsystem aufsetzt. Es muss also eine Schicht implementiert sein, die zwischen dem Hypervisor und der Hardware liegt. Siehe Abbildung [2.1.3.b].

Diese Schicht wird durch ein Betriebssystem realisiert, das dem Hypervisor den Zugang zur Hardware durch die eigenen Hardwaretreiber ermöglicht. Ist das Betriebssystem mit der Hardware kompatibel, ist transitiv gesehen, der Hypervisor ebenfalls mit installiert- und ausführbar. Dies vereinfacht die Installation gegenüber dem Hypervisor Typ 1.

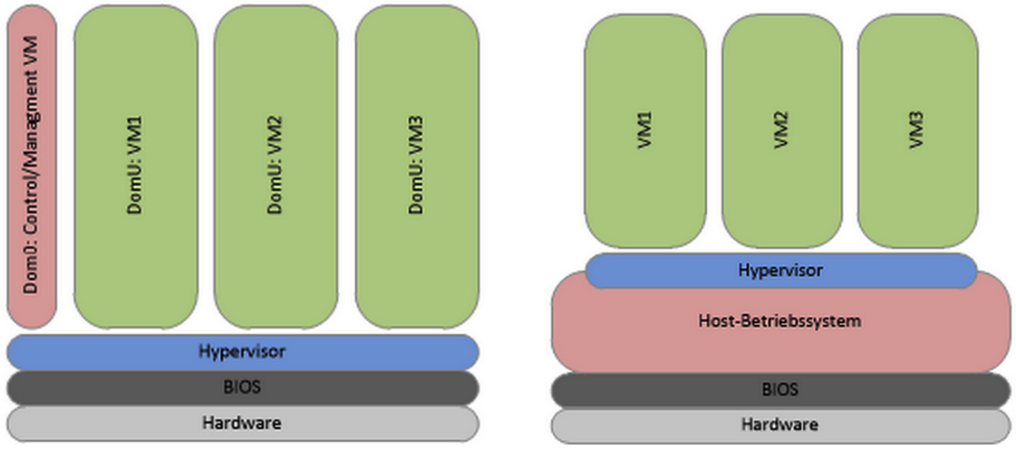
Aus Implementierungssicht gibt es für beide Hypervisoren Vor- und Nachteile. Für einige Bereiche ist die Anforderung eines Betriebssystems nur von Vorteil. Vor allem, wenn es um Hardware- und Treiber-Kompatibilität, Konfigurationsflexibilität und vertraute Management-Tools geht.

Auf der anderen Seite kann genau das zum Nachteil gereichen. Es entsteht nicht nur ein höherer Management-Aufwand, um das Betriebssystem zu konfigurieren und zu verwalten, auch die Performance und der Sicherheitsaspekt leiden unter dieser zusätzlichen Schicht für das Betriebssystem.

2.2 Provisioning/Konfigurationsmanagement

TODO: WELCHER BEGRIFF IST BESSER?

Die Hauptaufgabe eines Konfigurationsmanagement-Systems, im Folgenden nur noch KMS genannt, ist es, eine zuvor definierte Zustandsbeschreibung eines Hosts umzusetzen. Dies kann das Installieren von Softwarepaketen bedeuten, aber auch starten oder beenden von Diensten oder Konfigurationen erstellen/anpassen/entfernen lassen. In der Regel verwenden KMS eigene Agenten auf den Zielsystemen, über die die Kommunikation läuft und die Zustandsbeschreibung realisiert wird. Neuere Anwendungen wie 'Ansible' aus Kapitel 4.2.1, die Konfigurationsmanagement unterstützen, benötigen diese Agenten nicht mehr und realisieren die Kommunikation über eine SSH-Schnittstelle. Pull-basierte Tools, wie beispielsweise 'Puppet', fragen in regelmäßigen Abständen ein zentrales Konfigurations-Repository ab, in



(a) Bezeichnung der linken Grafik

(b) Bezeichnung der rechten Grafik

Abbildung 2.2: Hypervisor Typ 1 und 2

dem die jeweils aktuelle Zustandsbeschreibung der Maschine gespeichert ist. Sie sorgen dafür, dass die Änderungen auf dem Ziel-Client ausgeführt werden. Es spielt keine Rolle, ob das Ziel eine virtuelle Maschine ist oder eine Standard Maschine ist. KMS sind in der Regel dazu fähig, ganze Gruppen an Rechnern parallel zu bearbeiten und die entsprechenden Zustandsbeschreibungen umzusetzen. Bei dem im oberen Abschnitt bereits genannten Beispiel 'Ansible', können mehrere Rechner simpel in Inventory-Dateien als Gruppen zusammengefasst werden. Diese können dann jeweils durch 'Ansible' angesprochen werden, um entsprechende Zustandsbeschreibungen dort auszuliefern. Siehe 2.1

Listing 2.1: Beispiel Inventory-Datei

```
1 [atomic]
2  192.168.100.100
3  192.168.100.101
4 [webserver]
5  192.168.1.110
6  192.168.1.111
```

3 Anforderungsanalyse

Die Anforderungsanalyse hilft Systemeigenschaften und Systemanforderungen der einzelnen beteiligten Gruppen, auch als Stakeholder bezeichnet, zu erfassen, zu analysieren und ggf. eine Diskussionsgrundlage zu schaffen. Das resultierende Ergebnis, kann dann wiederum als Grundstein für ein zukünftiges Lastenheft genutzt werden.
Um die aus dieser Anforderungsbeschreibung hervorgehenden Kernfunktionalitäten und Qualitätsmerkmale näher zu betrachten, wird als erstes das Ziel der Anwendung beschrieben. Dann werden die Stakeholder definiert und ihnen im Anschluss die Kernfunktionalitäten zugeordnet. Im Weiteren wird auf die Qualitätsmerkmale eingegangen, die auch als nichtfunktionale Anforderungen bezeichnet werden.

3.1 Zielsetzung

'Keine Systementwicklung sollte ohne wenigstens eine halbe Seite schriftliche Ziele angegangen werden. Dabei ist es wichtig, quantifizierbare Angaben aufzuzählen, denn Ziele sind Anforderungen auf einer abstrakten Ebene.' Rupp und die SOPHISTen (2014)

Die zu erstellende Applikation soll den Anwender in der Umsetzung und Konfiguration von virtuellen Entwicklungsumgebungen unterstützen. Angestrebte Funktionalitäten, wie der Aufbau einer Entwicklungsumgebung inklusive der automatisierten Installation von Programmen, oder der Austausch von bereits erstellten Entwicklungsumgebungen zwischen beteiligten Benutzern, soll den Anwender in seiner Tätigkeit unterstützen. Dabei spielen das User-Interface und der Funktionsumfang der Applikation eine entscheidende Rolle. Während der Aufbau des User-Interface hilft sich mit geringem Zeitaufwand in die Applikation einzuarbeiten, vereinfacht ein übersichtliches Konfigurationsspektrum die Erstellung der gewünschten virtuellen Umgebung. Die Konfiguration einer virtuellen Maschine muss auch für unerfahrene Nutzer möglich sein und keine speziellen Kenntnisse voraussetzen. Alle virtuelle Maschinen, die zu einem Zeitpunkt aktiv sind, sollten in getrennten Instanzen laufen und von einander unterscheidbar sein. Die Unterscheidbarkeit soll Funktionen wie den Export oder den entfernten

Zugriff auf die Maschine unterstützen. Sie soll dem Anwender die Möglichkeit schaffen, die gewünschte virtuelle Maschine zu beeinflussen, in dem er die Umgebung abschalten oder zerstören kann. Des Weiteren soll die Anwendung Unterstützung bieten, voneinander abhängige Applikationen zu konfigurieren und automatisiert zu installieren.

3.2 Stakeholder

'*Stakeholder* in Softwareentwicklungsprojekten sind alle Personen (oder Gruppen von Personen) die ein Interesse an einem Softwarevorhaben oder dessen Ergebnis haben.' Zörner (2012)

Rolle	Anwender
Beschreibung	Ein Anwender ist ein Benutzer des Systems, ohne administrative Einflüsse auf die Applikation.
Ziel	Gute Benutzbarkeit, schnell erlernbar, komfortable Steuerung, leichter Aufbau der gewünschten Umgebung

Rolle	Administrator
Beschreibung	Der Administrator kann die Applikation wie der Anwender nutzen. Er hat erweiterte Möglichkeiten, im Bezug auf die Konfiguration des Systems.
Ziel	leicht ersichtliche Konfigurationsmöglichkeiten, schnelles auffinden von auftretenden Fehlern, gut protokollierte Fehler

Abbildung 3.1: Stakeholder

3.3 Funktionale Anforderungen

Anforderungen - Anwender

FA 1. Die Anwendung muss über den Browser ausführbar sein, ohne zusätzliche lokale Installationen auf Anwenderseite.

FA 2. Falls der Anwender keine zusätzliche Software auf der virtuellen Maschine haben möchte, muss die Anwendung eine entsprechende Option dafür anbieten.

FA 3. Die Anwendung muss dem Anwender die Möglichkeit bieten, Software auf der gewünschten virtuellen Maschine mit zu installieren.

FA 4. Falls der Anwender diese Option nutzt, sollte die Anwendung Softwarekomponenten vorschlagen oder es ermöglichen eigene Dateien auszuwählen.

FA 5. Die Anwendung sollte fähig sein, dem Administrator Bearbeitungsmöglichkeiten für die vorzuschlagenden Softwarekomponenten anzubieten.

FA 6. Die Anwendung sollte dem Anwender die Option bieten, virtuelle Maschinen zu exportieren.

FA 7. Die Anwendung sollte fähig sein den Export zu komprimieren.

FA 8. Ist der Export durchgeführt worden, muss die Anwendung dies mit einer Meldung auf dem Bildschirm bestätigen.

FA 9. Die Anwendung muss fähig sein, Exporte wieder importieren zu können. Falls währen des Imports Datenfehler auftreten, muss die Anwendung den jeweiligen Fehler (Fehlerbeschreibung) auf dem Bildschirm ausgeben.

FA 10. Ist der Import erfolgreich durchgeführt worden, sollte die Anwendung eine entsprechende Meldung anzeigen.

FA 11. Wenn der Anwender eine virtuelle Maschine erstellen möchte, muss die Anwendung bei wichtigen Konfigurationsschritten, für den Benutzer sichtbare Statusmeldungen anzeigen.

FA 12. Treten Fehler bei der Erstellung einer virtuellen Maschine auf, muss das System eine Fehlermeldung ausgeben.

FA 13. Die Applikation sollte fähig sein, anderen Anwendern bereits erstellte virtuelle Maschinen über das Internet und das lokale Netzwerk zur Verfügung zu stellen.

FA 14. Möchte der Anwender eine bereits erstellte und laufende virtuelle Maschine beenden, muss die Anwendung dafür eine entsprechende Option bieten.

FA 15. Falls der Anwender eine bereits erstellte virtuelle Maschine löschen möchte, muss die Anwendung ihm dafür ein Hilfsmittel bereitstellen.

Anforderungen - Administrator

FA 16. Falls während des Betriebes der Anwendung Änderungen an der Konfiguration durchgeführt werden müssen, sollte die Anwendung dies nur durch einen expliziten Administratoren-Account zulassen.

FA 17. Solange der Administrator angemeldet ist, sollte die Anwendung ihm die Möglichkeit bieten, den Speicherort und Namen von Logdateien persistent zu ändern.

FA 18. Falls während der Änderung ein Fehler auftritt, muss die Anwendung eine Fehlermeldung auf dem Bildschirm ausgeben.

FA 19. Die Anwendung sollte dem Administrator die Option bieten, sich den Inhalt von Logdateien anzeigen zu lassen.

FA 20. Falls eine zu installierende Applikation, Abhängigkeiten zu anderer Software beinhaltet, sollte die Anwendung ein Menü zur Verfügung stellen, dass die Konfiguration von Abhängigkeiten zulässt.

FA 21. Möchte ein erfahrender Benutzer eigene Zustandsbeschreibungen erstellen, sollte die Anwendung das ermöglichen.

Definition Funktionale Anforderungen nach Rupp und die SOPHISTen (2014)

3.4 Use-Cases

Use-Case (Anwendungsfälle) helfen, die fachlichen Anforderungen eines Systems zu bezeichnen, indem dort Interaktionen zwischen System und Benutzer dargestellt werden und das System grob in seine Hauptfunktionen gegliedert wird. Der Business-Use-Case spiegelt dabei ein Gesamtbild aller Funktionalitäten der Applikation wieder und grenzt diese voneinander ab, während die darauf folgenden System-Use-Cases helfen eine erste Skizze der zu entwickelnden Hauptfunktionalitäten zu erstellen, die sich wie folgt aus den funktionalen Anforderungen in Kapitel 3.3 ergeben haben:

1. Erstellung einer virtuellen Maschine
2. Export einer vorhandenen Maschine
3. Der Import einer zuvor erstellten Maschine
4. Eine virtuelle Maschine zugreiffbar für andere Anwender machen (teilen)
5. Software-Abhängigkeiten konfigurieren

3.4.1 Business-Use-Case

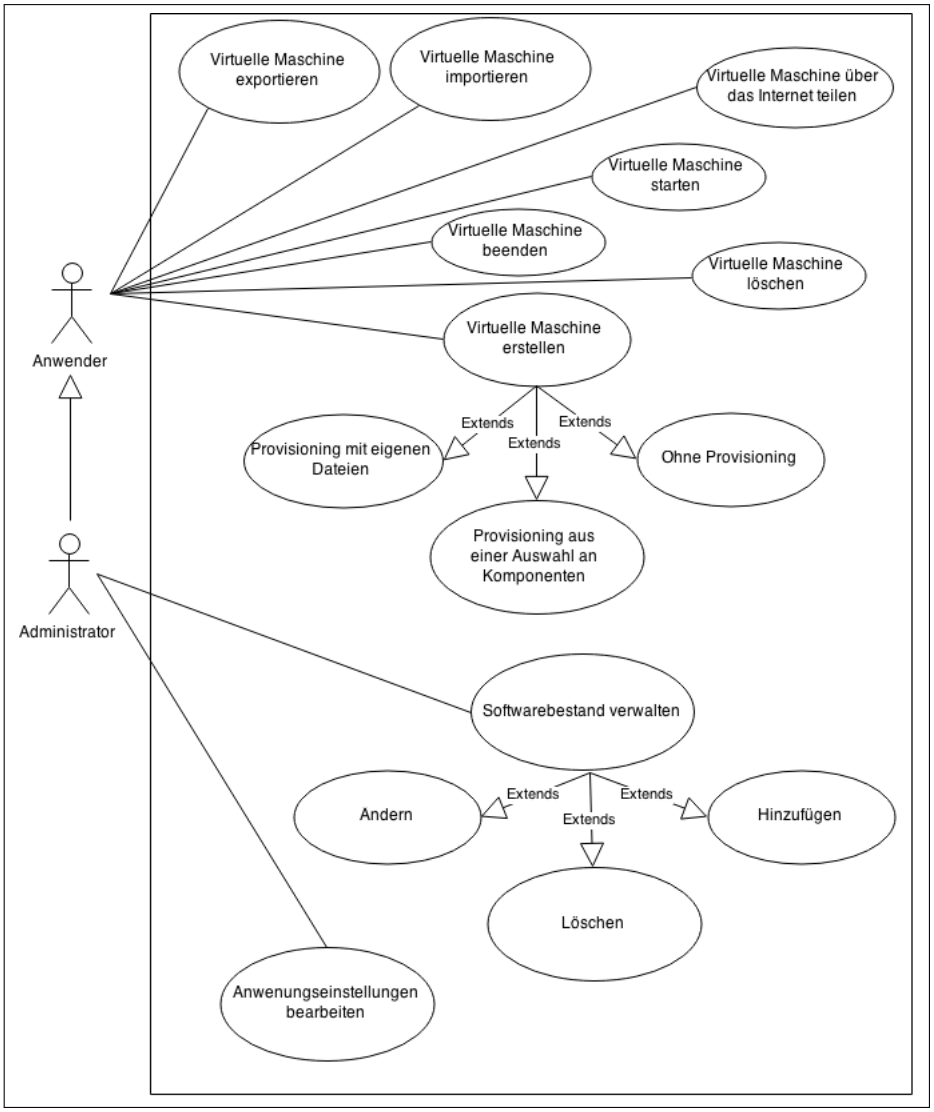


Abbildung 3.2: Titel

3.4.2 System-Use-Case

Use-Case 1 - Virtuelle Maschine erstellen

Bezeichnung	Virtuelle Maschine erstellen
Ziel im Kontext	Erstellung einer virtuellen Maschine
Akteur	Anwender
Auslöser	Der Anwender möchte eine virtuelle Maschine erstellen
Vorbedingung	Die Anwendung ist installiert und lauffähig Der Anwender kann auf die Benutzeroberfläche zugreifen.
Nachbedingung	Der Anwender erhält eine gepackte Datei, in der alle Daten enthalten sind, die für einen erneuten Import nötig wären.
Anforderungen	FA 1, FA 3, FA 4, FA 11, FA 12, FA 13

