Höhere Technische Bundeslehranstalt Wien 3, Rennweg
IT & Mechatronik

HTL Rennweg :: Rennweg 89b
A-1030 Wien :: Tel +43 1 24215-10 :: Fax DW 18

Diplomarbeit

Vsistent
Simplify Your Lab Experience

ausgeführt an der
Höheren Abteilung für Informationstechnologie/Netzwerktechnik
der Höheren Technischen Lehranstalt Wien 3 Rennweg

im Schuljahr 2018/2019

durch

Andreas Himmler
Julian Kern
Raffael Zbiral

unter der Anleitung von

August Hörandl
Matthias Drucks

Wien, 19. Mai 2019

Kurzfassung

Die grundlegende Idee ist es, das Starten der virtuellen Maschinen im EDV-Laborbetrieb mit den benötigten Toolsets zu vereinfachen. Diese Vereinfachung soll durch das Entwickeln einer eigenen Applikation geschehen, in welcher Benutzer virtuelle Maschinen einfach klonen und starten können. Zusätzlich sollen Lehrerinnen und Lehrer mit dieser Anwendung neue virtuelle Maschinen erstellen können. Außerdem soll Administratorinnen und Administratoren das Verteilen von virtuellen Maschinen vereinfacht werden. Um die Benutzerfreundlichkeit des Programmes im Schulalltag zu testen, soll es auf Computern in den EDV-Sälen der Schule installiert werden.

Abstract

The fundamental idea is the simplification of the starting process for virtual laboratory environments for students. This is achieved by developing an application which reduces the effort of cloning and starting a virtual machine to a simple button press. Additionally, teachers should be able to create new virtual machines with this program. Moreover, this application should enable administrators to deploy virtual environments more easily. The program should also be installed on computers in the computer laboratories of the HTL Rennweg to verify its usability.

Ehrenwörtliche Erklärung

Ich erkläre an Eides statt, dass ich die individuelle Themenstellung selbstständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wien, am 19. Mai 2019

Andreas Himmler

Julian Kern

Raffael Zbiral

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Ausgangssituation

Die Schülerinnen und Schüler arbeiten in den EDV-Sälen der HTL Rennweg fast ausschließlich auf virtuellen Maschinen. Das bedeutet, dass sie nach dem Anmelden auf einem Computer zuerst VMware Workstation Pro starten müssen. Anschließend suchen sie die benötigte Master-VM, klonen diese und erstellen einen Snapshot. Erst nach dieser Prozedur können die Benutzer die virtuelle Maschine starten.

Diese Arbeitsschritte beinhalten einige Klicks durch die Benutzeroberfläche von VMware Workstation Pro und nehmen dadurch auch relativ viel Zeit in Anspruch. Deshalb kann ein Absturz einer virtuellen Maschine in einer Prüfungssituation viel Stress erzeugen.

1.2 Unsere Idee

Um diese Lage zu verbessern, haben wir uns als Ziel gesetzt ein Programm zu entwickeln, welches die Arbeiten der Schülerinnen und Schüler mit virtuellen Maschinen vereinfacht.

Um eine möglichst ideale Lösung zu entwickeln, werden zuerst verschiedene Hypervisoren miteinander verglichen und bewertet. Um Erfahrung mit den Hypervisoren und ihren Kommandozeilentools zu sammeln, werden Testumsetzungen erstellt. Diese sollen auch schon Verbesserungen für den Schulalltag mit sich bringen.

Abschließend wird ein Programm entwickelt, mit welchem die Benutzer einfacher und effizienter mit virtuellen Maschinen interagieren können. Außerdem sollen Lehrerinnen und Lehrer dadurch einfacher virtuelle Maschinen für ihren Unterricht erstellen können und das Verteilen dieser soll für den Administrator vereinfacht werden.

1.3 Unsere Lösung

1.3.1 Überblick

Im Rahmen dieser Diplomarbeit wurden nach den in Kapitel 2 beschriebenen Evaluierungen diverse unterschiedliche Lösungsansätze verfolgt und umgesetzt. Zum einem wurde eine Benutzer- eine LehrerInnen- und eine AdministratorInnen-Applikation geschrieben, welche den Umgang mit virtuellen Maschinen vereinfacht. Für diese grafischen Oberflächen wurde das plattformunabhängige Qt Framework verwendet, welches in Kapitel 3.1 Qt Framework genauer beschrieben wird.

Zusätzlich zu diesen drei betriebssystemunabhängigen Hauptanwendungen wurde jeweils eine weitere Lösung für Linux und Windows Computer entwickelt. Das Ziel dieser Implementierungen war es, das Starten von virtuellen Maschinen zu automatisieren und Shared-Folder automatisch in das Host-System einzubinden. Dadurch können Dateien einfacher zwischen dem Host- und dem Gastsystem geteilt werden und darin abgelegten Dateien sind auch vor dem Ausfall einer virtuellen Maschine gesichert.

1.3.2 Benutzer-Applikation

1.3.2.1 Beschreibung

Mithilfe der grafischen Benutzeroberfläche soll es dem Benutzer möglich sein alle wesentlichen Funktionen für alle virtuellen Maschinen, durchzuführen. Ein Ziel war es auch, diese grafische Oberfläche einfach im Aufbau zu halten, um eine intuitive Verwendung zu gewährleisten.

Um den Aufbau möglichst modular zu halten, haben wir für die unterschiedlichen Interaktionen mit den virtuellen Maschinen verschiedene Skripte verwendet. Dadurch wird auch ein einfacher Wechsel zwischen verschiedenen Hypervisoren ermöglicht, da dafür nur die Skripte geändert werden müssen, nicht die Schnittstellen. Weitere Informationen zu der Benutzer-Applikation werden in Kapitel 3.2 Benutzer-Applikation beschrieben.

1.3.2.2 Abgrenzung

Das Programm wurde entwickelt, um das Arbeiten mit virtuellen Maschinen zu vereinfachen, wenn auf dem Hostcomputer wenig in Bezug auf die Arbeitsumgebungen geändert wird. Es werden zwar neue Master-VMs erkannt, aber nur, wenn sie in einem angegebenen Ordner gespeichert werden. Außerdem können Benutzer zwar mit den

virtuellen Maschinen einfach interagieren, jedoch können eine Reihe von Aktionen nicht durchgeführt werden, welche prinzipiell direkt mit VMware Workstation Pro möglich sind. Zum Beispiel kann man über das Programm die virtuelle Hardware oder die Netzwerkanbindung der Gastsysteme nicht anpassen. Es muss auch erwähnt werden, dass diese Applikation zum Arbeiten mit jeweils einer virtuellen Maschine eines Typs gedacht ist, weshalb es nicht möglich ist dieselbe virtuelle Maschine zweimal zu klonen und auszuführen.

Diese Kompromisse wurden aber alle eingegangen, um eine möglichst intuitive Benutzeroberfläche zu entwickeln, welche in der Praxis das Arbeiten mit virtuellen Maschinen verbessert.

1.3.3 LehrerInnen-Applikation

1.3.3.1 Beschreibung

Die grundsätzliche Idee dieser grafischen Oberfläche war es, Lehrerinnen und Lehrern das Erstellen und Verteilen der virtuellen Maschinen, die im Unterricht benötigt werden, deutlich zu vereinfachen.

Um dies zu bewerkstelligen, werden alle Einstellungen, welche im Normalfall im Hypervisor getätigt werden müssen, wie auch das Klonen und das Erstellen der eigentlichen virtuellen Maschine, von dieser Applikation automatisch durchgeführt.

Die einzigen Handgriffe, die eine Lehrkraft tätigen muss, sind die Eingabe von Metadaten, wie dem Namen der Gast-Maschine oder auch der E-Mail-Adresse des Lehrpersonals. Ist dies erledigt, kann die Lehrkraft die virtuelle Maschine starten und alle benötigten Programme installieren. Sobald dies abgeschlossen wurde, muss nur eine Schaltfläche betätigt werden und die Gast-Maschine wird auf den Server hochgeladen. Nun kann die neue virtuelle Maschine von einer Administratorin oder einem Administrator begutachtet und verteilt werden. Weitere Informationen zu der LehrerInnen-Applikation werden in Kapitel 3.4 LehrerInnen-Applikation beschrieben.

1.3.4 AdministratorInnen-Applikation

1.3.4.1 Beschreibung

Mit der AdministratorInnen-Applikation soll die Verwaltung von virtuellen Maschinen im Schulnetz für die Administratorinnen und Administratoren vereinfacht werden. Hierbei war das Ziel eine einfache grafische Oberfläche, mit der man, mit wenig Aufwand, die Gast-Maschinen verteilen und verwalten kann.

Dabei können die Administratorinnen und Administratoren die virtuellen Maschinen, die von Lehrpersonal eingereicht wurden, auf einen Blick sehen und daraufhin entscheiden, ob diese ausgerollt werden soll oder nicht. Des Weiteren können die Gast-Maschinen, die bereits ausgerollt wurden, wieder gelöscht werden.

Hierbei ist die AdministratorInnen-Applikation jedoch nur eine grafische Oberfläche zum Verwalten des Servers, der die ganze Logik beherbergt. Weitere Informationen zu dieser Applikation werden im Kapitel 3.3 AdministratorInnen-Applikation beschrieben.

1.3.5 Backend – Server

1.3.5.1 Beschreibung

Die Server-Applikation soll die gesamte Logik des Verwaltens und Verteilens der virtuellen Maschinen übernehmen. Das bedeutet, dass alle Funktionen der AdministratorInnen-Applikation im Hintergrund über diese Anwendung laufen. Des Weiteren beinhaltet der Server auch die Schnittstelle zur Diplomarbeit „SystemTrojaner“, da die Software dieser Diplomarbeit das Verteilen der Gast-Maschinen übernimmt. Weitere Informationen zum Backend und dem Server werden in Kapitel 4.1 Backend – Server beschrieben.

1.3.6 Linuxskripte

1.3.6.1 Beschreibung

Das Ziel dieser Lösung ist es, das Starten und Klonen von virtuellen Maschinen auf einem Linux-Hostrechner zu vereinfachen und beschleunigen.

Außerdem soll automatisch ein Shared-Folder eingebunden werden, damit Schülerinnen und Schüler ihre Dateien in diesem Ordner speichern können. Dadurch können Dateien einfacher zwischen dem Host- und dem Gastsystem geteilt werden und die darin abgelegten Dateien sind auch vor dem Ausfall einer virtuellen Maschine gesichert. Damit diese Lösung auch von möglichst vielen Benutzern verwendet wird, waren die einfache Verwendung und das Einbinden von Verknüpfungen auch wichtige Punkte bei der Entwicklung dieser Lösung. Weitere Informationen zu den Linuxskripten werden in Kapitel A.1.3 Linuxskripte beschrieben.

1.3.6.2 Abgrenzung

Diese Lösung wurde, wie auch die Benutzer-Applikation, zum Starten von einer virtuellen Maschine entwickelt, weshalb man von einer Master-VM nicht zwei Klone erstellen kann. Dies liegt an der Namensgebung bei den Klonen der virtuellen Maschinen, da an dem Namen der Master-VM „-Clone“ angehängt wird und dadurch zwei Klone der gleichen virtuellen Maschine nicht unterschieden werden können. Das automatische Einhängen der Shared-Folder funktioniert nur, wenn die Anmeldedaten im Gastbetriebssystem nicht geändert werden. Diese Einschränkung besteht, weil man bei dem Ausführen eines Skriptes in der Gastmaschine die Anmeldedaten benötigt und diese dem Startskript mitgegeben werden. Diese Skripte können außerdem nur zum Starten der virtuellen Maschinen verwendet werden. Es wird zwar nach dem Klonen ein Snapshot erstellt, jedoch müssen weitere Snapshots und etwaige andere Anpassungen, wie das Ändern der virtuellen Hardware, über das User Interface von VMware Workstation durchgeführt werden.

1.3.7 Windowsskripte

1.3.7.1 Beschreibung

Durch die Windowsskripte wird für die Schülerinnen und Schüler das Starten von virtuellen Maschinen und die dazu erforderlichen Schritte auf Windows-Host-Systemen automatisiert. Zusätzlich wird ein Shared-Folder eingebunden, der das Host-Betriebssystem mit dem Guest-Betriebssystem verbindet. Dieser soll den Schülerinnen und Schülern die Möglichkeit geben, ihre in der virtuellen Maschine erstellte Daten, einfach und unkompliziert auf das Host-Betriebssystem zu übertragen.

Da es sich bei dieser Lösung um eine vorläufige Testlösung handelt, war es besonders wichtig, dass der bereits etablierte Umgang mit virtuellen Maschinen in keiner Weise betroffen ist. Es sollte sich eben nur um eine Erweiterung handeln, welche für ein angenehmes Arbeiten mit virtuellen Maschinen verwendet werden kann, aber nicht muss. Weiter Informationen zu den Windowsskipten werden in Kapitel A.1.4 Windowsskripte beschrieben.

1.3.7.2 Abgrenzung

Da diese Lösung nur das Starten von virtuellen Maschinen automatisiert, ist es nicht möglich etwaige Funktionen für laufende virtuelle Maschinen auszuführen. Mit dieser Implementierung ist es beispielsweise nicht möglich, einen Snapshot automatisiert zu erstellen oder den Klon zu einem Snapshot zurückzusetzen. Solche Funktionen sind nur über die grafische Oberfläche von VMware Workstation Pro möglich.

Des Weiteren ist es nur möglich einen Klon je Master-VM zu erstellen, dies liegt an der statischen Namensgebung des Klons. Ein weiterer Nachteil liegt darin, dass bei dieser Umsetzung für jede Master-VM zwei eigene Skripte erforderlich sind. Auch ist keine grafische Oberfläche für den Benutzer vorhanden, weshalb die Skripte manuell ausgeführt werden müssen.

1.4 Ziele und Userstories

1.4.1 Hauptziele

Ziel-H 1 Erstellung von Evaluierungskriterien für die Hypervisoren
Es wird eine Bewertungsmatrix erstellt, anhand von welcher die in Frage stehenden Hypervisoren verglichen werden können.

Es werden diverse Hypervisoren in Betracht gezogen und um einen von diesen auswählen zu können, müssen Kriterien für die Evaluierung festgelegt werden. Diese Kriterien werden anhand des Verwendungszweckes ausgewählt.

Ziel-H 2 Evaluierung der Virtualisierung mit Hyper-V
Es wird festgestellt und dokumentiert, ob die Virtualisierung mit dem Hypervisor Hyper-V für unsere Anwendungszwecke geeignet ist.

Es wird die Virtualisierung mit dem Hypervisor Hyper-V analysiert und evaluiert. Die Vorteile und Nachteile dieses Hypervisors werden dokumentiert und diese werden als Grundlage für den Vergleich mit den anderen Hypervisoren herangezogen.

Ziel-H 3 Evaluierung der Virtualisierung mit VMWare
Es wird festgestellt und dokumentiert, ob die Virtualisierung mit dem Hypervisor VMWare für unsere Anwendungszwecke geeignet ist.

Es wird die Virtualisierung mit dem Hypervisor VMWare analysiert und evaluiert. Die Vorteile und Nachteile dieses Hypervisors werden dokumentiert und diese werden als Grundlage für den Vergleich mit den anderen Hypervisoren herangezogen.

Ziel-H 4 Evaluierung der Virtualisierung mit Virtual Box
Es wird festgestellt und dokumentiert, ob die Virtualisierung mit dem Hypervisor Virtual Box für unsere Anwendungszwecke geeignet ist.

Es wird die Virtualisierung mit dem Hypervisor Virtual Box analysiert und evaluiert. Die Vorteile und Nachteile dieses Hypervisors werden dokumentiert und diese werden als Grundlage für den Vergleich mit den anderen Hypervisoren herangezogen.

Ziel-H 5 Entwicklung der finalen Lösung
Es wird ein Tool entwickelt, welches den Umgang mit den vituellen Maschinen in EDV-Laborumgebungen vereinfacht.

Es wird das Starten und Klonen von virtuellen Maschinen in EDV-Laborumgebungen durch unser Tool, welches entwickelt wird, vereinfacht. Da die Entwicklung des Tools jedoch agil geplant wird, wird erst im Laufe des Projektes festgelegt, wie unser Tool genau aussehen wird. Um eine Entscheidung für für den zu verwendenden Hypervisor zu treffen, wird jedes Teammitglied seine Ergebnisse der Recherche präsentieren.

Ziel-H 6 Testinstallation
Unsere Software wird zumindest auf zwei Computern in einem EDV-Saal implementiert, um die Funktionalität der Software im Normalbetrieb mit Schülerinnen und Schülern zu testen.

Damit die Software auf ihre Tauglichkeit im Schulalltag getestet werden kann, wird diese auf mindestens zwei Arbeitsplatz-PCs in Betrieb sein. Um diesen Testlauf auch durchführen zu können, wird ein Handbuch für die Schüler und Lehrer geschrieben und gegebenenfalls auch eine Schulung für diese durchgeführt. Dieser Test ermöglicht es auch festzustellen, wie praktikabel der schulweite Einsatz unserer Software ist.

Ziel-H 7 Webseite
Es wird eine Webseite für unser Projekt erstellt und veröffentlicht.

Für unser Projekt wird eine Webseite erstellt, auf welcher wir Informationen über unser Projekt veröffentlichen und über unseren Fortschritt berichten.

1.4.2 Haupt-User-Stories

Zusätzlich zu den Hauptzielen wurden User-Stories definiert, wie dies bei agilen Projektplanungsmethoden üblich ist.

User-Story-H 1
Als USER (SCHÜLER, LEHRER) kann ich unmittelbar nach dem Start des Arbeitsplatz-PCs und erfolgter AD-Anmeldung aus mehreren Möglichkeiten eine Arbeitsumgebung auswählen und für den aktuellen Unterricht, die aktuelle Aufgabe oder Prüfung verwenden.

User-Story-H 2
Als LEHRER möchte ich neue Arbeitsumgebungen, die meinen Bereich betreffen, den Schülern zur Verfügung stellen und alte löschen können, um diese zu verbessern oder aktuell zu halten.

User-Story-H 3
Als ADMIN kann ich kontrollieren, welche Arbeitsumgebungen den USERN zur Verfügung stehen und dementsprechend auch neue Umgebungen in das System integrieren.

User-Story-H 4
Als ADMIN möchte ich das System einfach integrieren, erweitern und warten können, damit es auf Umstellungen des Unterrichts angepasst werden kann.

1.4.3 Optionale Ziele

Ziel-O 1 Evaluierung alternativer Hypervisoren unter Windows
Es wird festgestellt und dokumentiert, ob Hypervisoren unter Windows existieren, welche für unseren Anwendungszweck geeignet sind.

Es wird die Virtualisierung mit anderen Hypervisoren unter Windows evaluiert und dokumentiert. Wenn für unseren Anwendungszweck geeignete Hypervisoren gefunden werden, werden diese auch mit den anderen Hypervisoren verglichen.

Ziel-O 2 Evaluierung alternativer Hypervisoren unter Linux
Es wird festgestellt und dokumentiert, ob Hypervisoren unter Linux existieren, welche für unseren Anwendungszweck geeignet sind.

Es wird die Virtualisierung mit anderen Hypervisoren unter Linux evaluiert und dokumentiert. Wenn für unseren Anwendungszweck geeignete Hypervisoren gefunden werden, werden diese auch mit den anderen Hypervisoren verglichen.