Erfolgsszenario

1. Der Anwender startet die Applikation über den Browser
2. Der Anwender wird gebeten persönliche Konfigurationsparameter für die zu erstellende Maschine einzugeben
3. Die Applikation schlägt dem Anwender vor, Software mit auf die gewünschte Maschine zu installieren
4. Nach der entsprechenden Auswahl zeigt die Applikation den aktuellen Aufbaustatus der virtuellen Maschine
5. Die Applikation zeigt dem Anwender die Zugriffsmöglichkeiten für die Maschine an

Use-Case 2 - Virtuelle Maschine exportieren

Bezeichnung	Virtuelle Maschine exportieren
Ziel im Kontext	Export aller notwendigen Konfigurationsdateien, um eine Maschine mit der gleichen Konfiguration erneut erstellen zu können
Akteur	Anwender
Auslöser	Der Anwender möchte eine virtuelle Maschine exportieren
Vorbedingung	Die zu exportierende virtuelle Maschine existiert bereits
Nachbedingung	Der Anwender erhält eine gepackte Datei, in der alle nötigen Daten enthalten sind, die für einen erneuten Import nötig wären.
Anforderungen	FA 1, FA 6, FA 7, FA 8, FA 13

Erfolgsszenario

1. Der Anwender startet die Applikation über den Browser
2. Der Anwender wählt die Exportfunktion aus und die Applikation zeigt dem Anwender die verfügbaren Maschinen an
3. Der Anwender sucht sich die entsprechende Maschine aus und mit einem weiteren Klick wird der Download mit den benötigten Dateien gestartet

Use-Case 3 - Virtuelle Maschine importieren

Bezeichnung	Virtuelle Maschine importieren
Ziel im Kontext	Erstellung einer Maschine aus einem Import
Akteur	Anwender
Auslöser	Der Anwender möchte eine virtuelle Maschine importieren
Vorbedingung	Die Anwendung ist installiert und lauffähig Der Anwender kann auf die Benutzeroberfläche zugreifen
Nachbedingung	Die Maschinenkonfiguration konnte importiert werden und eine virtuelle Maschine wurde erstellt
Anforderungen	FA 1, FA 9, FA 10, FA 11, FA 12

Erfolgsszenario

1. Der Anwender startet die Applikation über den Browser
2. Der Anwender wählt die Importfunktion aus und kann die gewünschte(n) Datei(en) hochladen

3. Die Applikation zeigt dem Anwender, dass der Import erfolgreich war
4. Der Anwender kann nun entscheiden, ob er die virtuelle Maschine erstellen lassen möchte

Use-Case 4 - Virtuelle Maschine teilen

Bezeichnung	Virtuelle Maschine teilen
Ziel im Kontext	Eine erstellte Maschine über das Internet oder das lokale Netzwerk für andere Anwender zugreifbar machen
Akteur	Anwender
Auslöser	Der Anwender möchte eine virtuelle Maschine für andere Anwender zugreifbar machen
Vorbedingung	Die Anwendung ist installiert und lauffähig Die zu teilende Maschine ist erstellt
Nachbedingung	Die virtuelle Maschine ist von Intern und/oder Extern zugreifbar
Anforderungen	FA 13
Erfolgsszenario	

1. Der Anwender startet die Applikation über den Browser
2. Der Anwender wählt die gewünschte virtuelle Maschine aus und aktiviert die Teil-Option
3. Die Applikation zeigt dem Anwender die Zugriffsmöglichkeiten auf die virtuelle Maschine

Use-Case 5 - Software-Abhängigkeiten konfigurieren

Bezeichnung	Software-Abhängigkeiten konfigurieren
Ziel im Kontext	Dem Anwender eine Applikation zur Verfügungen stellen, die Installationen von anderen Softwarekomponenten benötigt, welche diese mit installieren soll
Akteur	Administrator / Erfahrener Anwender
Auslöser	Eine Installation einer Anwendung, die mehrere Softwarekomponenten benötigt
Vorbedingung	Wissen über die zu installierenden Softwarekomponenten
Nachbedingung	Eine Applikation, die dem Anwender zur Verfügung steht und alle abhängigen Softwarekomponenten verbirgt und installiert
Anforderungen	FA 3, FA 5, FA 20, FA 21

Erfolgsszenario

1. Der Administrator startet die Applikation über den Browser
2. Der Administrator begibt sich in das Konfigurationsmenü
3. Der Administrator kann dort neue Software zur bestehenden Auswahl hinzufügen
4. Der Administrator gibt die entsprechenden Optionen wie z.B. Name der Softwarekomponente ein
5. Wählt Abhängigkeiten aus
6. Definiert Kopier- und Entpackvorgänge
7. Wählt ggf. auszuführende Scripte aus
8. Oder definiert sich selber eine Zustandsbeschreibung im gewünschten Format

3.5 Nichtfunktionale Anforderungen

In der Literatur findet sich keine einheitliche Definition von nichtfunktionalen Anforderungen, Bezogen auf das Design eines Systems sind die nichtfunktionalen Anforderungen für den Architekten von besonderer Bedeutung (Chung u. a. (1999)).
Durch nichtfunktionale Anforderungen können neue Lösungsmöglichkeiten vorgegeben werden oder schlicht die Menge an potentiellen Designentwürfen, im Bezug auf die Funktionalitäten, reduziert werden (Burge und Brown (2002)).

Im Wesentlichen gibt es eine begrenze Auswahl an Definitionen und Perspektiven, die im Folgenden nach Rupp und die SOPHISTen (2014) zusammengefasst werden.

1. **Technologische Anforderungen**
   Die detailliertere Beschreibung von Lösungsvorgaben oder der Umgebung, in der das System betrieben werden soll, können und sollen den Lösungsraum, für die Realisierung des Systems beschränken.

2. **Qualitätsanforderungen**
   Qualitätsanforderungen lassen sich in detailliertere Unterpunkte unterteilt werden. Dies kann nach zwei Standards erfolgen: ISO 25000 und ISO 9126. Mittlerweile ist jedoch der ISO 9126 Standard in ISO 25000 aufgegangen. Beide Standards sollen sicherstellen, dass über Qualitätsanforderungen die Qualität des Systems und des Entwicklungsprozesses festgelegt wird.

3. **Anforderungen an die Benutzeroberfläche**
   Die Anforderungen, die festlegen, wie sich die Anwendung dem Benutzer darstellt, werden unter 'Anforderungen an die Benutzeroberfäche' gebündelt.

4. **Anforderungen an die sonstigen Lieferbestandteile**
   Alle Produkte, die zu dem System oder der Anwendung geliefert werden müssen, wie z.B. ein Handbuch oder Quellcode, werden unter 'Anforderungen an die sonstigen Lieferbestandteile' beschrieben.

5. **Anforderungen an durchzuführende Tätigkeiten**
   Die 'Anforderungen an durchzuführende Tätigkeiten' beeinflussen Tätigkeiten, wie Wartung oder Support, die der Entwicklung nachgelagert sind.

6. **Rechtliche-vertragliche Anforderungen**
   'Rechtliche-vertragliche Anforderungen' beschreiben die Regelungen, die zwischen Auftraggeber und Auftragnehmer vor der Entwicklung des System des Auftraggebers oder deren Anwendung, festgelegt werden müssen.

Im Folgenden werden die nichtfunktionalen Anforderungen hinsichtlich der Punkte 1) 'Technologische Anforderungen' und 3) 'Anforderungen an die Benutzeroberfläche' betrachtet, da diese Zielführend für die Entwicklung der angestrebten Applikation sind.

Technologische Anforderungen

1. Das für den Betrieb der Anwendung zugrunde liegende Betriebssystem muss Ubuntu 12.04 oder höher sein.
2. Die Anzahl der gleichzeitig laufenden virtuellen Umgebungen, liegt maximal bei 10.
3. Die Kommunikation zwischen Frontend und Backend muss nicht verschlüsselt ablaufen.
4. Softwareupdates der benutzen Softwarekomponenten müssen mit Rücksprache des Entwicklers erfolgen.

Qualitätsanforderungen

1. Die Installation und Inbetriebnahme der Anwendung sollte über einen automatischen Installationsprozess erfolgen.
2. Die Anwendung sollte zu 99.0 Prozent der Zeit lauffähig sein.
3. Jeder auftretende Fehler ist eindeutig identifizierbar und nachvollziehbar.
4. Änderungen am vorgeschlagenen Softwarebestand müssen innerhalb von <10 Sekunden in der Anwendungsoberfläche sichtbar sein (bezogen auf Funktionale Anforderungen FA 4.).
5. Falls das Betriebssystem auf eine höhere Version aktualisiert werden soll, muss dies ohne Änderungen an dem Quellcodes der Anwendung vorgenommen werden können.
6. Soll ein anderer Provisionierer verwendet werden, muss der Aufwand des Austausches bei unter einem Personentag liegen.
7. Wird angedacht weitere Grundfunktionalitäten zu implementieren, soll dies möglichst einfach erfolgen.
8. Das Importieren von virtuellen Maschinen sollte <10 Minuten betragen.
9. Die Validierung der zu importierenden Daten sollte <2 Minuten betragen.
10. Der beim Import verwendete Validierungsalgorithmus muss unter 0.5 Personentagen austauschbar sein. (bezogen auf Funktionale Anforderungen FA 9.)
11. Der Export einer virtuellen Maschine sollte <5 Minuten betragen.

Anforderungen an die Benutzeroberfläche

1. Ein Benutzer ohne Vorkenntnisse muss bei der erstmaligen Verwendung des Systems innerhalb von maximal 10 Minuten in der Lage sein, die gewünschte Funktionalität zu lokalisieren und zu verwenden.

2. Die Anwendung muss den Oberflächendialog 'Virtuelle Maschine exportieren' mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ...). Keine umzusetzende Anforderung sind die genaue Größe und die Anordnung der einzelnen Elemente.
   BILD EINFÜGEN

3. Die Anwendung muss den Oberflächendialog 'Virtuelle Maschine importieren' mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ...). Keine umzusetzenden Anforderungen sind die genaue Größe und die Anordnung der einzelnen Elemente.
   BILD EINFÜGEN

4. Die Anwendung muss den Oberflächendialog 'Virtuelle Maschine erstellen - Mit Provisioning aus einer Auswahl an Komponenten' mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ...). Keine umzusetzenden Anforderungen sind die genaue Größe und die Anordnung der einzelnen Elemente.
   BILD EINFÜGEN

5. Die Anwendung muss den Oberflächendialog 'Virtuelle Maschine über das Internet teilen"mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ...). Keine umzusetzende Anforderungen sind die genaue Größen und die Anordnung der einzelnen Elemente.

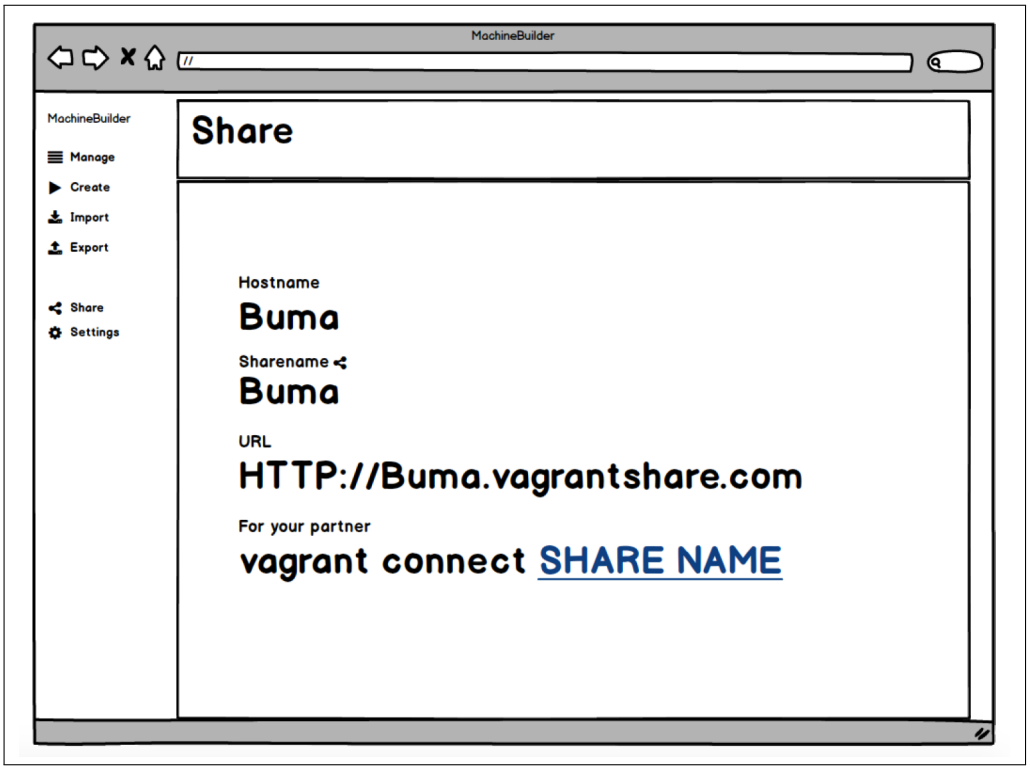


Abbildung 3.3: Titel

3.6 Randbedingungen

Um anwendungs- und problembezogene Entwurfsentscheidungen treffen zu können, werden Faktoren analysiert, die die Architekturen der Anwendung beeinflussen oder einschränken können. Dies geschieht über die im Vorfeld formulierten Anforderungen. Laut Starke (2014) werden diese Einflussfaktoren und Randbedingungen in folgende Kategorien unterteilt:

- Organisatorische und politische Faktoren: Mancher solcher Faktoren wirken auf ein bestimmtes System ein, während andere eine Vielzahl an Projekten eines Unternehmens beeinflussen können. (Rechtin und Maier, 2000) charakterisiert diese Faktoren als 'facts of life'.
- Technische Faktoren: Durch sie wird das technische Umfeld geprägt und entsprechend eingeschränkt. Sie betreffen nicht nur die Ablaufumgebung des zu entwickelnden Systems, sondern umfassen auch die technischen Vorgaben für die Entwicklung, einzusetzender Software und vorhandener IT-Systeme.

Da weder die organisatorischen-, noch die politischen Faktoren auf dieses Projekt Einfluss haben, werden diese nicht weiter betrachtet.