Ziel-O 3 Evaluierung von Bare-Metal Hypervisoren
Es wird festgestellt und dokumentiert, ob Bare-Metal Hypervisoren existieren, welche für unseren Anwendungszweck geeignet sind.

Es wird die Virtualisierung mit Bare-Metal Hypervisoren evaluiert und dokumentiert. Wenn für unseren Anwendungszweck geeignete Hypervisoren gefunden werden, werden diese auch mit den anderen Hypervisoren verglichen.

Ziel-O 4 Verwaltung durch Systemtrojaner
Das Betriebssystem des Arbeitsplatz-PCs und die Betriebssysteme der Arbeitsumgebungen werden in die Verwaltung durch die Diplomarbeit Systemtrojaner eingebunden.

Die Diplomarbeit Systemtrojaner entwickelt eine Verwaltungssoftware, welche die Arbeit der Schuladministratoren erleichtern soll. Um dies mit unseren Arbeitsumgebungen zu ermöglichen, müssen diese deshalb in die Verwaltung des Systemtrojaners eingebunden werden.

1.4.4 Optionale User-Stories

Zusätzlich zu den optionalen Zielen wurden optionale User-Stories definiert, wie dies bei agilen Projektplanungsmethoden üblich ist.

User-Story-O 1
Als SCHÜLER kann ich eine neue virtuelle Maschine verwenden und kann in dieser neuen Arbeitsumgebung weiterhin auf meine Daten zugreifen, welche in der vorherigen Arbeitsumgebung gespeichert wurden.

User-Story-O 2
Als LEHRER kann ich den Netzwerkzugang der Arbeitsumgebungen während des Unterrichts beziehungsweise einer Prüfung einschränken.

1.4.5 NICHT Ziele

RE-N 1 Fortführende Wartung des Systems
Das System wird nach Abschluss der Diplomarbeit von uns gewartet.

2 Evaluierung

2.1 VMware Workstation

2.1.1 Allgemeines

VMware Workstation Pro wird von VMware entwickelt, einem der größten Unternehmen für Virtualisierungs- und Cloud-Computing-Software. VMware Workstation Pro ist eine der am weitesten verbreitetsten Softwarelösungen, um die Ausführung, ein oder mehrere Gastbetriebssysteme, als virtuelle Maschine, auf einem Linux- oder Windows PC, zu ermöglichen. Diese virtuellen Maschinen können entweder über die Benutzeroberfläche von VMware Workstation Pro erstellt, konfiguriert und gestartet werden, oder über das von VMware entwickelte Command-Line Interface *vmrun*. Die wesentlichen Konfigurationen dieser Maschinen werden in einer Datei mit der Endung .vmx gespeichert. [10]

2.1.2 Konfigurationsdatei

Die vmx-Datei enthält unter anderem Angaben zur virtuellen Hardware. In ihr werden Beispielsweise die Anzahl der CPU Kerne und die verfügbare RAM-Größe einer virtuellen Maschine festgehalten. Informationen bezüglich des Gastbetriebssystems oder zur Art der Netzwerkverbindung werden ebenfalls in dieser vmx-Datei gespeichert. Bei einer vmx-Datei handelt es sich um eine reine Textdatei, welche in ihrem Format jenem von .ini-Files ähnelt. Weitere Details zu diesem Format findet man unter [13].

2.1.3 vmrun

Vmrun ist das von VMware zur Verfügung gestellte Kommandozeilentool, um die wichtigsten Einstellungen und Funktionen von VMware Workstation Pro zu nutzen, ohne auf die grafische Benutzeroberfläche von VMware angewiesen zu sein. Das Zugreifen auf diese vmrun-Utility gestaltet sich je nach Host-Betriebssystem unterschiedlich. Auf einem Linux-Host muss hierzu nur der Befehl vmrun eingegeben werden, um die verschiedenen Funktionen zu erhalten. Unter Windows befindet sich dieses Tool im

gleichen Verzeichnis, in dem auch VMware Workstation installiert ist. Für das Nutzen der Funktionen von vmrun muss unter Windows-Hosts entweder der gesamte Pfad zu vmrun.exe angegeben werden, oder der Systempfad wird so geändert, dass dieser „VMware Workstation“ einschließt. [6]

2.1.4 Vorteile

Vorab lässt sich sagen, dass VMware Workstation Pro alle benötigten Funktionen zum problemlosen Arbeiten mit virtuellen Maschinen bereitstellt. Zusätzlich lassen sich diese Funktionen weitgehend mit dem Kommandozeilentool von VMware bedienen. Ein weiterer Vorteil ist, dass VMware Workstation Pro zurzeit auf den Rechnern in unserer Schule, verwendet wird. Demnach ist das Lehrpersonal bereits mit diesem Hypervisor vertraut und die Installation zusätzlicher Software kann minimal gehalten werden. Zudem fallen zumindest keine zusätzlichen Aufwände und Kosten bezüglich der Lizenzierung eines alternativen Hypervisors an.

2.1.5 Nachteile

Ein Nachteil von VMware Workstation Pro gegenüber einem alternativen Typ-1-Hypervisor liegt in der Performance, die bei einem Typ-2-Hypervisor spürbar schwächer ist [8]. Ein weiterer Nachteil von VMware Workstation Pro besteht durch die Einschränkungen der vmrun-Utility, welche nur die wesentlichsten Funktionen von VMware Workstation Pro abdeckt. Aus diesem Grund musste die Konfigurationsdatei teilweise manuell editiert werden, um weiter Funktionen von VMware Workstation Pro zu nutzen. Konkret werden diese manuellen Anpassungen in der VMware Befehlsreferenz beschrieben (siehe 2.1.7 Befehlsreferenz).

2.1.6 Fazit

VMware Workstation Pro ist für die Anwendung in unserer Schule die beste Option. Wir bauen direkt auf der zurzeit implementierten Lösung auf. Dadurch ist es möglich, unsere Software als Erweiterung anzusehen, wodurch die Möglichkeit des manuellen Startens von virtuellen Maschinen erhalten bleibt. Trotz gewisser Einschränkungen bietet VMware Workstation Pro alle für uns benötigten Funktionen (siehe 2.1.7 Befehlsreferenz).

2.1.7 Befehlsreferenz

Funktion	Befehl
Starten einer VM	`vmrun start` *<path to .vmx>*
Stoppen einer VM	`vmrun stop` *<path to .vmx>*
Löschen einer VM	Löschen des Verzeichnisses der VM
Klonen einer VM	`vmrun clone` *<path to .vmx>* `[full | linked]` *<snapshot name>*
Erstellen eines Snapshots	`vmrun snapshot` *<path to .vmx>* *<snapshot name>*
Snapshots auflisten	`vmrun listSnapshots` *<path to .vmx>*
Zurücksetzen einer VM	`vmrun revertToSnapshot` *<path to .vmx>* *<snapshot name>*

Starten im Vollbildmodus	Zusätzliche Einträge in die .vmx Datei:
	`gui.fullScreenAtPowerOn = "TRUE"`
	`gui.viewModeAtPowerOn = "fullscreen"`

Tabelle 2.1: VMware Workstation Befehlsreferenz

2.2 Hyper-V

2.2.1 Allgemeines

Hyper-V ist ein Hypervisor aus dem Hause *Microsoft*, welcher entweder ohne ein zusätzliches Betriebssystem, auf *Windows Pro* oder auch auf *Windows Server* betrieben werden kann. Dieser Hypervisor kann entweder als Typ-1 oder auch, wie es manche machen als Typ-1/2-Hybrid Hypervisor angesehen werden. Das hat den Grund, dass, wenn man *Hyper-V* installiert, das ursprüngliche Betriebssystem auch zu einer Art virtuellen Maschine umgewandelt wird, die hardware-nahe Funktionen realisiert. [14]

Grundsätzlich besitzt *Hyper-V* keine grafische Oberfläche. Der Hypervisor kann jedoch mithilfe einer *Microsoft Management Console* verwaltet werden, wenn die *Hyper-V Management Tools* installiert sind.

Will man die grafische Oberfläche einer Gast-Maschine betrachten, muss man entweder eine Bildschirm-Übertragung direkt zu der virtuellen Maschine starten, oder man verbindet sich per RDP auf den *Hyper-V*-Host. Um hier jedoch das Bild von dem Gast-Betriebssystem und nicht das des Hosts zu bekommen, muss man sich mit dem Port 2179 verbinden. Des Weiteren muss der PCB (preconnection blob) definiert werden. Der Wert, den hier mitgegeben werden muss, ist die VMID. Dies ist eine eindeutige Kennung, mit der jede virtuelle Maschine identifiziert werden kann.

Dies kann automatisiert mit dem Programm *vmconnect* erledigt werden, welches über die *Windows Features* installiert werden kann. In diesem Programm muss nur der Netzwerkname des Rechners, auf dem *Hyper-V* installiert ist, angegeben werden. Sobald dies geschehen ist, werden automatisch alle virtuellen Maschinen, die auf dem Host registriert sind, zur Auswahl aufgelistet. Wird eine dieser virtuellen Maschinen ausgewählt und auf *„Ok“* geklickt, wird automatisch eine Verbindung, wie im vorherigen Absatz beschrieben, aufgebaut. Eine andere Möglichkeit wäre die Bibliothek *„mstscax.dll“*. Mit dieser kann man eine solche Verbindung ausprogrammieren, wie im Kapitel A.1.2.3 Connection beschrieben wird.

Die VMID kann nur mithilfe der *PowerShell* herausgefunden werden. Dies funktioniert mithilfe des folgenden Befehls:

```
1 Get-VM -Name \textit{<vmname>} | Select vmid
```

Listing 2.1: PowerShell-Befehl zum Herausfinden der VMID

Hierbei muss für *vmname* der Name der virtuellen Maschine, von der man die VMID herausfinden möchte, angeben. Führt man diesen Befehl somit aus, bekommt man als Ausgabe den VM-Identifier.

2.2.2 Hyper-V PowerShell Module

Wenn man *Hyper-V* installiert, wird automatisch das Kommandozeilenwerkzeug mitinstalliert. Dieses nennt sich *Hyper-V PowerShell Module*. Alle Aktionen, die man mit der *Microsoft Management Console* tätigen kann, können auch per *PowerShell* getätigt werden. Sollte die Möglichkeit bestehen, ist es zu empfehlen auf das *Hyper-V PowerShell Module* zurückzugreifen. Dies hat mehrere Gründe. Einerseits können in der Konfigurationsdatei keine Änderungen durchgeführt werden, da diese im Binärcode gespeichert wird. Anderseits, gäbe es zwar die *Microsoft Management Console*, diese beherbergt jedoch nicht alle Funktionen, die *Hyper-V* zu bieten hat. Diese fehlenden Funktionen sind alle mit dem *Hyper-V PowerShell Module* ausführbar.

2.2.3 Vorteile

Sollte *Windows Pro* oder auch einen *Windows Server* in Betrieb sein, können auf diesen Betriebssystemen der Hypervisor *Hyper-V* kostenlos installiert werden. Alle Funktionen stehen auch von Haus aus zur Verfügung und es müssen keine weiteren hinzugekauft werden.

Des Weiteren sind von *Microsoft* alle Funktionen des Hypervisors in den *Microsoft Docs* beschrieben und damit gut dokumentiert.

Das Skripting kann mithilfe des *Hyper-V PowerShell Modules* realisiert werden. Dieses ist, wie beschrieben, sehr ausführlich und man kann im Gegensatz zu *VMware Workstation* alles mit dieser machen und muss nicht in Konfigurationsdateien selbst Änderungen vornehmen.

2.2.4 Nachteile

Hyper-V kann nur installiert werden, wenn man entweder *Windows Pro* oder einen *Windows Server* hat. Dies hat zur Folge, dass man mehr Geld in das Betriebssystem investieren muss, nur um den Hypervisor zu verwenden. Außerdem ist er dadurch, im Gegensatz zu den anderen Virtualisierungsumgebungen, nicht plattformunabhängig.

Der Hypervisor *Hyper-V* ist außerdem nicht sehr intuitiv. Daher ist die Lernkurve zum Verwenden dieses Hypervisors durchaus anspruchsvoll im Gegensatz zu den anderen, die in dieser Evaluierung eine Rolle spielen.

2.2.5 Fazit

Hyper-V ist eine sehr gute Option, wenn man viel Performance und ein ausgereiftes Kommandozeilentool benötigt. Jedoch ist dieser Hypervisor nicht für Laien des Gebietes geeignet, da dieser sehr unterschiedlich zu den anderen Virtual-Machine-Montitore in dieser Liste ist. Außerdem muss man in jedem Fall Zeit investieren um mit *Hyper-V* klarzukommen, da dieser sich in der Handhabung sehr stark von seinen Konkurrenten unterscheidet. Da es ein Ziel der Vsistent-Lösung war, dass diese plattformunabhängig sein sollte, wurde dieser Hypervisor nicht weiter in Betracht gezogen.

2.2.6 Befehlsreferenz

Die Befehle sind einerseits im Manual beschrieben und andererseits wurde dies in den *Microsoft Docs* auch sehr ausführlich dokumentiert. Das Manual kann man in der *PowerShell* mithilfe dieses Befehls aufrufen:

```
1 Get-Command -Module Hyper-V
```

Listing 2.2: PowerShell-Befehl zum Aufrufen des Manuals zu Hyper-V

Hier sind die wichtigsten Befehlszeilen gelistet, die in der Vsistent-Lösung zum Einsatz kommen würden: [2]

Funktion	Befehl
Starten einer VM	`Start-VM -Name` *<vmname>*
Stoppen einer VM	`Stop-VM -Name` *<vmname>*
Löschen einer VM	`Remove-VM` *<vmname>* `-Force` `Remove-Item -Path` *<path-to-vhdx>*
Klonen einer VM	`New-VHD -Differencing -Path` *<new-vhdx>* ‘ `-ParentPath` *<master-vhdx>* `New-VM -Name` *<vmname>* `-VHDPath` *<new-vhdx>*
Erstellen eines snapshots	`Checkpoint-VM -Name` *<vmname>* ‘ `-SnapshotName` *<snapshotname>*
Snapshots auflisten	`Get-VMSnapshot -VMName` *<vmname>*
Zurücksetzen einer VM	`Restore-VMSnapshot -Name` *<snapshotname>* ‘ `-VMName` *<vmname>*
Starten im Vollbildmodus	Muss selbst programmiert werden, wie im Kapitel A.1.2.3 beschrieben
Verstecken von GUI-Elementen	Muss selbst programmiert werden, wie im Kapitel A.1.2.3 beschrieben

Tabelle 2.2: Hyper-V Befehlsreferenz

2.3 VirtualBox

2.3.1 Allgemeines

VirtualBox ist ein Open-Source Typ-2-Hypervisor von Oracle, welcher auf Windows, Linux und OS X eingesetzt werden kann.[9] Mit diesem Virtual-Machine-Monitor können virtuelle Maschinen über die grafische Oberfläche oder über die Kommandozeile konfiguriert werden. Sämtliche Konfigurationen der virtuellen Maschinen werden in ihren Konfigurationsdateien gespeichert. Bei dieser Datei handelt es sich um eine XML-Datei, welche mit der Dateiendung “.vbox” gespeichert wird.

2.3.2 VBoxManage

Das Kommandozeilentool von VirtualBox nennt sich *VBoxManage*, welches automatisch mit VirtualBox installiert wird. Unter Windows wird eine .exe-Datei mitinstalliert, welche man auf der Kommandozeile ausführt. Auf Linux gibt es nach der Installation von VirtualBox über die Paketverwaltung den Befehl *vboxmanage* in der Shell.

Mit diesem Tool kann man alle Interaktionen mit einer virtuellen Maschine durchführen. Im Gegensatz zu VMware Workstation Pro sollte man keine Änderungen direkt in der Konfigurationsdatei der virtuellen Maschine vornehmen, da diese Änderungen überschrieben werden könnten. Stattdessen sollen alle Anpassungen der virtuellen Maschinen entweder über die Benutzeroberfläche oder mit *VBoxManage* durchgeführt werden. Um einen Überblick über die Befehle zu bekommen, gibt es von Oracle ein Manual zu *VBoxManage*, in welchem viele Befehle erklärt werden.

2.3.3 Vorteile

Zu Beginn muss erwähnt werden, dass VirtualBox, im Gegensatz zu VMware Workstation Pro, kostenfrei ist. Außerdem sind die Konfigurationsmöglichkeiten über die Kommandozeile sehr vielfältig und werden im Manual von VirtualBox ausführlich dokumentiert. Wegen des letzten Aspekts ist VirtualBox für Anwendungen wie diese ideal geeignet.

2.3.4 Nachteile

Da es sich um einen Typ-2-Hypervisor handelt, ist die Performance der virtuellen Maschinen signifikant schlechter als bei einem physischen Rechner oder einem Typ-1-Hypervisor. Außerdem ist die grafische Oberfläche im Vergleich mit VMware Workstation Pro nicht so übersichtlich gestaltet. In der untersuchten Anwendung ist das im Normalbetrieb nicht relevant, jedoch müssen zum Beispiel Änderungen der virtuellen Hardware über die Oberfläche des Hypervisors erfolgen.

2.3.5 Fazit

VirtualBox ist eine brauchbare Option, wenn man einen Typ-2-Hypervisor benötigt. Die zahlreichen Konfigurationsmöglichkeiten über die Kommandozeile sind besonders bei der geplanten Anwendung in der Diplomarbeit wichtig. Jedoch sollte die während der Diplomarbeit entstehende Software direkt in der Schule eingesetzt werden, weshalb die vorhandenen Skripte für VMware Workstation Pro geschrieben wurden.

Ein Ziel der finalen Applikation ist ein modularer Aufbau, welcher das Wechseln zwischen verschiedenen Hypervisoren erleichtert, weshalb die Anwendung im Nachhinein leicht an einen anderen Virtual-Machine-Monitor wie VirtualBox angepasst werden kann.

2.3.6 Befehlsreferenz

Funktion	Befehl
Starten einer VM	`vboxmanage startvm` <*vmname*>
Stoppen einer VM	`vboxmanage controlvm` <*vmname*> `acpipowerbutton`
Löschen einer VM	`vboxmanage unregistervm` <*vmname*> `--delete`
Klonen einer VM	`vboxmanage clonevm` <*vmname*> `--snapshot` \ <*snap-name*>
Erstellen eines Snapshots	`vboxmanage snapshot` <*vmname*> `take` <*snap-name*>
Snapshots auflisten	`vboxmanage snapshot` <*vmname*> `list`
Zurücksetzen einer VM	`vboxmanage snapshot` <*vmname*> `restore` \ <*snap-name*>
Starten im Vollbildmodus	`vboxmanage setextradata` <*vmname*> \ `GUI/Fullscreen true`
Verstecken von GUI-Elementen	`vboxmanage setextradata` <*vmname*> \ `GUI/Customizations noMenuBar,NoStatusBar` `vboxmanage setextradata` <*vmname*> \ `GUI/ShowMiniToolBar no`

Tabelle 2.3: VirtualBox Befehlsreferenz

Die Befehle sind Großteils im Manual[7] von VBoxManage zu finden, jedoch sind besonders die Befehle, welche die grafische Oberfläche anpassen, nicht darin gelistet. Es wird der allgemeine Aufbau des Befehls beschrieben, um ein ExtraData-Element zu setzten. Es werden aber der Key und der Value für die Einstellungen benötigt, welche nicht im Manual gelistet werden. Ein Beispiel für ein solches Wertepaar ist *GUI/Fullscreen* und *true* aus der Befehlsreferenz. Um die richtigen Werte zu finden, kann man entweder auf anderen Seiten recherchieren oder die Einstellungen über die grafische Oberfläche von VirtualBox durchführen und die Änderungen in der Konfigurationsdatei betrachten.