3.6.1 Technische Randbedingungen

TODO: 64 Bit OS müssen möglich sein!

Randbedingung	Erkläuterung
Server Hardware	Die Leistung des Servers sollte entsprechend der Anforderungen genügen. Es muss möglich sein, mehrere virtuelle Maschinen gleichzeitig laufen zu lassen, ohne das es die einzelnen Maschinen beeinflusst.
Server Betriebssystem	Die Lösung sollte auf einer einem Ubuntu Server 64Bit installiert und betrieben werden.
Implementierung in Ruby	Implementiert der Anwendung erfolgt in Ruby mit dem Framework Sinatra. Da keine seperate Frontend Kommunikation benötigt wird, bietet sich Ruby als Backend Sprache an. Entwickelt wird in der Version 1.9, welche als Stabil gilt.
Fremdsoftware	Fremdsoftware die hinzugezogen wird, sollte idealerweise frei verfügbar und kostenlos sein. Es sollte drauf geachtet werden, dass die Software etabliert ist und der Fokus auf eine Stabile Version gelegt wird.
Web-Frontend	Die Kompatibilität zu allen gängigen Browsern wird nicht weiter betrachtet. Optimiert wird für ausschließlich für Mozilla Firefox.

Abbildung 3.4: Titel

3.7 Zusammenfassung

TODO

4 Evaluation

Um herauszufinden, ob und welche Softwareprodukte der Anwendung hinzugefügt werden können, muss eine Evaluation der vorhanden Produkte, die auf dem Markt sind durchgeführt werden durch eine Marktrecherche. Es ist zu eruieren, ob gewünschte Funktionalitäten unter Verwendung der technischen Randbedingungen aus Kapitel 3.6.1 - Unterpunkt 'Femdsoftware', von externen Softwarekomponenten unterstützt oder sogar durchgeführt werden können, die etabliert und kostenlos sind.

4.1 Virtualisierungsprodukte

Virtualisierungsprodukte oder Tools helfen virtuelle Maschinen für temporäre Zwecke aufzubauen. Beispielsweise um ein alternatives Betriebssystem zu testen, Softwareprodukte auszuprobieren oder Entwicklungsumgebungen zu erstellen. Die wohl derzeit (Stand Mitte 2015) bekanntesten und etabliertesten Virtualisierungslösungen werden von drei verschiedenen Herstellern präsentiert und zum Teil als Freeware angeboten.

1. **VMware Player / Plus**
   Preiswerte Virtualisierungslösung für Unternehmen. Source-Code.

2. **Microsoft Hyper-V**
   Microsofts Virtualisierungslösung, die nur in Windows 8 oder höher lauffähig ist.

3. **Oracle VM VirtualBox**
   Gratis Open-Source-Tool von Oracle mit großem Funktionsumfang und frei verfügbar.

4.1.1 VMware Player (Plus)

Eins der bekanntesten Unternehmen für Virtualisierungslösungen ist VMware. VMware bietet für den Privatanwender den VMware Player an, der das kostenlose Pendant zur professionellen Lösung VMware Workstation / VMware Fusion darstellt. Mittlerweile ist der VMware Player für Unternehmen unter dem Namen VMware Player Plus, zum Preis von ca. 90 Euro zu erwerben. Der VMware Player (Plus) unterstützt ca. 200 Gast-Betriebssysteme, ganz gleich ob 32Bit

oder 64Bit und lässt sich auf Windows und verschiedenen Linux-Distributionen installieren. Virtuelle Maschinen lassen sich nur als VMDK-Format (Virtual Machine Disk) speichern, was somit die Wahl der Dateiformate enorm einschränkt. Außerdem sind Snapshots möglich. Features wie eine virtuelle Maschine in einen vorherigen Systemzustand zurückzuversetzen fehlt gänzlich, genau so wie die Funktion des Klonens. Also des Duplizierens einer Maschine. Export- und Import Funktionen werden vom VMware Player allerdings unterstützt.

4.1.2 Microsoft Hyper-V

Microsoft hat in Windows 8 (und höher) Hyper-V integriert, das über Windows-Funktionen einfach nachinstalliert werden kann. Damit dies möglich ist, mpssen allerdings müssen gewisse Faktoren gegeben sein. Windows 8 muss als Professional-Version vorliegen und als 64-Bit Version. Der Vorteil gegebüber der Konkurrents besteht in der Fähigkeit mehrere virtuelle Maschinen gleichzeitig parallel betreiben zu können ohne signifikanten Performanceverlust. Dies erreicht Hyper-V dank SLAT, einer Technik zur dynamischen RAM Verwaltung.**[TODO: Weitere Infos suchen in der BA]**

4.1.3 Oracle VM VirtualBox

Die im April 2005 entstandene Virtualisierungssoftware von Oracle (erst InnoTek Systemberatung GmbH) eignet sich für Windows, Linux, Mac OS X, FreeBSD und Solaris als Hostsystem. An Gastsystemen wird eine Vielzahl von x64- als auch x86-Betriebssysteme unterstützt, wie diverse Linux Distributionen, Windows ab Version 3.11, Mac OS X, IBM OS/2 und FreeBSD. VirtualBox lässt dem Anwender viele Freiheiten im Speichern seiner virtuellen Umgebung. Etwa vier gängige Formate werden angeboten und ermöglichen einen leichteren Austausch unter Anwendern. Ein weiterer Vorteil besteht darin, dass VirtualBox vollständbdig kostenlos und OpenSource ist. Der Quellcode steht jedem Interessierten zur Verfügung. Im Funktionsumfang enthalten sind das Importieren sowie Exportieren von virtuellen Maschinen, das Erstellen von Snapshots und das Klonen von virtuellen Maschinen.

4.1.4 Zusammenfassung

Die wichtigsten Faktoren bezüglich der zu entwickelnden Anwendung zusammengefasst:

	VMware Player (Plus)	Microsoft Hyper-V	Oracle VM VirtualBox
Host-Betriebssystem	Windows, diverse Linux Distributionen	Windows 8 Pro 64 Bit	Windows, Linux, MacOS X
Gast-Betriebssysteme	Mehr als 200 Gast-Betriebssystemen		Diverse Linux Distributionen, Windows, FreeBSD, Mac OS X, IBM OS/2
64-Bit-Gast-Betriebssystem	Ja	Ja	Ja
Dateiformate für virtuelle Disks	VMDK	VHDX	VMDK, VHD, Parallels Hard Disk, OVF
Snapshots	Nein	Nein	Ja
Klonen	Nein	Nein	Ja
Export von virtuellen Maschinen	Ja		Ja
Preis	90 Euro für Unternehmen	Kostenlos	Kostenlos

4.1.5 Fazit

Auch wenn VMware zu den bekannteren Herstellern gehört und der VMware Player (Plus) eine Vielzahl an Funktionen bietet, macht die Unterscheidung zwischen der Unternehmensvariante und der für Privatanwender die zukünftige Benutzung kompliziert. Sollte die zu erstellende Anwendung in einem nicht privaten Umfeld betrieben werden, so muss auf den VMware-Player Plus zugegriffen werden, sind ggf. Änderungen an der Implementierung vorzunehmen und die entsprechenden Lizenzkosten zu beachten. Außerdem schränkt nicht nur die geringe Auswahl an Dateiformaten zum Export von Maschinen die Funktionsweise ein, sondern auch das Nichtvorhandensein von Snapshot- und Klon Funktionen. Microsofts Lösung stellt sich von den Grundanforderungen her als nicht nutzbar heraus, da Hyper-V Windows als Grundlage benötigt, dieses jedoch nicht den gestellten technischen Randbedingungen (Kapitel 3.6 - Server Betriebssystem) an die zu erstellende Software genügt. Oracles VirtualBox hingegen vereinigt viele für die geforderten Funktionen unterstützende und hilfreiche Aspekte. Die Vielzahl an unterstützenden Dateiformaten kann für die geforderte Exportfunktion interessant

sein. Da VirtualBox für Privatanwender und Firmenkunden kostenlos ist, fällt der Lizensierungsaspekt weg. VirtualBox lässt sich auf diversen Linux-Umgebungen installieren und somit ebenfalls mit den technischen Randbedingungen aus Kapitel 3.6.1 in Einklang bringen. Ggf. könnte der Punkt, dass VirtualBox Open-Source ist, für eine spätere Weiterentwicklungen interessant werden. VirtualBox wäre durch seinen Funktionsumfang in der Lage, wichtige Funktionen in der angestrebten Applikation zu übernehmen. Dies hätte zudem den Vorteil, dass die Funktionsbestandteile nicht aufwändig implementiert werden müssten.

4.2 Konfigurationsmanagement-Systeme

Wie in Kapitel 2.2 beschrieben, helfen Konfigurationsmanagement-Systeme bei der Umsetzung von sogenannten Zustandsbeschreibungen eines Hostes. Kurz gefasst, Konfigurationsmanagement-Systeme können gewünschte Software auf Zielrechner(n) installieren, Dienste und Applikationen starten/beenden, Konfigurationen ändern/erstellen/löschen und wenn gewünscht, dies alles gleichzeitig auf einem oder mehreren Zielrechnern. Durch die Anwendung von Konfigurationsmanagement-Systemen, kann die aus der Anforderungsanalyse herausgestellte Anforderung nach automatisierter Installation von Software übernommen werden. Auch hier gibt es bekannte und häufig verwendete Applikationen, die im Folgenden betrachtet werden.

4.2.1 Ansible

Die Hauptdesignidee bei dem in Python geschriebenen Ansible ist es, Konfigurationen so leicht wie möglich durchführen zu können. Es werden weder aufwändige Deployment-Scripts benötigt, noch komplizierte Syntaxen verwendet. Ansible wird nur auf der Maschine installiert, die die Infrastruktur verwaltet und kann von dort auf die gewünschten Maschinen zugreifen. Die Clients benötigen weder eine lokale Ansible Installation noch andere spezielle Softwarekomponenten. (Hall (2013)) Die Kommunikation zwischen dem Host, auf dem Ansible installiert ist und den Clients wird über SSH ausgeführt. Für Linux-Distributionen, auf denen SSH für den Root Benutzer gesperrt sind, kann Ansible 'sudo' Befehle emulieren, um die gewünschte Zustandsbeschreibung durchzuführen. Windows wird in der aktuellen Version 1.9.1 nicht unterstützt. Zustandsbeschreibungen werden in YAML Syntax ausgeführt und in sogenannte Playbooks geschrieben. Playbooks haben eine simple YAML Struktur und können somit schon vorgefertigt als Template gespeichert und wieder verwendet werden (ScriptRock (2014))

4.2.2 Saltstack

Saltstack oder kurz 'Salt' ist wie Ansible in Python entwickelt worden. Zur Kommunikation mit den gewünschten Clients wird ein Master benötigt und sogenannte Agenten oder Minions, die auf den Zielclients installiert werden müssen. Die eigentliche Kommunikation funktioniert über eine ZeroMQ messaging lib in der Transportschicht, was die Verständigung zwischen Master und Minions vereinfacht. Dadurch ist die Kommunikation zwar schnell, jedoch nicht so sicher die SSH Kommunikation von Ansible. ZeroMq bietet nativ keine Verschlüsselung an und transportiert Nachrichten über IPC, TCP, TIPC und Multicast. Der größte Vorteil von Salt ist die Skalierbarkeit. Diese macht es möglich mehrere Ebenen an Mastern zu erstellen, um so eine bessere Lastverteilung zu erhalten. Salt benutzt für seine Konfigurationsdateien zwar ebenfalls YAML, allerdings müssen in der Konsole ausgeführte Befehle in Python oder PyDSL geschrieben werden. ScriptRock (2014)

4.2.3 Puppet

Der größte Vertreter auf dem Markt im Bereich Konfigurationsmanagement-Systeme ist wohl Puppet von 'puppet labs'. Puppet hat nicht nur eine ausgereifte Monitoring-Oberfläche und läuft auf allen gängigen Betriebssystemen, sondern ist darüber hinaus Open-Source und bietet einen professionellen Support. Entwickelt wurde Puppet in Ruby. Entsprechend ist das Command-Line Interface an Ruby angelehnt, was natürlich den Nachteil mitsicht bringt, dass neben den Puppet Befehlen auch noch Ruby gelernt werden muss. Wie bei Salt muss es einen Master geben, auf dem der Puppet-Daemon (Puppetmaster) installiert ist. Der Puppetmaster hält die Zustandsbeschreibung für die jeweiligen Clients und verteilt diese auf Anfrage via REST-API. Die Clients selbst benötigen ebenfalls einen Agenten (Puppet-Agent), um die Zustandsbeschreibungen zu erfragen. Dieser vergleicht die Zustandsbeschreibung mit der aktuellen Konfiguration des Clients und nimmt entsprechende Änderungen vor (Rhett (2015)).

4.2.4 Zusammenfassung

Die wichtigsten Faktoren bezüglich der zu entwickelnde Anwendung zusammengefasst.

	Vagrant	Saltstack	Puppet
Host-Betriebssystem	Diverse Linux Distributionen	Diverse Linux Distributionen	Linux Distributionen, Windows
Client-Betriebssysteme	Diverse Linux Distributionen, Windows	Diverse Linux Distributionen, Windows	Linux Distributionen, Windows
Lokale Client Installation nötig	Nein	Ja	Ja
Command-line Interface Sprache	Python	Python	Ruby
Zustandsbeschreibung	YAML	YAML	Puppet Code
Push oder Pull	Push	Push	Push und Pull
Open-Source	Ja	Ja	Ja
Preis	Kostenlos	Kostenlos	Kostenlos

4.2.5 Fazit

Der größte Unterschied in den verglichenen Konfigurationsmanagement-Systemen besteht im Kontext, in dem das jeweilige System eingesetzt werden soll. Alle drei Anwendungen erfüllen die Grundbedingung, auf Linux lauffähig zu sein und diverse Linux-Distributionen als Client bespielen zu können, wobei Puppet sich hervorragend im Umfeld größerer Systemlandschaften eignet, d.h. wenn es darum geht, mehrere Clients gleichzeitig zu verwalten und zu konfigurieren. Wie Puppet benötigt Staltstack für die Clients extra Software, um die gewünschte Zustandsbeschreibung durchzuführen, was einen gewissen Zeitaufwand und eine Fehlerquelle mehr bedeutet. Ansible ist zwar der unbekanntere Vertreter unter den Konfigurationsmanagement-Systemen, kann aber leicht eingerichtet werden und benötigt keine weitere Ergänzungen auf der Clientseite. Dadurch wird auf den zu erstellenden Maschinen, ausschließlich das Gewünschte installiert, der gesamte Installationsprozess zudem beschleunigt und der erforderliche Zeitaufwand minimiert.