2.4 Bewertungsmatrix

Um den Funktionsumfang der Kommandozeilentools der Hypervisoren zu erfassen, wurde eine Bewertungsmatrix der verfügbaren Funktionen erstellt. Wie die einzelnen Aufgaben mit den diversen Virtual-Machine-Monitoren vollzogen werden, kann man aus der Befehlsreferenz von Hyper-V (siehe 2.2.6), VirtualBox (siehe 2.3.6) und VMware Workstation Pro (siehe 2.1.7) ablesen. Diese Tabelle zeigt ausschließlich, welche Funktionen mit den Kommandozeilenwerkzeugen der Hypervisoren durchgeführt werden können. Das heißt auch, dass es andere Möglichkeiten für das Durchführen der Funktionen geben kann, welche nicht direkt mit dem Kommandozeilentool durchgeführt werden können. Dies wird auch bei den Befehlsreferenzen der Hypervisoren genauer erläutert.

Funktion	**Hyper-V PS Module**	**VBoxManage**	**vmrun**
Starten einer VM	Ja	Ja	Ja
Stoppen einer VM	Ja	Ja	Ja
Löschen einer VM	Ja	Ja	Nein
Klonen einer VM	Ja	Ja	Ja
Erstellen eines Snapshots	Ja	Ja	Ja
Snapshots auflisten	Ja	Ja	Ja
Zurücksetzen einer VM	Ja	Ja	Ja
Starten im Vollbildmodus	Nein	Ja	Nein

Tabelle 2.4: Bewertungsmatrix essentieller Funktionen

3 Applikation

3.1 Qt Framework

3.1.1 Warum Qt?

Die Entscheidung für das Qt Framework fiel, da wir in der Schule sowohl Linux als auch Windows Computer im Einsatz haben und Qt eine plattformübergreifende Entwicklung von Programmen unterstützt. Als eines der weitverbreitetsten Frameworks gibt es auch eine Vielzahl an APIs und Libraries, welche gut dokumentiert und einfach zu verwenden sind. Auch die große Entwickler Gemeinschaft, hat sich bei spezifischen Fragen oftmals als hilfreich erwiesen.

3.1.2 Was ist Qt?

Qt ist ein Anwendungs-Framework zur plattformübergreifenden Entwicklung von Programmen und grafischen Benutzeroberflächen. Es unterstützt unter anderem Linux, Windows, OS X, Android, iOS, BlackBerry und viele weitere Betriebssysteme. Qt ist keine Programmiersprache an sich, es ist ein Framework geschrieben in C++. Für Qt gibt es auch Anbindungen für andere Programmiersprachen, unter anderem für Python (PyQt), Ruby (QtRuby), C# (QtSharp), Perl (PerlQt) und Java (Qt Jambi). In unserer Diplomarbeit verwenden wir ausschließlich das native Qt, welches eine Programmierung in C++ voraussetzt. [5]

3.1.3 Lizensierung

Qt ist eine Software mit einem dualen Lizenzsystem. Das bedeutet, es kann entweder unter einer proprietären Lizenz kommerziell genutzt werden oder auch für Freie Software mit der GNU General Public License oder der GNU Lesser General Public License. Da mit Vsistent keine kommerzielle Software entwickelt wurde konnten das Qt Framework für diese Diplomarbeit kostenlos verwendet werden. [1]

3.1.4 Qt Creator

Qt Development Frameworks bietet, mit dem *Qt Creator*, eine eigene integrierte Entwicklungsumgebung für Linux, OS X und Windows an. Zu den wichtigsten Eigenschaften dieser IDE gehören unter anderem ein Quellcode-Editor mit Syntaxhervorhebung und Autovervollständigung, eine strukturierte Projektverwaltung, ein GUI-Designer, ein Debugger und eine detaillierte Dokumentation.

3.1.5 Signals and Slots

In Qt gibt es den *Signals-and-Slots*-Mechanismus, welcher zur Kommunikation zwischen zwei Objekten dient. Dabei können Signale ausgesendet werden, nachdem eine Änderung vollzogen wurde, von welcher andere Objekte informiert werden sollen. Damit bei dem Aussenden eines Signales auch etwas ausgeführt wird, muss ein Signal mit einem Slot verbunden werden. Slots sind Methoden, die man durch ein Signal aufrufen kann. Man kann sie aber auch wie herkömmliche C++-Methoden aufrufen [4].

Ein typisches Beispiel für einen solchen Mechanismus ist ein Knopfdruck in der grafischen Oberfläche, bei welchem das Fenster geschlossen werden soll. Das Verbinden des Signales mit dem Slot sieht folgendermaßen aus, wenn man das Beispiel realisieren möchte:

```
1 connect(button, SIGNAL(clicked()), window, SLOT(close()));
```

Listing 3.1: Beispiel: Signals and Slots

3.1.6 QProcess

Die einzelnen Skripte werden im C++ Code mit der Klasse *QProcess* aufgerufen. Mit einem solchen Prozess kann man ein Skript folgendermaßen aufrufen:

```
1 QProcess process;
2 process.start(program, arguments);
```

Listing 3.2: Beispiel: Starten eines *QProcess*

Um Skripte auszuführen, werden die Skripte mit dem Programm ausgeführt. Bei den Parametern handelt es sich um eine QStringList, welche als erstes Element den Namen des Skriptes beinhaltet. Alle übrigen Elemente sind die Argumente, mit welchen man das Skript aufrufen möchte.

3.2 Benutzer-Applikation

3.2.1 Grafische Oberfläche

Ursprünglich war es geplant alle vorhanden virtuellen Maschinen, unabhängig von ihrem Zustand darzustellen, jedoch ging dadurch die Übersichtlichkeit unseres Programmes verloren. Daher werden nur mehr die virtuellen Maschinen mit dem gleichen Status gleichzeitig angezeigt. Somit kann sich der Benutzer jetzt entscheiden ob er die Master-VMs, also virtuelle Maschinen, von denen noch kein Klon generiert wurde, geklonte VMs oder laufende VMs angezeigt haben möchte.

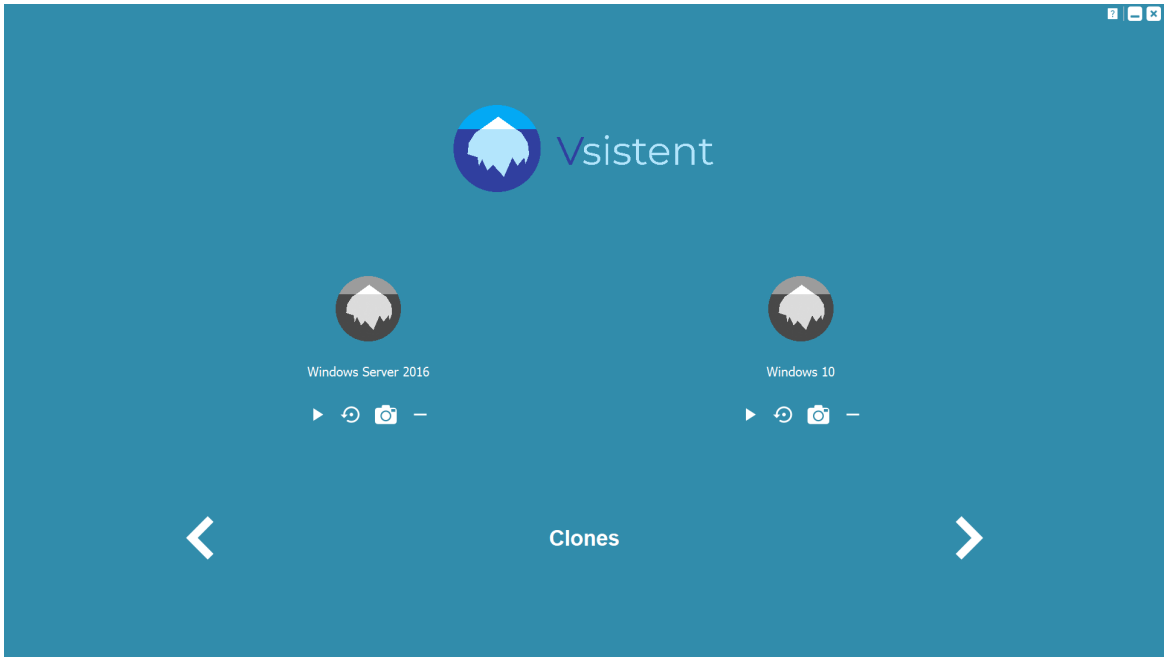


Abbildung 3.1: Grafische Oberfläche - Benutzer-Applikation

Je nachdem, in welchem Status sich die virtuellen Maschinen gerade befinden, unterscheiden sich auch die für sie anwendbaren Funktionen. So ist es für Master-VMs möglich, einen neuen Klon zu erstellen oder die virtuelle Maschine zu starten. Um das direkte Starten der Master-VM zu ermöglichen wird im Hintergrund ein Klon erstellt. Geklonte VMs können ebenfalls direkt gestartet werden. Zusätzlich kann ein Snapshot für jede einzelne, geklonte virtuelle Maschine erstellt werden oder der zuletzt erstellte

Snapshot, der virtuellen Maschine, geladen werden. Auch ein bereits existierender Klon kann über die grafische Oberfläche entfernt werden. Laufende virtuelle Maschinen können auch über die Benutzeroberfläche heruntergefahren werden.

3.2.2 Umsetzung

3.2.2.1 Layout

Um einen übersichtlichen Aufbau zu gewährleisten, haben wir uns dazu entschieden, die gesamte grafische Oberfläche mit einem vertikalen Layout zu versehen. Diesem wurden drei Reihen mit gleicher Höhe hinzugefügt. In der obersten Reihe befindet sich zentriert das Logo mit dem klassischen „Vsistent“ Schriftzug.

Im mittleren Bereich der Oberfläche befinden sich ein bis fünf virtuelle Maschinen, welche durch ein Icon gekennzeichnet werden. Unter diesem Icon befindet sich der Name der virtuellen Maschine mit den dazugehörigen Knöpfen, welche sich jedoch je nach Status der virtuellen Maschine unterscheiden. Sollten mehr als fünf virtuelle Maschinen mit demselben Status vorhanden sein, erscheinen zwei zusätzliche pfeilförmige Knöpfe, welche das Anzeigen der weiteren virtuellen Maschinen ermöglichen.

Im untersten Drittel befindet sich ein Schriftzug, welcher den Status der angezeigt virtuellen Maschinen beinhaltet. Um virtuelle Maschinen von einem anderen Status anzeigen zu lassen, sind links und rechts des Textes zwei pfeilförmige Knöpfe, um diesen zu wechseln. Sollte es beispielsweise keine geklonten und laufenden virtuellen Maschinen geben, werden diese Knöpfe nicht angezeigt.

Um eine gute Übersicht zu gewährleisten ist es nicht möglich, die grafische Benutzeroberfläche in einem Fenster Modus laufen zu lassen. Dazu gibt es das *resizeEvent* von Qt, welches eine automatische Änderung der Größe erkennt und das Fenster wieder in den Vollbildmodus versetzt, ohne dass der Benutzer dies bemerkt. Durch das Minimieren in die Taskleiste und das erneute Öffnen danach, kommt es zu so einer automatischen Änderung der Größe. Um dies zu verhindern wurde folgender Event-Handler implementiert:

```
1 void Vsistent::resizeEvent(QResizeEvent *event){
2     QMainWindow::resizeEvent(event);
3     this->setWindowState(Qt::WindowMaximized);
4     this->setWindowState(Qt::WindowFullScreen);
5 }
```

Listing 3.3: vsistent.cpp: 435-439

3.2.2.2 Anzeigen der virtuellen Maschinen

Auch intern werden die virtuellen Maschinen in drei Listen gegliedert. Jeder dieser Listen bildet alle virtuellen Maschinen eines Status ab. In diesen Listen wird für jede virtuelle Maschine der gesamte Pfad zur jeweiligen Konfigurationsdatei gespeichert, da dieser für die in Kapitel 3.2.4 beschriebenen modularen Skripte benötigt wird. Beim Starten des Programmes werden diese Listen automatisch befüllt, in dem der angegebene Pfad nach Konfigurationsdateien von virtuellen Maschinen durchsucht wird. Einer dieser drei Listen ist immer die dargestellte Liste. Beim Ändern des angezeigten Status wird auch die dargestellte Liste dementsprechend geändert. Standardmäßig ist die Liste der bereits existierenden Klons die dargestellte Liste. Gibt es jedoch beim Starten des Programmes noch keinen Klon, werden die vorhandenen Master-VMs angezeigt.

Um Objekte mit Qt grafisch darzustellen, muss ein Qt-Objekt in einem Qt-Layout hinzugefügt werden. Aus diesem Grund wurde in unserem Programm für jedes Element in der *listShow*-Liste ein neues Qt-Objekt mit einem vertikalen Layout erstellt. In der obersten Reihe wird ein Icon für die virtuelle Maschine hinzugefügt.

Darunter wird der Name der VM angezeigt. Dafür gibt es sogenannte *QLabel*, welche einen *QString* als Text zugeordnet haben. Um den Namen der virtuellen Maschinen als *QString* zu erhalten, wurde die Methode *convertPathToName* geschrieben (Siehe 3.2.6.1 convertPathToName).

In der untersten Reihe wird dem vertikalen Layout ein neues horizontales Layout hinzugefügt, welchem verschiedene *ScriptButton* hinzugefügt werden. So wird einer Master-VM ein Knopf für das Klonen der virtuellen Maschine und ein Knopf für das Klonen und Starten dieser hinzugefügt. Einer geklonten virtuellen Maschine wird ein Knopf zum Starten, einer zum Erstellen eines Snapshots, einer zum Laden des letzten Snapshots und einer zum Entfernen des Klons hinzugefügt. Laufende virtuelle Maschinen erhalten nur einen Knopf, dieser dient zum Herunterfahren.

Hier ein exemplarischer Ausschnitt des Codes, der die benötigten Knöpfe zu einer Master-VM hinzufügt:

```
1 if(active=="master"){
2     scriptButton *cloneAndStartB = new scriptButton(
3         cloneAndStartScript,
4         name,
5         QIcon(":/imgs/icon/imgs/icon/start.svg"),
6         "", sh, clone);
7     cloneAndStartB->setStyleSheet(
8         "padding:5px;"
9         "border-style:none;"
```

```
10          );
11      cloneAndStartB->setIconSize(QSize(size/2,size/2));
12      cloneAndStartB->setMaximumWidth(size*2);
13      scriptButton *cloneB = new scriptButton(cloneScript,
           name, QIcon(":/imgs/icon/imgs/icon/add.svg"), "", sh
           , clone);
14      cloneB->setStyleSheet(
15          "padding:5px;"
16          "border-style:none;"
17          );
18      connect(cloneB, SIGNAL(clicked()),this,SLOT(toClones()
           ));
19      cloneB->setIconSize(QSize(size/2,size/2));
20      cloneB->setMaximumWidth(size*2);
21      bLayout->addWidget(cloneB);
22      bLayout->addWidget(cloneAndStartB);
23      }
```

Listing 3.4: vsistent.cpp: 211-227

3.2.3 Linux-Support

Wie bereits oben erwähnt, sollte die Benutzeroberfläche auf Linux- und auf Windows-Hostsystemen verwendet werden können. Die grafische Oberfläche, welche den Benutzern die Funktionen zur Verfügung stellt, sollte auf beiden Betriebssystemen verwendbar sein. Die Unterstützung von beiden Plattformen bei der Funktion der Anwendung kann leicht realisiert werden, da die Kommunikation mit dem Hypervisor in Skripte ausgelagert wurde. Damit die Skripte bei den verschiedenen Systemen richtig aufgerufen werden, muss der Typ des Betriebssystems abgefragt werden. Dies wird in Qt mit folgendem Befehl durchgeführt:

```
1 QSysInfo::kernelType().toLocal8Bit().data()
```

Listing 3.5: Auslesen des Betriebssystems

Wenn die Applikation auf einem Windows 10 Host läuft, liefert diese Zeile den String „winnt“. Sollte es sich dagegen um einen Linux Mint Host 19 handeln, ist der Rückgabewert „linux“. Anhand dieser Information werden die Parameter eines *QProcess* richtig eingestellt werden.

3.2.4 Skripte

Bei den Skripten handelt es sich um PowerShell-Skripte bei einem Windows-Host, beziehungsweise um Bash-Skripte bei einem Linux-Hostrechner. Die einzelnen Skripte führen jeweils nur eine oder wenige Interaktionen mit dem Hypervisor durch. Die einzelnen Skripte führen die folgenden Funktionen durch:

Name	Parameter	Funktion
clone	name, clone	Klont die VM *name*
cloneAndStart	name, clone	Klont die VM *name* und startet den Klon
remove	name	Löscht die VM *name*
reset	name	Setzt die VM *name* auf den letzten Snapshot zurück
running	-	Gibt eine Liste laufenden VMs zurück
start	name	Startet die VM *name*
stop	name	Fährt die VM *name* herunter
takeSnapshot	name	Erzeugt einen Snapshot der VM *name*

Tabelle 3.1: Skripte der Benutzeroberfläche

Allgemein ist bei den Skripten zu beachten, dass die Pfade zu den Skripten in der Datei *vsistent.h* definiert werden müssen. Unter Windows wird den Skripten noch der Pfad zu der Datei *vmrun.exe* als Argument mitgegeben. Dadurch kann der Pfad zu dieser Datei zentral in einer Header-Datei gespeichert werden und muss nicht in jedem Skript definiert werden. Unter Linux muss ein solches Argument nicht mitgegeben werden, da *vmrun* nach der Installation von VMware Workstation Pro als Shell-Befehl zur Verfügung steht.

Im Speziellen ist bei dem *clone*-Skript noch zu erwähnen, dass nach dem Klonen einer virtuellen Maschine der Klon automatisch konfiguriert wird. Diese Konfigurationen sind das Wechseln in den Vollbildmodus und das Verhindern der Pop-Up-Meldungen von VMware Workstation Pro. Anschließend wird automatisch ein Snapshot mit dem Namen „Snapshot created by Vsistent“ erstellt. Außerdem wird bei dem *takeSnapshot*-Skript der Name des Snapshots aus „Vsistent Snapshot“ und der Uhrzeit beim Erstellen zusammengesetzt.