4.3 Vagrant

Ein weiteres Produkt, das die zu erstellende Applikation unterstützen könnte, ist Vagrant. Dabei handelt es sich um ein Softwareprojekt, welches 2010 von Mitchell Hashimoto und

John Bender 2010 ins Leben gerufen wurde. Vagrant ist ein Entwicklungswerkzeug, das als Wrapper zwischen Virtualisierungssoftware wie VirtualBox und Konfigurationsmanagement-System fungiert. Das command-line Interface und die einfache Konfigurationssprache helfen, virtuelle Maschinen schnell zu konfigurieren und zu verwalten. Die Konfiguration einer Maschine geschieht über das 'Vagrantfile', in dem Parameter wie IP-Adresse konfiguriert und Konfigurationsmanagement-Systeme hinzugeschaltet werden können. Da das Vagrantfile in einer Ruby Domain Specific Language geschrieben wird, bedeutet das für den Anwender, ein problemloses Teilen des Vagrantfiles mit anderen Kollegen über Versionskontrollen (z.B. Git oder Subversion).
In puncto Konfigurationsmanagement-Systeme, wird dem Benutzer die Freiheit gegeben, auf eine Vielzahl bekannter Konfigurationsmanagement-Systeme zurückzugreifen, unter anderem auch Ansible, Salt und Puppet.
Um die Virtualisierung vorzunehmen, greift Vagrant standardmäßig auf VirtualBox zurück. Allerdings werden auch Amazon-Web-Services und VMware-Fusion unterstützt und Teamarbeit ermöglicht durch eine 'sharing'-Option, die mit Version 1.5 implementiert wurde. Das Teilen einer Maschine ermöglicht es Teams, an gemeinsamen und entfernten Standorten auf die gleiche Maschine zuzugreifen (Peacock (2013)). Gerade die zusätzlichen Funktionen von Vagrant und das Vereinen von Konfigurationsmanagement und Virtualisierer in einem Produkt würde den Entwicklungsprozess der Applikation unterstützen helfen. Funktionen, wie das Teilen einer Maschine, könnten direkt übernommen und in die Applikation eingefügt werden.

5 Softwareentwurf

Nach Balzert (2011) ist der Softwareentwurf die Entwicklung einer software-technischen Lösung im Sinne einer Softwarearchitektur auf Basis der gegebenen Anforderungen an ein Softwareprodukt. Die Kunst bei einem Softwareentwurf besteht entsprechend darin, eine Softwarearchitektur zu entwerfen, die die zuvor erarbeiteten funktionalen (Kapitel 3.3) und nichtfunktionalen Anforderungen (Kapitel 3.5) betrachtet einschließlich der Berücksichtigung von Einflussfaktoren, wie definierte Randbedingungen (Kaptitel 3.6). Der Softwareentwurf ist als Richtlinie zu sehen, der bei der Umsetzung der geforderten Software unterstützt. Die zu erstellende Softwarearchitektur hingegen beschreibt Architekturbausteine, deren Interaktionen und Beziehungen untereinander sowie ggf. deren Verteilung auf physischer Ebene. Dabei ist die Spezifizierung der entsprechenden Schnittstellen der einzelnen Architekturbausteine mit zu beachten. Zur Visualisierung können verschiedene Abstufungen von Sichten herangezogen werden, die Kontextabgrenzung, Bausteinsicht, Laufzeitsicht und die Verteilungssicht.

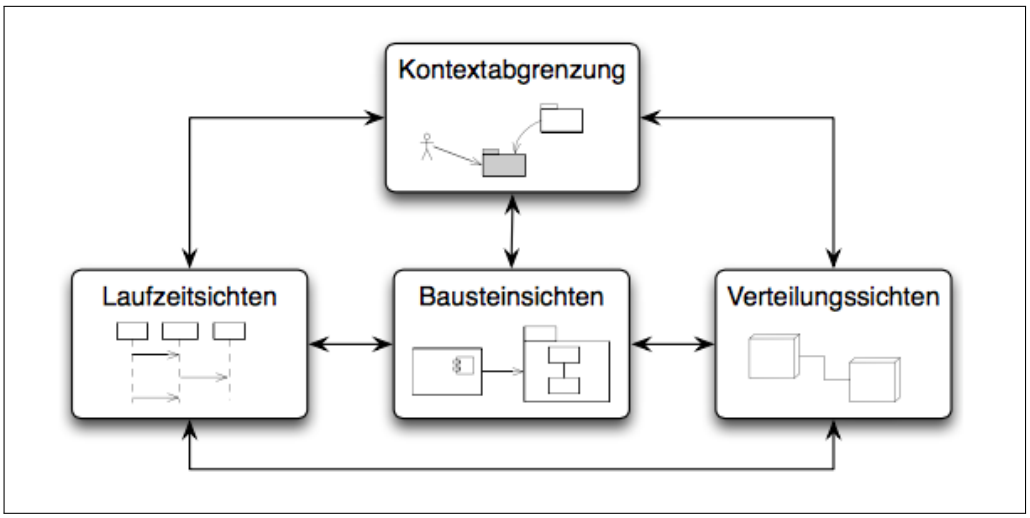


Abbildung 5.1: Vier Arten von Sichten (Starke (2014))

5.1 Kontextabgrenzung

Die Kontextabgrenzung beschreibt die Einbettung des Systems in seine Umgebung sowie die wesentlichen Teile der umgebenden Infrastruktur. Die ermittelten Anforderungen aus Kapitel 3.3 und **??** haben ergeben, dass die Hauptfunktionalitäten aus Erstellen, Exportieren, Importieren, Teilen und dem Provisioning von virtuellen Maschinen bestehen. Um dies weiter zu bündeln, können Teile der Hauptfunktionaliäten bestimmter Produkte übernommen werden, die in Kapitel 4. betrachtet wurden. VirtualBox kann das Erstellen, Exportieren und den Import von virtuellen Maschinen übernehmen. Das Konfigurationsmanagement-System Ansible ist darauf ausgelegt, mit bekannten Virtualisierungslösungen zusammen arbeiten zu können und übernimmt somit die gewünschte Anforderung nach automatisierter Softwareinstallation. Um die beiden Anwendungen zu vereinen, wird als Wrapper Vagrant eingesetzt, der zusätzliche Funktionen, wie das Teilen (Sharen) einer Maschine, mitbringt. Der Kern der Anwendung ist als eine weitere Schicht anzusehen, die die drei Softwareprodukte noch mehr vereint und dem Anwender eine entsprechende Benutzeroberfläche zur Verfügung stellt.

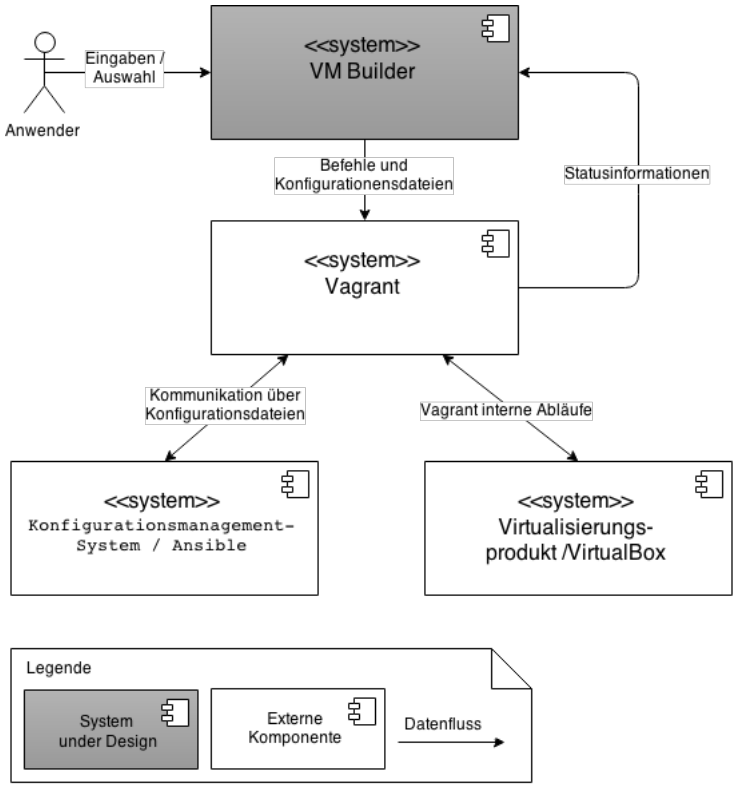


Abbildung 5.2: Kontextsicht

5.1.0.1 Kurzbeschreibung der externen Schnittstellen

Eingaben / Auswahl	Der Anwender tätigt Eingaben und wählt unter bereitgestellten Optionen aus. Diese werden direkt von der Applikation verarbeitet
Befehle und Konfigurationen	Die Applikation erstellt nötige Konfigurationsdateien für Vagrant und Ansible und leitet Befehle für die Weiterverarbeitung an Vagrant weiter
Kommunikation über Konfigurationsdateien	Vagrant ruft über die erstellten Konfigurationsdateien den Konfigurationsmanager Ansible auf, um die virtuelle Maschine in den beschriebenen Zustand zu überführen
Vagrant interne Abläufe	VirtualBox erstellt die virtuelle Maschine und gibt Statusmeldungen an Vagrant weiter. Dies sind interne Abläufe, die zwischen Vagrant und VirtualBox ablaufen und nicht vom Entwicklungsprozess beeinflusst werden können

Die Applikation selbst, inklusive der oben genannten Produkte, läuft auf einem Server, der zentralisiert positioniert ist und entsprechend angesprochen werden kann. Dem Anwender wird von der Applikation eine Weboberfläche angeboten, die es ermöglicht Eingaben zu tätigen und Optionen auszuwählen, um eine virtuelle Maschine zu erstellen oder zu verwalten. Um die Applikation auf dem Server zu betreiben, muss eine Internetverbindung bestehen, die es Vagrant ermöglicht, das gewünschte Abbild des Betriebssystems herunterzuladen und die virtuelle Maschine mit anderen Anwendern zu teilen. Die vom Anwender gestellten Anfragen an den VM-Builder, werden in Konfigurationsdateien übersetzt, die speziell für Vagrant und Ansible passend erstellt werden. Diese Konfigurationsdateien dienen nicht nur zur Erstellung der gewünschten virtuellen Maschine, sondern auch zur Kommunikation zwischen Vagrant und Ansible. Vagrant entnimmt den Konfigurationen das gewünschte Image und leitet den entsprechenden Download des Betriebssystems ein. Allerdings nur, falls das Image des Betriebssystems nicht schon auf dem Server vorliegt.
Durch die Hilfe von VirtualBox und ggf Ansible, wird die zu erwartende virtuelle Maschine komplettiert.

5.2 Verteilungssicht

Um die Beschreibung aus 5.1.0.1 der Kontextabgrenzung visuell zu unterstützen, wird in diesem Fall die Verteilungssicht herangezogen. In Starke (2011) wird die Verteilungssicht als '[...] die Verteilung von Systembestandteilen auf Hard- und Softwareumgebungen' beschrieben, die '[...] klärt, welche Teile des Systems auf welchen Rechnern, an welchen geographischen Standorten oder in welchen Umgebungen ablaufen können, wenn es in einer konkreten technischen Infrastruktur installiert wird'.

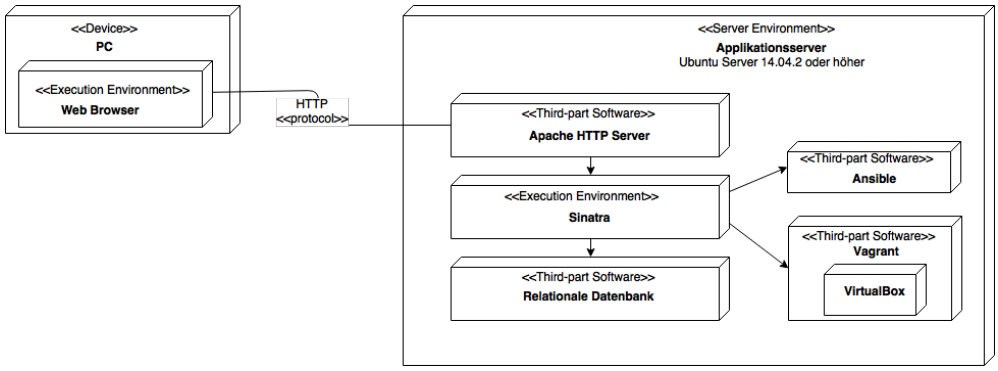


Abbildung 5.3: Verteilungssicht des VM-Builders

Die genaue Platzierung der Softwarekomponenten aus Abschnitt 5.1.0.1 verhilft zu einer besseren Umsetzung der kompletten Softwarestruktur und der Kommunikation der einzelnen Komponenten. In Kapitel 3 unter den Abschnitten 3.5 und 3.6.1 wurden technologische Anforderungen und entsprechende Randbedingungen beschrieben und definiert. Die dortige Beschreibung besagt, dass die zu verwendene Software frei verfügbar und kostenlos sein sollte. Ausserdem soll die Leistung des Servers den Anforderungen entsprechen. Aus diesem Grunde, wird bei dem Betrieb des Servers auf Ubuntu zurückgegriffen, wobei hier von aktuellen Version (Stand Juni 2015) ausgegangen wird. Als Webserver kann Apache eingesetzt werden, der mittels dem zustandslosen HTTP Protokoll mit dem Client kommuniziert. Die benötigte rationale Datenbank kann z.B durch MySQL umgesetzt werden. Sie wird für das Speichern von Konfigurationen einzelner virtueller Maschinen genutzt und für das zu Verfügungstellung der zu installierenden Softwarekomponenten. Apache und MySQL sind lizensfreie und kostenlose Produkte, die etabliert und leistungsfähig sind. In dem Execution Framwork wird die hier zu planende Anwendung ausgeführt, die wiederum an die Drittanbieter Ansible und Vagrant angebunden ist. Wie bereits in Kapitel 4. aufgeführt, sind beide Produkte ebenfalls kostenlos

und frei verfügbar. Da Hardwarespezifikationen in dieser Phase noch rein spekulativ sind, wird dieser Aspekt in der Umsetzung weiter betrachtet.

5.3 Systemarchitektur

Nachdem die Umsysteme, deren Verteilung und der technische Aufbau der Applikation konzipiert ist, steht die Struktur der Anwendung im Fokus. Dafür gibt es etablierte Architektur- und Entwurfsmuster, auf die zurückgegriffen werden kann. Diese helfen bei der Verteilung der Verantwortlichkeiten und bilden eine Vorlage für die Systemstrukturen. Da sie sich in einigen Punkten überschneiden und sogar ergänzen, ensteht bei der Verwendung mehrerer Architekturmuster keine Widersprüche.

5.3.1 Client-Server-Modell

Um eine möglichst gute strukturelle Verteilung der Aufgaben und Dienstleistungen des VM-Builders auf physikalischer Ebene zu erhalten, wird auf das Client-Server-Modell zurückgegriffen. Das Client-Server-Modell verdeutlicht die Aufgabenverteilung innerhalb einer Applikation und die Zentralisierung von Prozessorenleistung und gemeinsamer Dienste (Schäfer (2009)). Die klare Unterscheidung zwischen den Client- und Servertätigkeiten kann durch einen geschichteten Architekturstil erreicht werden, der die Aufgaben in folgende Schichten unterteilt:

1. Benutzerschnittstelle (User interface)
   Die Benutzerschnittstelle enthält alles Erforderlich, um direkt mit dem Anwender zu interagieren

2. Verarbeitungsebene (Application)
   Die Verarbeitungsebene enthält die Anwendung / Kernfunktionalität

3. Datenebene (Database)
   Daten werden unabhängig von der Applikation persistent gespeichert

Im Zusammenhang mit dem Client-Server-Modell steht das n-Tier Model oder auch Schichtenmodell. Eine Schicht ist entweder ein physikalischer Rechner oder mindestens ein Systemprozess, der eine gewisse Aufgabe übernimmt. Die einfachste Anordnung dieser Schichten besteht darin, sie auf zwei Computer zu verteilen (2-Tier Model), den Client und den Server. Dabei können die Schichten wie in Abbildung 5.4 zwischen Client und Server verteilt sein (Tanenbaum und van Steen (2007)).