3.2.5 Funktion

Die Interaktionen mit dem Hypervisor werden bei diesem Programm mit zwei Klassen realisiert, der *Scripthandler*- und der *scriptButton*-Klasse. Der Ablauf wird in folgendem Diagramm dargestellt:

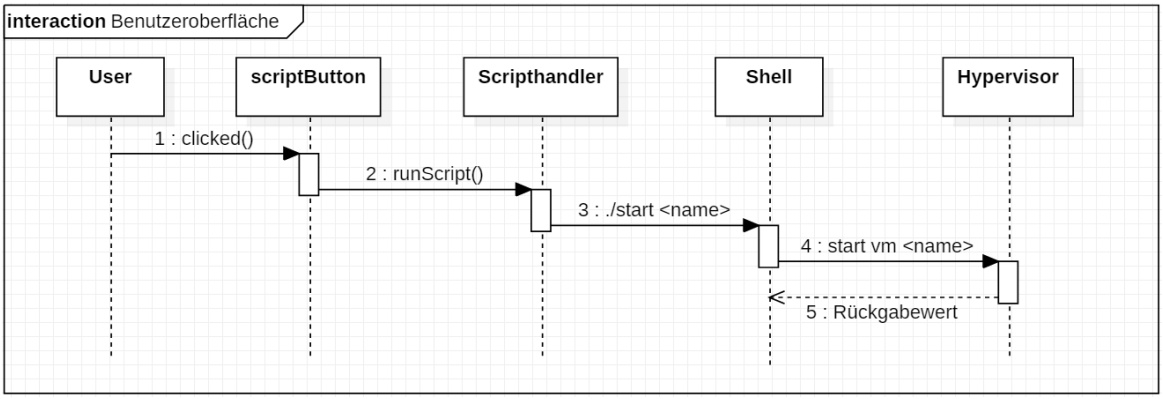


Abbildung 3.2: Ablaufdiagramm der Benutzeroberfläche

3.2.6 Klassen

3.2.6.1 Vsistent

Allgemeines Diese Klasse dient hauptsächlich zum Anzeigen der grafischen Oberfläche, wie es oben beschrieben wurde. In diesem Unterkapitel werden die Methoden beschrieben, welche Informationen über die virtuellen Maschinen bereitstellen.

getVMs Die Master-VMs und die geklonten virtuellen Maschinen werden aus Ordnern ausgelesen. Die Pfade zu den entsprechenden Ordnern werden in der Header-Datei definiert. Das Auslesen der Pfade zu den Konfigurationsdateien der virtuellen Maschinen wird mit Qt folgendermaßen realisiert:

```
1 QStringList vms;
2 QDirIterator it(path,
3     QStringList() << "*.vmx",
4     QDir::Files,
5     QDirIterator::Subdirectories);
6 while (it.hasNext()) {
7     vms.append(it.next());
8 }
```

Listing 3.6: Auslesen der .vmx-Datein

Die Methode *getVMs* gibt eine Liste zurück, die alle Pfade der virtuellen Maschinen beinhaltet, welche sich unter dem Pfad *path* befinden.

getRunningVMs Die Methode liest die laufenden virtuellen Maschinen aus. Dies geschieht mit Hilfe eines Skriptes, welches über einen *QProcess* ausgeführt wird. Der Standard Output des Prozesses sind die Pfade der laufenden virtuellen Maschinen. Dieser ist ein String mit Zeilenumbrüchen, welcher in eine Liste geteilt werden muss. Das Auslesen der Ausgabe des Skriptes und das Teilen des Strings wird wie folgt umgesetzt:

```
1 process.start(powershell, arguments);
2 QString out = process.readAllStandardOutput();
3 QStringList vms = out.split(QRegExp("[\r\n]"), QString::
    SkipEmptyParts);
```

Listing 3.7: Auslesen der laufenden VMs

In diesem Beispiel wird ein Skript auf Windows 10 ausgeführt, weshalb die Variable *powershell* der Pfad zur *powershell.exe*-Datei als *QString* ist.

convertPathToName Die oben genannten Methoden geben den absoluten Pfad zu den Konfigurationsdateien der virtuellen Maschinen zurück, welcher für von *vmrun* verwendet wird. Jedoch ist es für das Anzeigen der virtuellen Maschinen nicht ideal, weshalb diese Methode den Dateinamen ohne Dateiendung aus dem Pfad extrahiert. Dieser Name wird dann in der Benutzeroberfläche angezeigt, man kann dafür aber auch aussagekräftigere Namen für einzelne virtuelle Maschinen festlegen.

3.2.6.2 Scripthandler

Diese Klasse dient zum Ausführen von PowerShell- oder Shell-Skripten. Diese Funktion wird durch den Slot *runScript* realisiert, welche einen *QProcess* startet, der das gewünschte Skript ausführt. Die Parameter dieser Methode sind der Name des Skriptes, der Name der virtuellen Maschine und der Name des Klons, falls ein Neuer erstellt werden sollte.

Damit der grafischen Oberfläche mitgeteilt werden kann, wann ein Skript ausgeführt wurde, wird am Ende der Methode das Signal *finished()* emittiert. Dieses Signal dient nicht zur Ausführung von eigenen Funktionen, sondern wird lediglich mit einem Slot verbunden, welcher die Elemente der grafischen Oberfläche aktualisiert. Das *finished*-Signal wurde eingebaut, da sich die Elemente der grafischen Oberfläche nach dem Aufruf der meisten Skripte verändern. Die einzigen Skripte, welche zu keinen Änderungen in der Oberfläche führen, sind die Skripte *reset* und *takeSnapshot*.

Das Anzeigen der laufenden virtuellen Maschinen, wie im Unterkapitel 3.2.6.1 beschrieben, wird nicht mit dieser Klasse durchgeführt, da der Slot *runScript* die Ausgabe der Skripte nicht zurück gibt. Das Auslesen des Standard-Outputs eines *QProcess* ist in dem Slot auch nur über Umwege möglich, da er in eine Logdatei geschrieben wird. Das Logging wird im Unterkapitel 3.2.7 genauer erklärt.

3.2.6.3 scriptButton

Die Klasse *scriptButton* ist eine Erweiterung der *QPushButton*-Klasse. Diese Klasse wurde geschrieben, damit ein Skript mit den richtigen Argumenten aufgerufen wird. Um dies zu bewerkstelligen, nimmt diese Klasse zusätzlich den Namen der virtuellen Maschine, den Namen eines Skriptes und eventuell den Namen des Klons als zusätzliche Argumente. Der wichtigste Bestandteil dieser Klasse ist der Slot *runScript()*.

```
1 void scriptButton::runScript() {
2     emit sh->runScript(this->script, this->vm, this->clone
         );
3 }
```

Listing 3.8: scriptbutton.cpp: runScript()

Durch den Slot *runScript()* wird die gleichnamige Methode in der *Scripthandler*-Klasse aufgerufen. Dieser Methode werden aber die Parameter aus dem Konstruktor mitgegeben, damit das gewünschte Skript mit den richtigen Argumenten aufgerufen wird. Damit der Slot auch bei einem Knopfdruck aufgerufen wird, wird die Verbindung zu dem Signal *clicked()* der *QPushButton*-Klasse im Konstruktor hergestellt.

```
1 connect(this, SIGNAL(clicked()), this, SLOT(runScript()));
```

Listing 3.9: scriptbutton.cpp: Verbinden des Slots

3.2.7 Logging

Um das Finden und Beheben von Fehlern zu vereinfachen, wird der Standard-Output und der Fehler-Output des Prozesses, welcher das Skript aufruft, in eine Logdatei geschrieben. Das Bestimmen der Logdatei wird mit folgenden Befehlen realisiert:

```
1 process.setStandardOutputFile(logfile, QIODevice::Append);
2 process.setStandardErrorFile(logfile, QIODevice::Append);
```

Listing 3.10: scripthandler.cpp: Spezifizieren der Logdatei

Durch diesen Einbau von Logging wird zwar der gesamte Output von den Skripten in der Logdatei gespeichert, jedoch gibt es keine Zeitstempel oder Unterschiede zwischen Fehlern und der gewöhnlichen Ausgabe des Prozesses. Außerdem werden Fehlermeldungen nicht von Rückgabewerten von Skripten unterschieden.

Um Fehler des *QProcess* speziell zu klassifizieren, kann man für das Logging zwei Slots schreiben, welche ausgeführt werden, wenn der *QProcess* das eingebaute Signal *readyReadStandardOutput* oder *readyReadStandardError* emittiert. Dies geschieht immer, wenn es neue Meldungen vom Prozess gibt. In diesen Slots kann man zu den Nachrichten einen Zeitstempel und eine Klassifizierung hinzufügen.

Diese Art des Loggings wurde aber nicht gewählt, da Zeitstempel und das Markieren von Fehlern für den geplanten Anwendungsfall nicht von hoher Priorität sind. Außerdem war es aus zeitlichen Gründen nicht möglich die umfangreichere Variante des Loggings umzusetzen.

3.2.8 Untersuchte Ansätze

3.2.8.1 QPushButton

Ursprünglich wurde ohne der Klasse *scriptButton* gearbeitet, jedoch musste für jeden Knopf eine eigene Methode geschrieben werden, als mit der *QPushButton*-Klasse gearbeitet wurde. Dies machte den Code sehr redundant und unübersichtlich, weshalb dadurch auch das Finden und Korrigieren von Fehlern erschwert wurde. Außerdem müsste in einem solchen Aufbau für jede neue Funktion der Benutzeroberfläche zusätzlich zu einem neuen Skript eine Methode geschrieben werden, die dieses Skript aufruft.

3.2.8.2 Skriptaufruf mit system()

Ursprünglich wurden die Skripte im C++ Code mit dem Befehl *system()* aufgerufen. Jedoch gibt es bei diesem Befehl zwei wesentliche Nachteile für die Anwendung in der Benutzeroberfläche. Zum einen wird bei dem Aufruf eines Skriptes immer ein Kommandozeilenfenster angezeigt. Dies kann besonders bei den periodischen Aktualisierungen der grafischen Oberfläche irritierend oder störend sein.

Zum anderen ist das Auslesen des Outputs der Skripte nicht direkt möglich. Man kann zwar eine Methode schreiben, die den Rückgabewert eines Skriptes bekommt. Das Verwenden eines *QProcess* ist jedoch einfacher und übersichtlicher, weshalb diese Variante gewählt wurde.