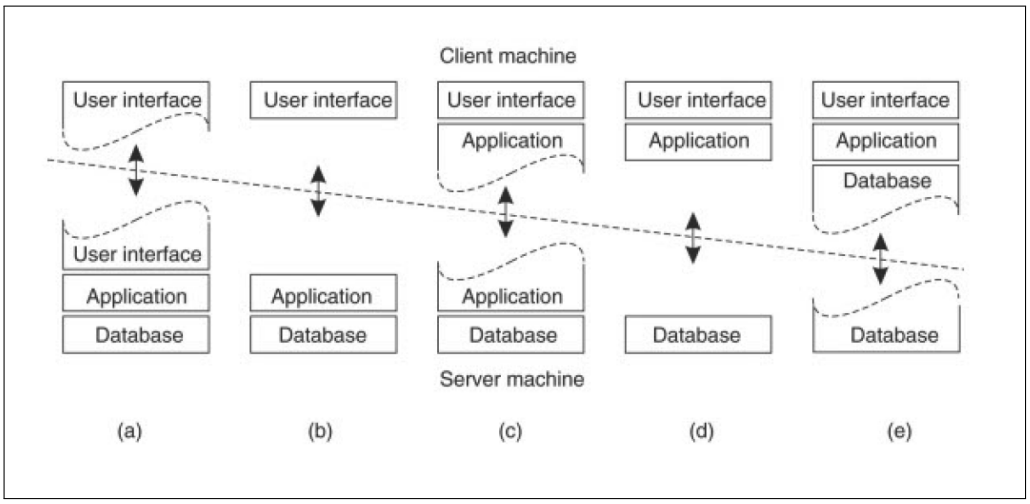


Abbildung 5.4: Client-Server-Anordnungen (Tanenbaum und van Steen (2007))

Die in Abbildung 5.4(a) bis (c) dargestellten Varianten gehören zu der Kategorie Thin-Clients, die für den zu entwickelnden 'VM Builder' im Fokus stehen. Die Varianten (d) und (e) sind sogenannte Fat-Clients.
Der Vorteil von Thin Clients ist, dass weniger bis keine Client-Software auf die Seite des Anwenders gebracht werden muss. Denn Software auf Clientseite ist nicht nur schwerer zu administrieren, sondern auch anfälliger für Fehler (Tanenbaum und van Steen (2007)).
Die hier angestrebte Client-Server-Anordnung ist die in Abbildung 5.4(a) gezeigte Variante. Der Client soll so schlank wie möglich gehalten werden, um dem Anwender einen schnellen Seitenaufbau zu ermöglichen und lokale Installationen zu verhindern. Variante (a) würde nur dann zum Einsatz kommen, wenn eine entfernte Steuerung der Darstellung gewünscht wäre, was hier nicht der Fall ist. Beim Client-Server-Aufbau mit Variante (b) geht man davon aus, dass Programmlogik mit zum Client übertragen wird, also z.B. eine Eingabeüberprüfung direkt im Frontend. Da in der Anforderungsanalyse Kapitel 3.6.1 festgelegt wurde, dass mit Ruby inkl. des Frameworks Sinatra gearbeitet werden soll, wird die Logik auf der Server-Seite implementiert und benötigt keinen Anteil auf Client-Seite. Entsprechend sind die Verarbeituns- (Application) und die Datenebene (Database) auf der Server-Seite angesiedelt. Auf Verarbeitungsebene wird der Webserver mit den Applikationskomponenten des VM-Builders realisiert. Die Datenebene spiegelt die zu verwendende Datenbank wieder, die für die Persistierung von Daten zuständig sein wird.

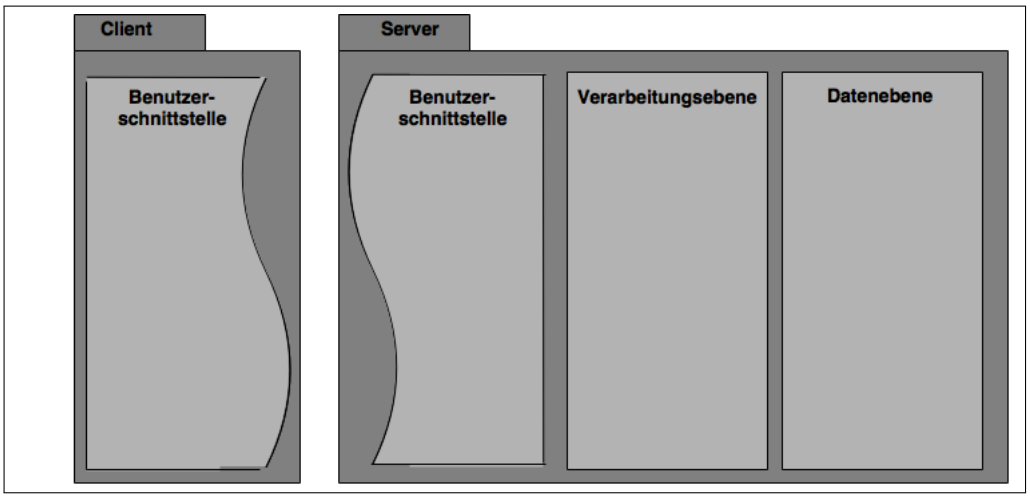


Abbildung 5.5: Model-View-Controller (Wikipedia (2015))

5.3.2 Model-View-Controller Entwurfsmuster

Für eine strukturierte Umsetzung der grafischen Komponente des 'VM Builders', wird auf das bekannte architektonische Model-View-Controller Entwufsmuster zurückgegriffen. Das Model-View-Controller (MVC) Entwurfsmuster findet beim Entwurf grafischer Benutzungsoberfläche anwendung, d.h. bei der Mensch-Maschine-Interaktionen. Dazu wird das System, das Interaktionen anbietet und ausführt, in drei Verantwortlichkeiten strukturiert:

1. **Model**
   kapselt alle fachlichen Daten und enthält den Anwendungskern

2. **View**
   bereitet Informationen für den Anwender grafisch auf

3. **Controller**
   nehmen Benutzereingaben/Events an, die entsprechend an das passende Model oder die View weitergeleitet werden.

Wie das Client-Server-Model, besteht auch das Model-View-Controller Entwurfsmuster aus drei Schichten. Anstatt diese drei Schichten auf mehrere physikalische oder virtuelle Systeme zu verteilen, werden sie auf Applikationsebene abgebildet. Das MVC-Entwurfsmuster kann zwar auf jeder der Schichten des Client-Server-Models implementiert werden, hat aber im Gegensatz zu dem Client-Server-Model entsprechend nichts mit der Verteilung des Systems zu tun.

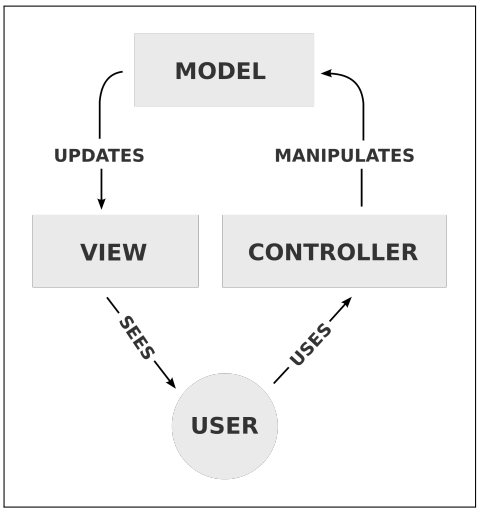


Abbildung 5.6: Model-View-Controller (Wikipedia (2015))

Abbildung 5.6 verdeutlicht die Kommunikation zwischen den Bestandteilen des Mode-View-Controller Entwurfsmusters.

1. Alle Eingaben/Änderungen des **User** werden von der Benutzeroberfäche an den Controller weitergegeben
2. Der **Controller** gibt Zustandsänderungen an das Model weiter
3. Das **Model** verarbeitet die erhaltenden Daten in dem diese z.B an den persistenten Speicher weitergeleitet werden oder Berechnungen stattfinden
4. Die resultierenden Ergebnisse / Änderungen werden dann über die View sichtbar gemacht

Das MVC-Entwurfsmuster entkoppelt das User-Interface von der Verarbeitungsebene. Es geht der Forderung nach, die View (Repräsentation) und das Model (Fachlichkeiten) zu trennen, da die Änderungshäufigkeit beider Ebenen, unterschiedlich ausfallen können. Durchschnittlich ändern sich z.B. Windows-Oberflächen etwa alle zwei Jahre, Fachlichkeit aber sehr viel langsamer in der Größenordnung von ca 10 bis 15 Jahren. Das MVC-Entwurfsmuster erlaubt also als eine Modernisierungsmaßnahme den entsprechenden Austausch der Oberfläche, ohne die Fachlichkeiten ändern zu müssen (Masak (2009)). Da das geforderte Framework Sinatra diese Konzept nativ umsetzen kann, lässt sich das Entwurfsmuster entsprechend gut anwenden. In Sinatra werden die Views separiert von den sogenannten Rounten (Controllern) implemen-

tiert, wodurch die 'V' und 'C' Eigenschaften erfüllt werden. Um die Modell-Anforderungen zu erfüllen, kann z.B. ein Datenbank-Framework, wie Active-Record angewendet werden.

Abbildung 5.7 zeigt die Integration des MVC-Entwurfsmuster in das Client-Server-Modell.

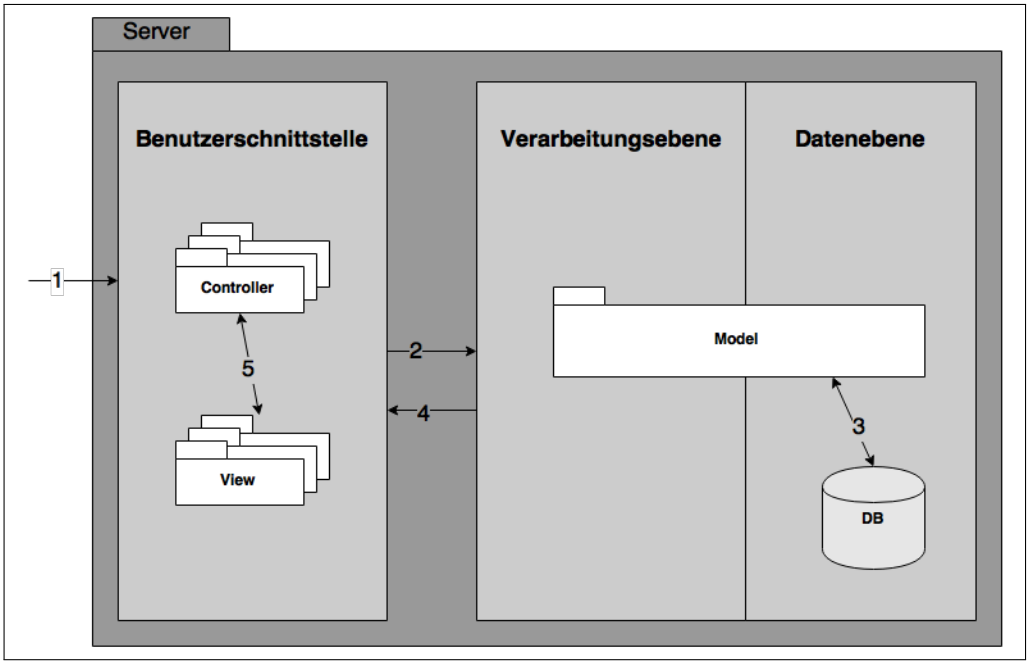


Abbildung 5.7: Model-View-Controller

Da der Client nur für die Darstellung zuständig ist, wird wie in Abbildung 5.7 zu sehen, das MVC-Entwurfsmuster nur auf Server-Seite implementiert. Entsprechend werden die Controller und die Views im serverseitigen Teil der Benutzerschnittstelle angesiedelt, während die Modells sich über die Verarbeitungsebene und Datenebene erstrecken, um den Zugriff auf die Datenhaltung zu gewährleisten.

5.4 Kommunikation

Die Kommunikation zwschen Client und Server wird über das zustandslose HTTP Protokoll realisiert. Möchte ein Client (der theoretisch ein Web-Browser, eine Web-Anwendung, ein Dienst, usw. sein kann) mit einem Webserver kommunizieren, erstellt der Client eine HTTP-Nachricht. Diese Nachricht ist in 'plain-text' geschrieben und Zeilen orientiert. Dementsprechend ist eine Nachricht leicht zu erstellen und auf Serverseite entsprechen leicht auszuwerten. Ist ein Server mit der Anfragebearbeitung fertig, sendet er in der Regel den Status (war die Client-Anforderung erfolgreich, ist ein Fehler aufgetreten, etc.), Inhalte und andere Daten zurück

an den Client. Nachrichten enthalten sogenannte HTTP-Verben, die den Typ der Anfrage definieren und wie der Server die Anfrage zu verstehen hat (Harris und Haase (2011)).

Standard HTTP-Verben	**Definition**
GET	Eine GET-Anforderung wird verwendet, um einen Server zu bitten, die Darstellung einer Ressource zurückzuliefern
POST	Eine POST-Anforderung wird verwendet, um Daten an einen Webserver zu übermitteln
PUT	PUT wird verwendet, um auf einem Server eine Ressource zu erstellen oder zu aktualisieren
DELETE	DELETE wird verwendet, um eine Ressource auf dem Server zu löschen

In dem zu verwendenen Framework Sinatra wird das Vokabular der HTTP-Verben benutzt, um Routen zu definieren. Um eine Route in Sinatra zu deklarieren, muss das HTTP-Verb zum Reagieren geliefert, die spezifische URL und dann das gegebenenfalls für die Route gewünschte Verhalten definiert werden, siehe Abbildung 5.1.

```
1  get '/' do
2    .. zeige etwas ..
3  end
4
5  post '/' do
6    .. erstelle etwas ..
7  end
8
9  put '/' do
10   .. update etwas ..
11 end
12
13 delete '/' do
```

```
14   .. entferne etwas ..
15 end
16
17 options '/' do
18   .. zeige, was wir können ..
19 end
20
21 link '/' do
22   .. verbinde etwas ..
23 end
24
25 unlink '/' do
26   .. trenne etwas ..
27 end
```

Listing 5.1: Routen in Sinatra (Sinatra (2015))

5.5 Server

Wie in Kapitel 5.3 beschrieben, wird der Server nach den dort geplanten Entwurfsmuster konstruiert. Damit ein vollständiges Bild des Serverkonstrukt entsteht, werden im Folgenden die noch offenen Sichten aus der Abbildung 5.1 entworfen.

5.5.1 Bausteinsicht

Die Bausteinsicht 'zeigt die statische Struktur des Systems, seinen Aufbau aus Softwarebausteinen sowie deren Beziehungen und Schnittstellen untereinander' Starke (2011). Ausgehend vom Systemkontext, wird das System hierarchisch zergliedert. Somit können Ebenen in unterschiedlichem Detailgrad entstehen. Ebene-0 stellt das System als Blackbox dar, die der Kontextsicht aus Abbildung 5.2 entspricht. In den höheren Ebenen, wird der Detailgrad erhöht und die dargestellte Blackbox zur Whitebox. Whiteboxen geben detailliertere Einblicke in ihre Struktur und Schnittstellen, die allerdings wiederum als Blackboxen dargestellt werden. Abbildung 5.8 ist eine Ebene-1 Darstellung der Serverapplikation in Form des MVC-Entwurfsmusters, in der die Anwendung mit ihren einzelnen Komponenten betrachtet wird, die als Blackboxen dargestellt sind.

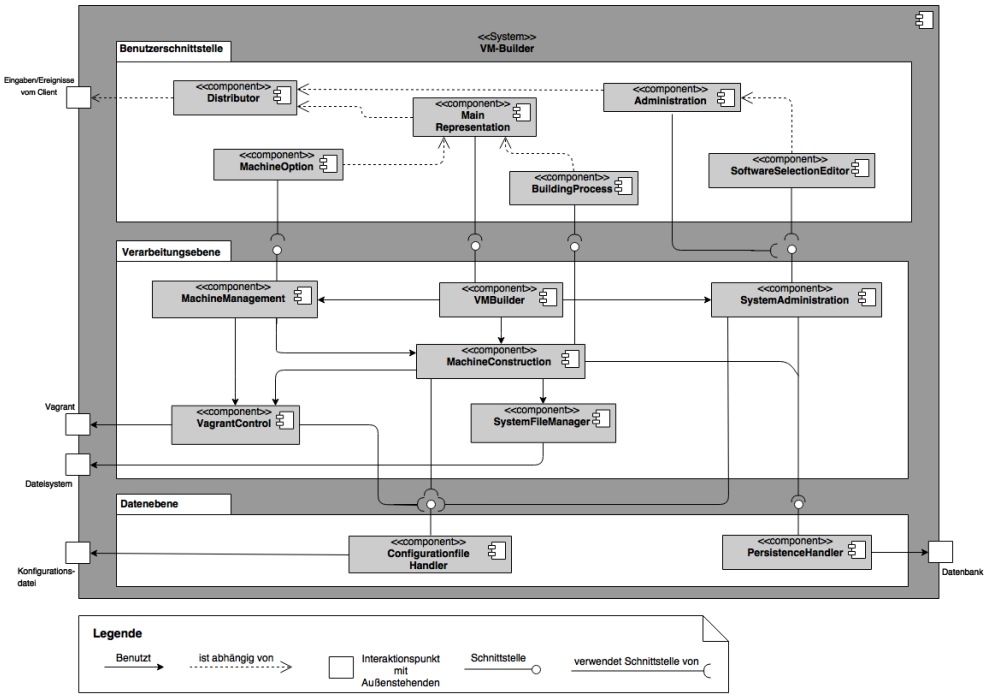


Abbildung 5.8: Bausteinsicht Level 1

Die Komponenten sind in ihre Zuständigkeiten gegliedert und den entsprechenden Schichten zugeordnet. Die Schnittstellen werden durch Interaktionspunkte zwischen den Schichten dargestellt. Jede Komponente enthält, der jeweiligen Schicht entsprechend, Controller, Views und/oder Models, die in dem folgenden Abschnitt detaillierter betrachtet werden.

5.5.1.1 Benutzerschnittstelle

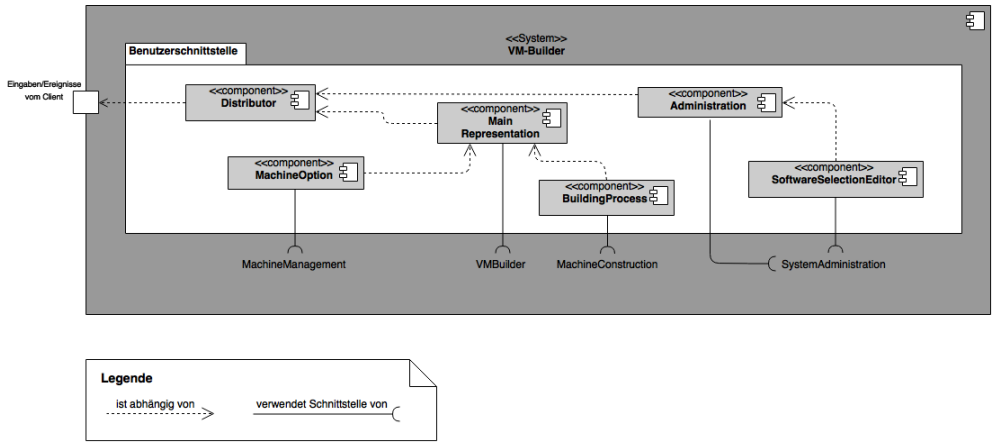


Abbildung 5.9: Benutzerschnittstelle

Der für die Kommunikation zuständige Anwendungsteil, ist die Benutzerschnittstelle (Abbildung 5.9). Die dort enthaltenen Komponenten sind für das Annehmen der User-Interaktionen zuständig und für die Repräsentation der gewünschten Inhalte.

Distributor - Komponente

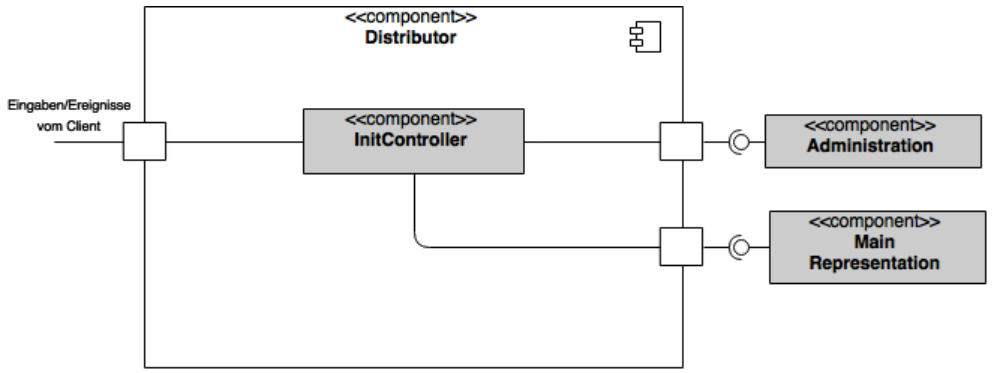


Abbildung 5.10: Komponentensicht VMBuilder

Die primäre Zuständigkeit der **Distributor** - Komponente liegt im Empfang der eingehenden Kommunikation und der Weiterleitung an den entsprechenden Controller. Da die Administration des VMBuilders eine gänzlich andere Hauptfunktion ist, als der Aufbau inklusive der Verwaltung von virtuellen Maschinen, entsteht durch den Einsatz dieser Komponente eine klare hierarchische Unterteilung der Hauptfunktionalitäten. Zudem erleichtert der Aufbau eine Erweiterung von neuen primären Funktionen der Applikation.

MainRepresentation - Komponente

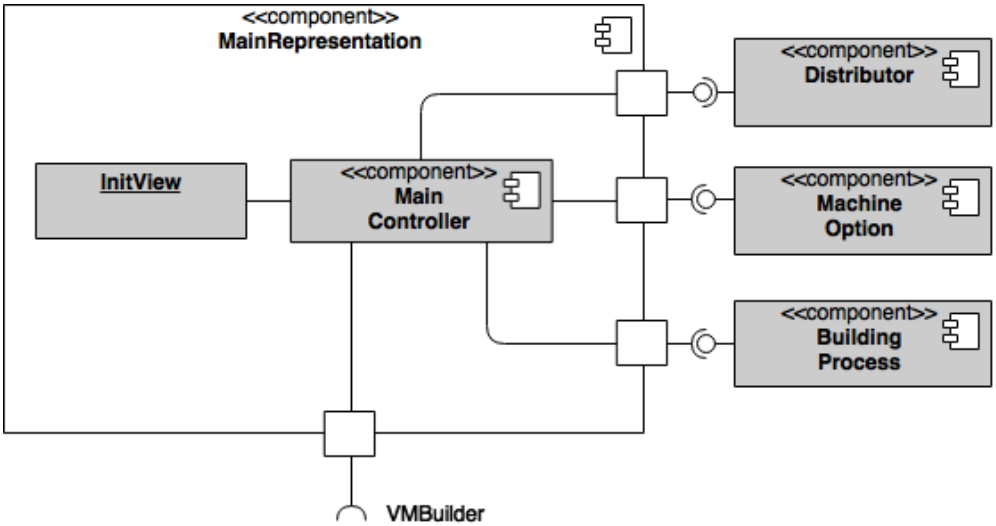


Abbildung 5.11: Komponentensicht VMBuilder

Im Inneren der **MainRepresentation** - Komponente ist der **MainController** platziert, der mit Hilfe der **InitView** eine Übersicht der bestehende virtuelle Maschinen verschafft. Um die Übersicht zu erhalten, muss die Schnittstelle zum **VMBuilder** der Verarbeitungsebene bestehen. Erst sie stellt die Daten für die Anzeige der vorhandenen Maschinen zur Verfügung. Aus dieser Ansicht kann ausserdem der Aufbauprozess einer neuen Maschine über die **BuildingProcess** - Komponente initialisiert, oder Optionen auf einzelne virtuelle Maschinen aufgerufen werden. Die verfügbaren Optionen werden durch die **MachineOption** - Komponente gesteuert und realisiert.

BuildingProcess - Komponente

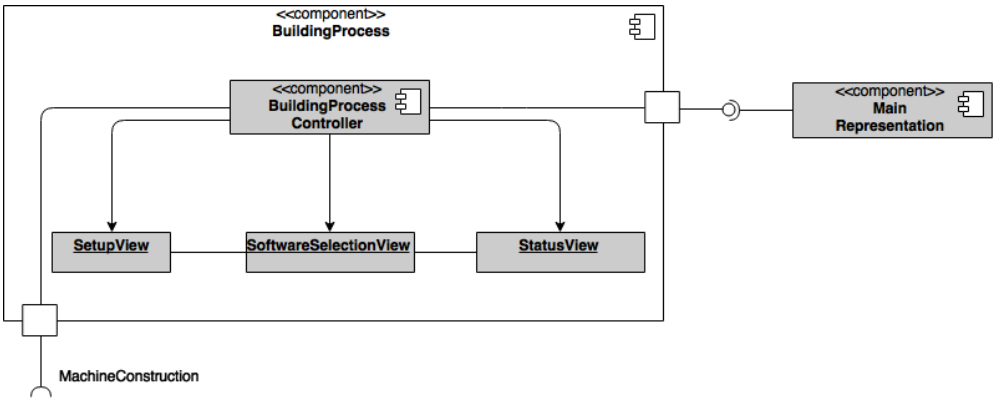


Abbildung 5.12: Komponentensicht BuildingProcess

Der Aufbauprozess einer virtuellen Maschine wird grafisch durch die **BuildingProcess** - Komponente realisiert. Die dort enthaltenen Views leiten den Anwender durch den Aufbauprozess, wobei jede View einem Konfigurationsschritt entspricht:

1. **SetupView**
   In der SetupView werden die Eigenschaften wie IP-Adresse und Name der zu erstellenden Maschine festgelegt

2. **SoftwareSelectionView**
   Um die virtuelle Maschine in den gewüschten Zustand zu versetzen, bietet die SoftwareSelectionView dem Anwender eine Auswahl, an zu installierender Softwarekomponenten und Paketen.

3. **StatusView**
   Die StatusView erstellt eine Übersicht über den aktuellen Aufbauverlauf und präsentiert die Zugangsmöglichkeiten zur virtuellen Maschine

Unterstützt wird der Aufbau durch den Zugriff auf die **MachineConstruction** - Komponente der Verarbeitungsebene. Dieser gibt unter anderem der **SoftwareSelectionView** eine Vorgabe an Auswahloptionen, die der Anwender in Betracht ziehen kann. Da der Aufbau einer virtuellen Maschine erst durch einen bestimmten Aufruf aus der MainRepresentation erfolgen soll, ist der **BuildingProcessController** auch nur über diese Komponente aufrufbar.

MachineOption - Komponente

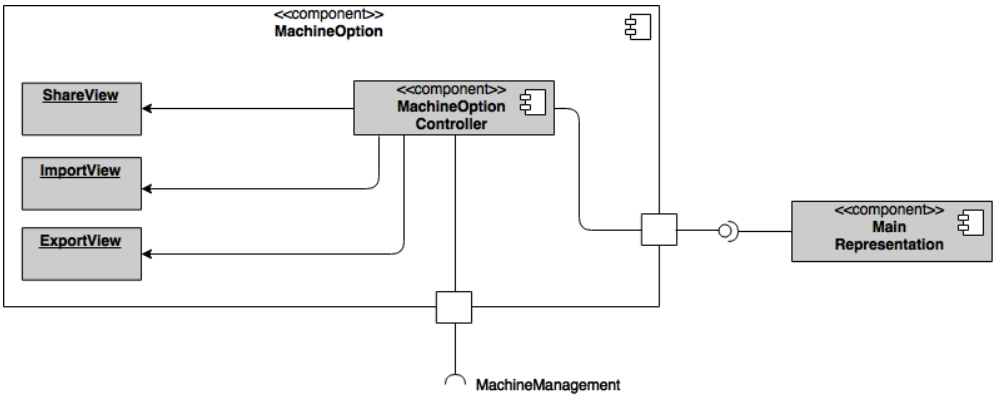


Abbildung 5.13: Komponentensicht MachineOption

In Abbildung 5.13 ist ersichtlich, dass die **MachineOption** - Komponente drei Views bereitstellt, die alle über den Controller indirekt in Beziehung stehen. Sie repräsentieren die Optionen, die ein Anwender auf eine bestehende Maschine ausführen kann. Über die Schnittstelle der **MachineManagement** - Komponente, werden die einzelnen Funktionalitäten bereitgestellt und ausgeführt.

Administration - Komponente

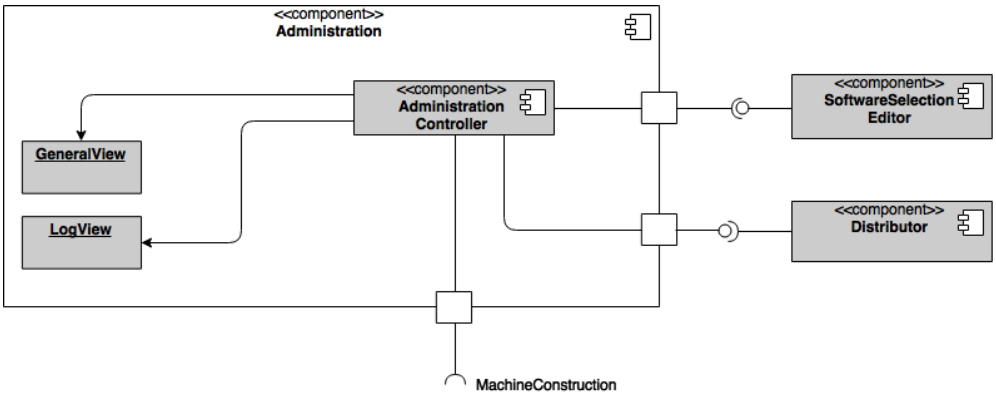


Abbildung 5.14: Komponentensicht Administration

Wie bereits am Anfang im Abschnitt **Distributor** - Komponente (5.5.1.1) erwähnt, gibt es neben der **MainRepresentation** - Komponente auch die **Administration** - Komponente. Sie ist für die Visualisierung der generellen Einstellungen, der Anzeige von Logdateien und dem Softwareeditor zuständig. Der Softwareeditor ist als eingenständige Komponente ange-

schlossen, da dieser spezielle Funktionen bereitstellt. Die Schnittstelle der **Administration** - Komponente ist mit dem SystemAdministration aus der Verarbeitungsebene verbunden, um die voreingestellten Anwendungseigenschaften abzurufen und Neue zu speichern.

SoftwareSelectionEditor - Komponente

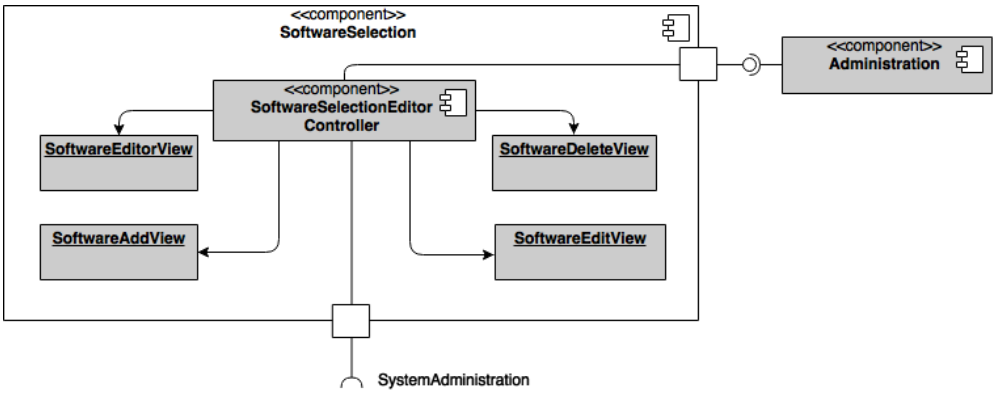


Abbildung 5.15: Komponentensicht SoftwareSelectionEditor

Die an der **Administrations** - Komponente angeschlossene **SoftwareSelectionEditor** - Komponente unterstützt den Anwender in der Konfiguration einzelner Softwarekomponenten und Pakete. Pakete bestehen aus einzelnen Softwarekomponenten, die in Abhängigkeit gestellt werden. So können beim Aufbau einer virtuellen Maschine, durch die Auwahl eines Paketes, mehrere Softwarekomponenten auf einmal installiert werden. Zudem ermöglicht die **SoftwareSelectionEditor** - Komponente neue Softwarekomponenten hinzuzufügen, zu bearbeiten und zu ändern. Entsprechende Views helfen dem Anweder die gewünschten Funktionen durchzuführen.

5.5.1.2 Verarbeitungseben

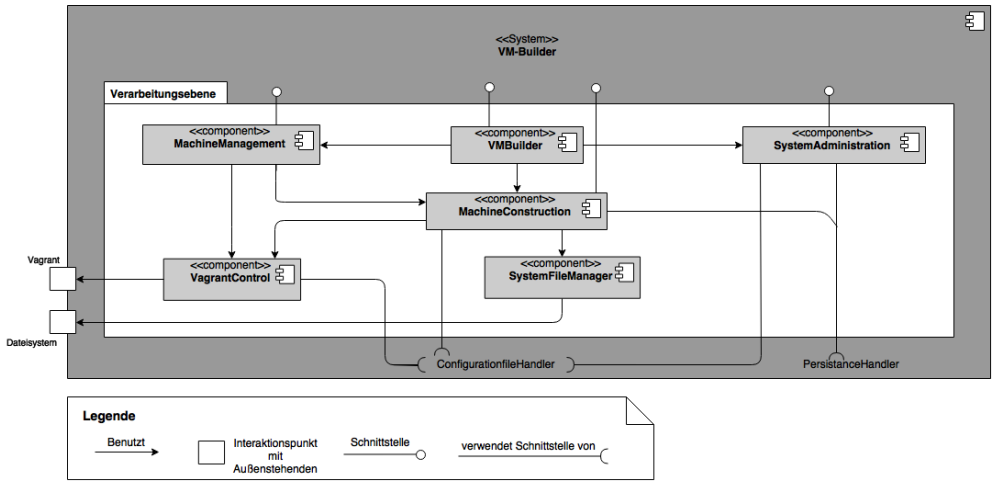


Abbildung 5.16: Ansicht der Verarbeitunsebene

Wie im MVC - Entwurfsmuster vorgesehen, beinhaltet die Verarbeitunsebene die Logik der Anwendung, die durch logisch konstruierte Komponenten repräsentiert wird. Da Komponenten wiederum aus Komponenten bestehen können, werden diese durch eine Whitebox-Darstellung explizit hervorgehoben. Ohne weitere Whitebox-Darstellung kommen die Komponenten aus, die für sich selber stehen.

SystemFileManager - Komponente

Die Zuständigkeit des **SystemFileManagers** besteht im Zugriff auf das Dateisystems. Durch den **SystemFileManager** werden Standard-Ordner Funktionen des Betriebssystems ermöglicht sowie Kopier-, Duplizierungs- und Erstellungsoperationen. Diese Komponente ist gerade im Bezug auf den Erstellungsprozess essenziell.

VagrantControl - Komponente

VagrantControl vereinigt Befehlsaufrufe für die Steuerung von Vagrant. Zudem extrahiert die Komponente essenzielle Informationen aus den Vagrant - Rückmeldungen, die bei der Ausführung von Vagrant erzeugt werden. Die Informationen werden interpretiert und in Rückgabewerte von Funktionsaufrufen umgewandelt, oder für die Generiert von Fehlermeldungen verwendet. Da Vagrant nur mit einem gültigen Vagrantfile lauffähig ist, übernimmt **VagrantControl** auch das Erstellen der genannten Datei.

MachineConstruction - Komponente

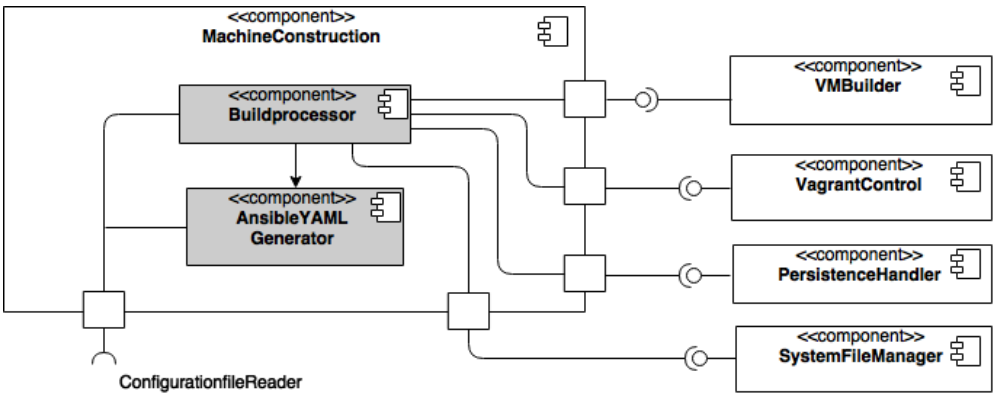


Abbildung 5.17: Komponentenansicht MachineConstruction

Der Aufbau einer virtuellen Maschine wird durch die **MachineConstruction** - Komponente realisiert. Genauer durch das Zusammenspiel zwischen **Buildprocessor** und **AnsibleYAML-Generator**. Eine Zweiteilung der beiden Komponenten ist im Bezug auf ihre Zuständigkeiten wichtig. So ist das Auswechseln der Komponenten und die Erweitern von Funktionalitäten einfacher. Der **AnsibleYAMLGenerator** ist die Komponente, die es Ansible ermöglicht zu wissen, welche Software provisioniert werden soll. Während der Aufgabenbereicht des **Buildprocessor** sich vom Einsammeln der Eingabeinformationen des Benutzers, über das Erstellen der nötigen Konfigurationsdateien bis hin zum starten des eigentlichen Aufbauprozesses einer Maschine erstreckt. Soll zukünftig der Aufbauprozess verändert, oder Ansible durch einen anderen Provisionierer ausgewechselt werden, ermöglicht die Aufteilung der Zuständigkeiten den unkomplizierten austausch der jeweiligen Komponente. Die Schnittstellen der **MachineConstruction** - Komponente sind unter anderem mit **VagrantControl**, dem **PersistanceHandler** und dem **SystemFileManager** verbunden. Um den Aufbau korrekt durchführen zu können, benötigt der **Buildprocessor** die Steuerbefehle für Vagrant und den Zugriff auf das Dateisystem um die virtuelle Maschine zu platzieren. Der **PersistanceHandler** liefert für den Aufbau die entsprechende Softwarevorschläge, die der Benutzer auf der virtuellen Maschine installieren kann. Der Provisionierer 'Ansible' arbeitet mit YAML-Datei, die Konfigurationseigenschaften enthalten, die unterstützend beim Aufbau einer virtuellen Maschine wirken können. Die YAML-Dateien sind optional, da sie die gewünschten Softwarekomponenten beinhalten. Wird keine Software benötigt, kann diese Datei weggelassen werden. Der **AnsibleYAMLGenerator** baut aus den Informationen, die er aus dem **BuildProcessor** erhält, die richtige Struktur und den inhaltlichen Kontext der YAML-Datei. Durch den Zugriff auf die **Configurationfi-**

leHandler - Komponente, die sich in der Datenebene befinden, erhalten Buildprocessor und AnsibleYAMLGenerator Grundkonfigurationseinstellungen, die zum Aufbau benötigt werden.

SystemAdministration - Komponente

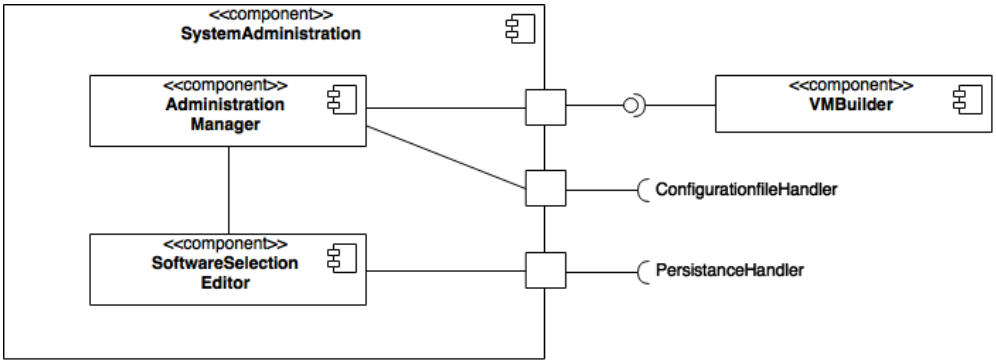


Abbildung 5.18: Komponentenansicht SystemAdministration

Für den administrativen Teil der Applikation ist der **AdministrationManager** zuständig. Der **AdministrationManager** ist die primäre Anlaufstelle für die Konfiguration der Applikation. Die Administration bietet Funktionen zum auslesen von Logdateien und liefert Grundeinstellungen aus dem **ConfigurationfileHandler**. Der **AdministrationManager** soll zudem einen Softwareeditor für Administratoren bereitstellen, der durch die **SoftwareSelectionEditor** - Komponente realisiert ist. Durch Verwendung des **PersistanceHandler** können neue Softwarebestandteile bearbeitet, gelöscht oder hinzugefügt werden. Zu den weiteren Hauptaufgaben gehört das Kreieren von Softwarepaketen. In denen werden Abhängigkeiten zu anderen Softwarebestandteilen geknüpft und optional Scripte hinzugefügt.

MachineManagement - Komponente

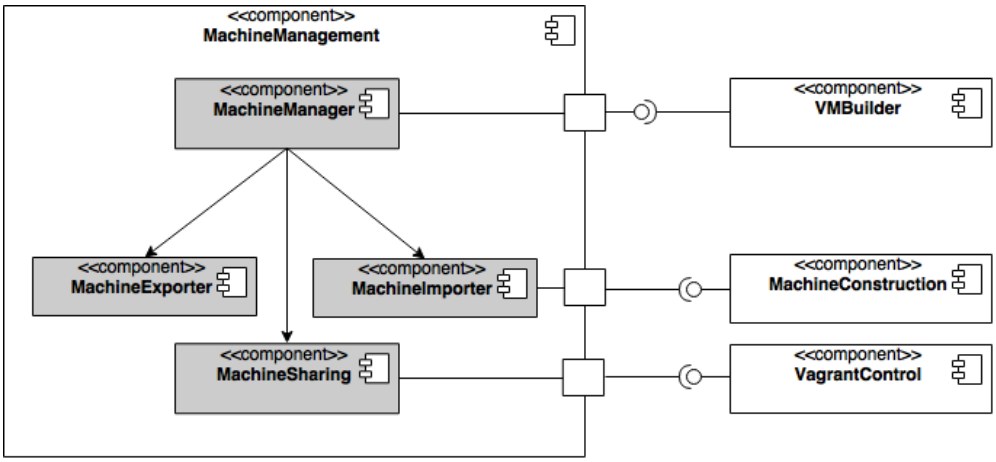


Abbildung 5.19: Komponentenansicht MachineManagement

Um dem Anwender Optionen für die bereits bestehenden virtuellen Maschinen anbieten zu können, gibt es den **MachineManager**. Dieser agiert als Verwalter für Optionen, die auf virtuelle Maschinen ausgeführt werden können. Wie in Abbildung 5.16 werden Optionen wie Export-, Import- und das Sharing von Maschinen angeboten, die durch hinzufügen weiterer Komponenten erweitert werden können. Die **MachineSharing** - Komponente bereitet eine virtuelle Maschine so vor, dass auf sie von überall aus zugegriffen werden kann. Die einzige Beschränkung sind die Richtlinien des Netzwerkes. Zudem ist der **MachineExporter** in der Lage eine virtuelle Maschine zu exportieren, in dem er Konfigurationsdateien packt und dem Anwender zur Verfügung stellt. Diese Dateien können in anderer Virtualisierungsprodukte geladen werden oder durch den **MachineImporter** wieder in die Anwendung importiert werden. Durch die Hinzunahme des **BuildProcessor** kann der Import wieder zu einer virtuellen Maschine aufgebaut werden.

VMBuilder - Komponente

Die zentrale Steuerung des VM-Builders ist die gleichnamige Komponente. Sie liefert nicht nur die Funktionen für die MainPresentation - Komponente aus der Benutzerschnittstelle, sondern ist das Herz der Applikation. (TODO: wird sich noch rausstellen ob das so ist.)

5.5.1.3 Datenebene

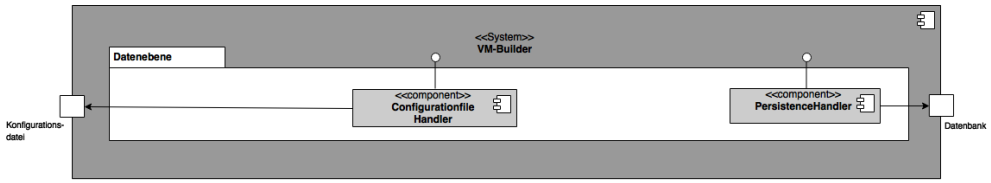


Abbildung 5.20: Ansicht der Datenebene

In der Datenschicht werden Funktionen verankert, die zum direkten Lesen aus dem Datenspeicher verwendet werden. Dies können Funktionen sein, die mittels SQL-Abfragen auf die Datenbank zugreifen oder die lesenden- und/oder schreibenden Zugriff auf Dateien ermöglichen.

PersistanceHandler - Komponente

Der **PersistanceHandler** ist eine der beiden Komponenten in der Datenebene. Durch die dort definierte Funktionen werden kontrollierte Zugriffe auf die Datenhaltung ermöglicht. Manipulation der Daten soll ausschliesslich durch diese Komponente erfolgen. Der **PersistenceHandler** wird somit zur Schnittstelle Richtung Datenbank.

ConfigurationfileHandler - Komponente

Die andere Komponente in der Datenebene ist der **ConfigurationfileHandler**. Durch sie werden Grundeinstellungen der VMBuilder - Applikation aus einer Konfigurationsdatei für das System les- und editierbar. Diese enthält Applikationseinstellungen, Einstellungen für Vagrant und Konfigurationen für Ansible. Der **ConfigurationfileHandler** extrahiert diese drei Einstellungstypen heraus und bereitet sie für den entsprechenden Anwendungszweck auf. So kann z.B der **AnsibleYAMLGenerator** seine benötigten Informationen aus dem **ConfigurationfileHandler** beziehen.

5.5.2 Laufzeitsicht

Nach Zörner (2012) zeigt die Laufzeitsicht Elemente der Bausteinsicht in Aktion, veranschaulicht dynamische Strukturen und das Verhalten des Systems. Um ein paar interessante Aspekte des VM-Builders herauszufiltern, werden auch nur diese in die Laufzeitsicht übernommen. Die dort verwendeten Funktionsaufrufe sind eine abstraktion der späteren Implementierung, da im Entwurf Programmiersprachen unabhängig gearbeitet, sowie auf implementierungsdetails verzichtet wird.

5.5.2.1 Aufbau einer virtuellen Maschine

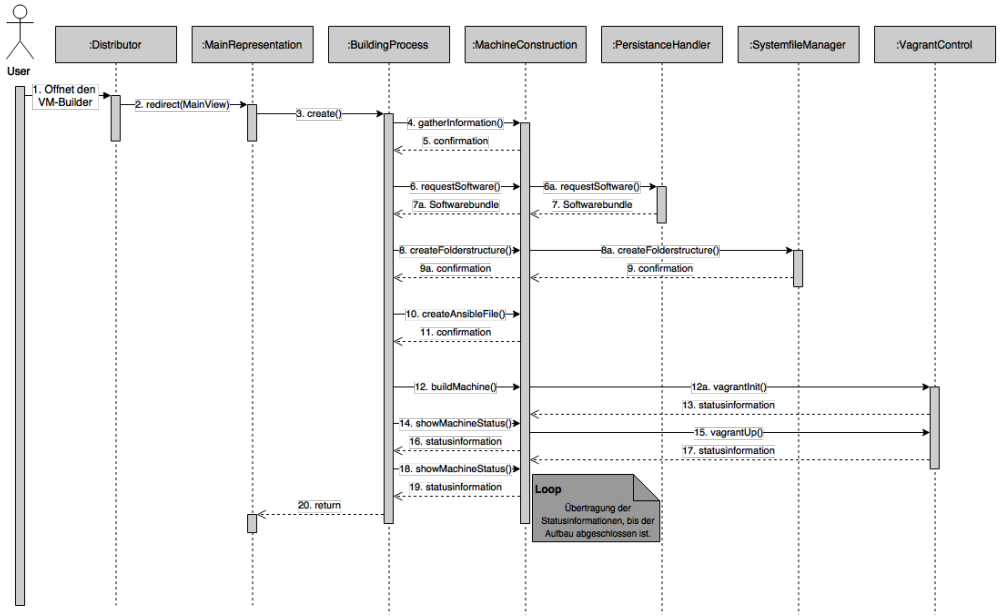


Abbildung 5.21: Laufzeitsicht eines VM-Aufbaus

Der Aufbau einer virtuellen Maschine ist eines der prägnanten Leistungsmerkmale, der hier zu entwerfenden Software. Das Sequenzdiagramm aus Abbildung 5.21 verdeutlicht den Aufbau einer virtuellen Maschine, in dem die Kommunikation zwischen den beteiligten Komponenten etwas genauer aufgezeigt wird.

Ausgangssituation:
Der Anwender hat Zugriff auf die Applikation und möchte sich eine virtuelle Maschine erstellen.

1. Der Anwender öffnet im Webbrowser dem VM-Builder...
2. ...und wird durch den Distributor auf die Hauptseite geleitet.
3. Dort hat der Anwender die Möglichkeit eine neue Maschine zu erstellen, in dem er dort den Create-Button betätigt. Daraufhin wird der erste Teil des Aufbaus im Buildingprocess angestossen.

4. Nachdem der Anwender Eingaben bezüglich der Grundinformation eingegeben hat, werden diese Informationen in der MachineConstruction gespeichert und ...

5. ... eine Rückmeldung über den Erfolg an die Ansicht gegeben, damit der nächste Schritt des Installationsprozesses aufgerufen werden kann.

6. Dazu benötigt die dort verwendete Ansicht alle Softwarekomponenten, die auf einer Maschine installiert werden können. Diese Informationen erhält die Sicht über einen Request an die MachineConstruction.

6a. Damit die angeforderten Daten an die anfragende Ansicht übermittelt werden könne, holt sich die MachineConstruction über den PersistanceHandler die Daten aus dem Datenspeicher.

7. Die Daten werden vom PersistanceHandler in Form eines Bundles an die MachineConstruction zurückgegeben,

7a. die wiederrum der Darstellung zur Verfügung gestellt werden, damit der Anwender seine Auswahl treffen kann.

8. Nach der Auswahl der Softwarekomponenten, weist die Ansicht die MachinConstruction an, die Ordnerstrucktur für die virtuelle Maschine zu erstellen.

8a. Um die Ordnerstruktur erstellen zu können, wird auf den SystemfileManager zurückgegriffen, der Dateisystemfunktionen bereitstellt.

9. Sind die Dateistrukturen angelegt worden, wird dies durch eine Rückmeldung des SystemfileManagers quittiert.

9a. Die nach Erhalt den nächsten Bearbeitungsschritt in der Oberfläche einleitet.

10. MachineConstruction konstruiert durch die Daten aus Schritt 4. die Konfigurationsdateien und ...

11. ... bestätigt dies.

12. Nach der Bestätigung aus Schritt 11. wird der eigentliche Aufbauprozess initialisiert.

12a. Danach wird VagrantControl angesprochen um den initialen Prozess in Vagrant anzustossen.

13. - 19 Durch automatisierten Aufbau von Vagrant, werden Statusinformationen erzeugt, die von VagrantControl verarbeitet und an die MachineConstruction weitergeleitet werden. Ein Asynchroner Prozess erfragt während des Aufbauprozesses immer wieder den Status, um den aktuellen Stand in der Anwendung anzeigen zu können.

20. Nach dem erfolgreichem Aufbau, wechselt die Anwendung wieder auf die Hauptseite.

5.5.3 Datenbank

5.5.3.1 Entity-Relationship-Model

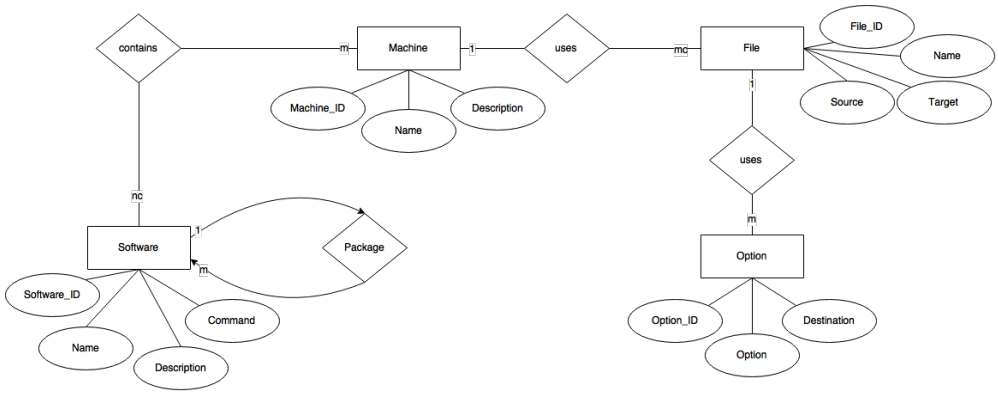


Abbildung 5.22: Entity-Relationship-Model

Wie in der Verteilungssicht (Abbildung 5.3) zu beschrieben, wird der Server eine relationale Datenbank bereit halten, deren Zuständigkeitsbereich im Speichern von Konfigurationen bereits gebauter virtuellen Maschinen und der Verwaltung von Softwarebestandteilen liegt. Der in Abbildung 5.22 veranschaulichte konzeptionelle Entwurf, zeigt das Tabellenkonstrukt für die angestrebte Datanbank in der Notation des Entity-Relationship-Model.

Die Tabelle **Machine** soll zukünftig jede Machine beinhalten, die durch das System aufgebaut wurde.

5.5.3.2 RelationalDatabaseDiagram

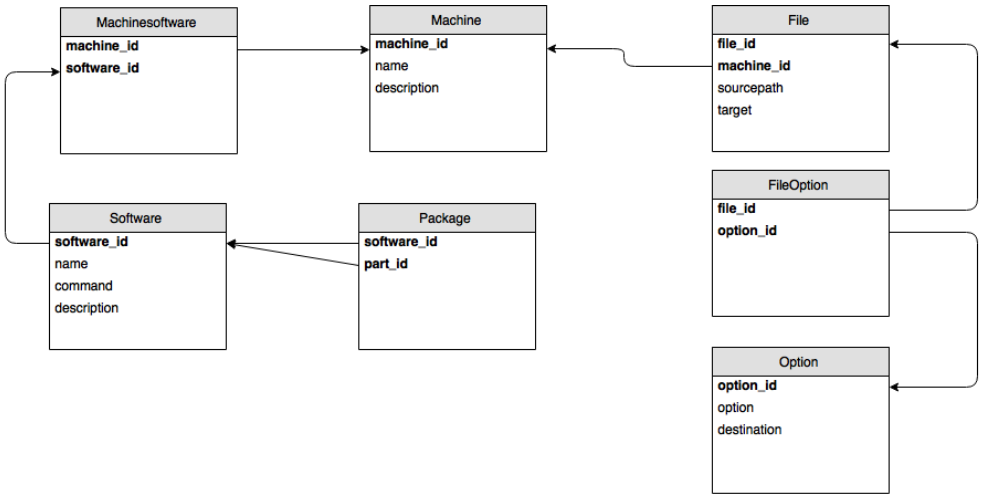


Abbildung 5.23: RelationalDatabaseDiagram

5.6 Client

Das hier angestrebte Konstrukt ist ein Thin Client. Diese Art von Clients benötigt entweder wenig oder gar keine Anwendungsteile auf dem Client-Rechner. In diesem Fall wird auf dem Client nur die Weboberfläche des VM-Builders aufgerufen um Interaktionen durchführen zu können. Die Logik für den Client ist komplett auf dem Applikationsserver implementiert. Entsprechend durch die Definition des Thin-Clients, ist an dieser Stelle keine weitere Planung oder Konzeptionierung notwendig. Die Darstellungen werden durch die Views in den entsprechenden Komponenten der Benutzerschnittstelle realisiert und zur Verfügung gestellt. In dem vorherigen Kapitel ... Abbildung ... wird die Interaktion zwischen Client-Aufrufen und serverseitigen Reaktionen genauer betrachtet.

5.7 Zusammenfassung

Literaturverzeichnis

[Balzert 2011] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. 3. Aufl. 2012. Spektrum Akademischer Verlag, 9 2011. – URL http://amazon.de/o/ASIN/3827417066/. – ISBN 9783827417060

[Bengel 2008] Bengel, Gunther: *Masterkurs Parallele und Verteilte Systeme: Grundlagen und Programmierung von Multicoreprozessoren, Multiprozessoren, Cluster und Grid (German Edition)*. 2008. Vieweg+Teubner Verlag, 6 2008. – URL http://amazon.de/o/ASIN/3834803944/. – ISBN 9783834803948

[Burge und Brown 2002] Burge, Janet E. ; Brown, David C.: NFR's: Fact or Fiction? / Computer Science Department WPI, Worcester. URL http://web.cs.wpi.edu/~dcb/Papers/CASCON03.pdf, 2002. – Forschungsbericht

[Chung u. a. 1999] Chung, Lawrence ; Nixon, Brian A. ; Yu, Eric ; Mylopoulos, John: *Non-Functional Requirements in Software Engineering (International Series in Software Engineering)*. 2000. Springer, 10 1999. – URL http://amazon.de/o/ASIN/0792386663/. – ISBN 9780792386667

[Hall 2013] Hall, Daniel: *Ansible Configuration Management*. Packt Publishing, 11 2013. – URL http://amazon.com/o/ASIN/1783280816/. – ISBN 9781783280810

[Harris und Haase 2011] Harris, Alan ; Haase, Konstantin: *Sinatra: Up and Running*. 1. O'Reilly Media, 12 2011. – URL http://amazon.com/o/ASIN/1449304230/. – ISBN 9781449304232

[Masak 2009] Masak, Dieter: *Der Architekturreview: Vorgehensweise, Konzepte und Praktiken (Xpert.press)*. 2010. Springer, 11 2009. – URL http://amazon.de/o/ASIN/3642016588/. – ISBN 9783642016585

[Peacock 2013] Peacock, Michael: *Creating Development Environments with Vagrant*. Packt Publishing, 8 2013. – URL http://amazon.de/o/ASIN/1849519188/. – ISBN 9781849519182

[Rechtin und Maier 2000] Rechtin, Eberhardt ; Maier, Mark: *The Art of Systems Architecting, Second Edition*. 0002. Crc Pr Inc, 6 2000. – URL http://amazon.de/o/ASIN/0849304407/. – ISBN 9780849304408

[Reuther 2013] Reuther, Claus: *Virtualisierung - VMware und Microsoft im Vergleich*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 1 2013

[Rhett 2015] Rhett, Jo: *Learning Puppet 4*. 1. O'Reilly Vlg. Gmbh and Co., 8 2015. – URL http://amazon.de/o/ASIN/1491907665/. – ISBN 9781491907665

[Rupp und die SOPHISTen 2014] Rupp, Chris ; SOPHISTen die: *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil*. 6., aktualisierte und erweiterte Auflage. Carl Hanser Verlag GmbH und Co. KG, 10 2014. – URL http://amazon.de/o/ASIN/3446438939/. – ISBN 9783446438934

[Schäfer 2009] Schäfer, Werner: *Softwareentwicklung - inkl. Lerntest auf CD: Einstieg für Anspruchsvolle (Master Class)*. 1. Addison-Wesley Verlag, 12 2009. – URL http://amazon.de/o/ASIN/3827328519/. – ISBN 9783827328519

[ScriptRock 2014] ScriptRock: *Ansible vs. Salt*. Januar 2014. – URL https://www.scriptrock.com/articles/ansible-vs-salt

[Seneca 2005] Seneca: *Von der Kürze des Lebens*. Deutscher Taschenbuch Verlag, 11 2005. – URL http://amazon.de/o/ASIN/342334251X/. – ISBN 9783423342513

[Siegert und Baumgarten 2006] Siegert, Hans-Jürgen ; Baumgarten, Uwe: *Betriebssysteme: Eine Einführung*. überarbeitete, aktualisierte und erweiterte Auflage. Oldenbourg Wissenschaftsverlag, 12 2006. – URL http://amazon.de/o/ASIN/3486582119/. – ISBN 9783486582116

[Sinatra 2015] Sinatra: *Sinatra Einführung*. Juni 2015. – URL http://www.sinatrarb.com/intro-de.html#Bedingungen

[Starke 2011] Starke, Gernot: *Software-Architektur kompakt (IT kompakt)*. 2nd Printing. Spektrum Akademischer Verlag, 3 2011. – URL http://amazon.de/o/ASIN/3827420938/. – ISBN 9783827420930

[Starke 2014] Starke, Gernot: *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. 6., überarbeitete Auflage. Carl Hanser Verlag GmbH und Co. KG, 1 2014. – URL http://amazon.de/o/ASIN/3446436146/. – ISBN 9783446436145

[Tanenbaum und van Steen 2007] Tanenbaum, Andrew S. ; Steen, Maarten van: *Verteilte Systeme: Prinzipien und Paradigmen (Pearson Studium - IT)*. 2., aktualisierte Auflage. Pearson Studium, 11 2007. – URL http://amazon.de/o/ASIN/3827372933/. – ISBN 9783827372932

[Wikipedia 2015] Wikipedia: *Model ? view ? controller — Wikipedia - The Free Encyclopedia*. 2015. – URL https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/2000px-MVC-Process.svg.png

[Zörner 2012] Zörner, Stefan: *Softwarearchitekturen dokumentieren und kommunizieren: Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten*. Carl Hanser Verlag GmbH und Co. KG, 5 2012. – URL http://amazon.de/o/ASIN/3446429247/. – ISBN 9783446429246

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. Januar 2015 Jan Lepel