3.2.9 Installation

Um die Benutzer-Applikation lokal auf einem physischen Rechner zu installieren, wurde mit Hilfe des Programmes *InstallForge* ein Installations-Wizard erstellt, welcher standardmäßig alle benötigten Dateien nach *C:\Program Files (x86)\vsistent* kopiert. Ohne einer Änderung im Source-Code ist es notwendig, dass die Master-VMs unter *C:\VM* abgespeichert sind und *vmrun* unter *C:\Program Files (x86)\VMware\VMware Workstation* installiert ist.

3.3 AdministratorInnen-Applikation

3.3.1 Klassen

3.3.1.1 MainWindow

In dieser Klasse wird die grafische Oberfläche generiert. Das bedeutet, dass alle GUI-Elemente hier dem Fenster hinzugefügt werden. Außerdem werden in dieser Klasse die Nachrichten vom Server so dekodiert, dass die Applikation den Status den einzelnen virtuellen Maschinen zuordnen kann.

3.3.1.2 ButtonHandler

Die Aufgabe der Klasse *ButtonHandler* ist es, einen vordefinierten Request für eine bestimmte virtuelle Maschine an den Server zu senden, nachdem ein bestimmter Button betätigt wurde. Ein *ButtonHandler* wird mit folgenen Zeilen konfiguriert und mit einem Button verknüpft:

```
1 ButtonHandler *bAdd = new ButtonHandler(deployAction + v);
2 QObject::connect(add, SIGNAL(clicked()),
3    bAdd, SLOT(buttonClicked()));
4 QObject::connect(bAdd, SIGNAL(wasChanged()),
5    this, SLOT(getVms()));
```

Listing 3.11: mainwindow.cpp:81-83

Dieses Beispiel zeigt einen Button, der eine virtuelle Maschine in die Verteilerwarteschlange hinzufügen soll. In der ersten Zeile wird hierbei der *ButtonHandler* mit der gewünschten Aktion erstellt. *„deployAction“* ist hier die Aktion und *„v“* der Name der Gast-Maschine, für die diese Aktion getätigt werden soll.

In Zeile zwei wird daraufhin der *ButtonHandler* mit der Aktivierung des Buttons verknüpft. In Zeile vier wird, sobald der *ButtonHandler* mit seinen Aktionen fertig ist, die grafische Oberfläche neu gezeichnet, sodass alle Änderungen, die am Server durchgeführt wurden, sofort in der GUI sichtbar gemacht werden.

3.3.1.3 RequestHandler

Hier werden die HTTP-Requests, wie im Kapitel 4.2 Communications beschrieben, durchgeführt. Wie man einen solchen *RequestHandler* erstellt, wird anhand diesem Beispiel erklärt:

```
1 RequestHandler *a = new RequestHandler(server, action,
     this);
2 a->requestData();
```

Listing 3.12: buttonhandler.cpp:9-10

In diesem Beispiel wird zuerst der *RequestHandler* mit den Parametern *„server“*, für die IP des Servers, und *„action“*, der den HTTP-Request beinhaltet, erstellt.

In der zweiten Zeile wird dieser nun mit der Methode *„requestData()“* gestartet. Sobald der *RequestHandler* vom Server eine Antwort erhalten hat, löst dieser ein Signal mit der Antwort des Servers aus:

```
1 emit vmsGotDelivered(vms);
```

Listing 3.13: requesthandler.cpp:27

Sollte dieses Signal nun „emittiert“ werden, wird die grafische Oberfläche neugezeichnet. Somit ist die GUI zu jedem Zeitpunkt aktuell.

3.4 LehrerInnen-Applikation

3.4.1 Klassen

3.4.1.1 CreateVM

Sobald eine Lehrerin oder ein Lehrer eine virtuelle Maschine erstellen möchte, betätigt die Lehrkraft die „VM erstellen“-Schaltfläche und es wird ein Fenster angezeigt. In diesem müssen die Metadaten eingegeben werden. Dieses Fenster wird von der Klasse CreateVM verwirklicht. Die Metadaten, die vom Lehrpersonal eingegeben werden müssen, werden in der folgenden Tabelle beschrieben:

Name	Parameter
vmName	Der Name der virtuellen Maschine.
vmDescription	Eine aussagekräftige Beschreibung, in der die Lehrkraft erklären soll, wieso diese VM erstellt wurde.
authorName	Der Name des Lehrers.
authorMail	Die E-Mail-Adresse des Lehrers.
vmMaster	Welche Master-VM verwendet wurde, um den Klon zu erstellen.
vmSubjects	Für welche Schulfächer diese virtuelle Maschine gedacht ist.
vmPrograms	Welche Programme auf der VM installiert werden.

Tabelle 3.2: Die Metadaten-Felder einer VM

Hat die Lehrkraft alle Metadaten eingestellt, wird zuerst die Master-VM geklont und danach wird eine JSON-Datei erstellt, die diese Daten enthält. Das Erstellen der Datei funktioniert wie folgt:

```
1 QJsonObject root;
2 root.insert("vmName", QJsonValue::fromVariant((vmNameInput
    ->text()).toUtf8().constData()));
```

```
3 [...]
4 QJsonDocument doc(root);
5 [...]
6 QFile jsonFile(clones + vmNameInput->text() + "\\metadata.
    json");
7 jsonFile.open(QFile::WriteOnly);
8 jsonFile.write(doc.toJson());
```

Listing 3.14: createvm.cpp:147-162

Hierbei werden mit der zweiten Zeile die benötigten Werte in das *QJsonObject* eingefügt. Diese werden dann in einem *QJsonDocument* gebündelt und zu guter Letzt mit den Zeilen sechs bis acht in die Datei *metadata.json* geschrieben.

3.4.1.2 EditVM

Hat eine Lehrkraft eine virtuelle Maschine erstellt und möchte einzelne Metadaten wieder ändern, kann diese dies mit der „Bearbeiten“-Schaltfläche erledigen. Diese Schaltfläche befindet sich neben den einzelnen Gast-Maschinen. Sollte diese Schaltfläche betätigt werden, wird mithilfe der *EditVM*-Klasse eine grafische Oberfläche generiert. Hier gibt es Unterschiede zur GUI von *CreateVM*: Erstens sind die Eingabefelder bereits mit den zuvor konfigurierten Metadaten gefüllt und zweitens sind die Felder *vmName* und *vmMaster* schreibgeschützt, da es hier nicht sinnvoll wäre, sie zu ändern.

3.4.1.3 SubmitHandler

Wenn alle Programme auf der virtuellen Maschine installiert sind und die Metadaten richtig konfiguriert sind, kann eine Lehrkraft diese virtuelle Maschine „einreichen“. Dazu muss diese die „Einreichen“-Schaltfläche betätigen. Daraufhin wird die Gast-Maschine mithilfe der Klasse *SubmitHandler* automatisch auf den Server hochgeladen. Zum Schluss wird dem Server, wie im Kapitel 4.2 Communications beschrieben, mitgeteilt, dass sich die virtuelle Maschine nun am Server befindet.

4 Backend & Communications

4.1 Backend – Server

4.1.1 Klassen/Dateien

4.1.1.1 Allgemeines

Im Gegensatz zu der grafischen Oberfläche wurde bei dem Server die Programmiersprache *Python* eingesetzt. Python wurde gewählt, da bei dem Server keine grafischen Elemente benötigt werden und mehr Erfahrung mit dieser Sprache besteht.

4.1.1.2 Server

Die *server.py* Datei ist die Datei, die ausgeführt werden muss, um den Server zu starten. In dieser werden die HTTP-Requests behandelt. Diese Datei startet also alle benötigten Threads und Methoden, die für den Betrieb des Servers benötigt werden.

4.1.1.3 Controller

Die einzige Aufgabe der Klasse *Controller* ist es, die Ordner „pending“ und „deploying“ zu überprüfen, ob sich in diesen Ordnern virtuelle Maschinen befinden.

Sollte keine virtuelle Maschine im Ordner „deploying“ sein, wird in Folge der Ordner „pending“ überprüft. Sollte hier eine VM enthalten sein, wird diese nach „deploying“ verschoben und *SysTNegotiator* gestartet.

4.1.1.4 variables_and_methods

Alle Variablen und Methoden, die von mehreren Klassen benötigt werden, sind in dieser Datei zu finden. Dies hat den Vorteil, dass man die Variablen an einem Ort hat.

Beispiele für solche Variablen sind die IP und der Port des MQTT-Brokers.

Des Weiteren sind hier die Methoden enthalten, die von mehreren Klassen benötigt werden. Dies sind Methoden wie das Auslesen der Metadaten oder auch das Senden von E-Mails.

4.1.1.5 SysTNegotiator

Hier wird ein MQTT-Client[3] ausgeführt, der mit dem *„SystemTrojaner“* kommuniziert. Neben der Kommunikation ist dieser dafür zuständig, dass die virtuellen Maschinen verteilt werden. Das heißt, wenn Fehler entstehen sollten, werden diese behandelt und der Vorgang erneut probiert.

Gestartet wird der Client in einer Endlosschleife als Thread, damit er einkommende Nachrichten nicht verräumt. Die möglichen Nachrichten werden im Kapitel 4.2.5 näher erläutert. Ein weiterer Grund für den Start als Endlosschleife ist, dass der Server dann nicht abstürzt, sollte es zu Problemen mit dem MQTT-Client kommen.

4.2 Communications

4.2.1 Allgemein

In diesem Kapitel werden die einzelnen Kommunikationsabläufe zwischen den einzelnen Komponenten der Vsistent-Lösung beschrieben. Da diese sehr komplex sind wurde folgende Grafik angefertigt:

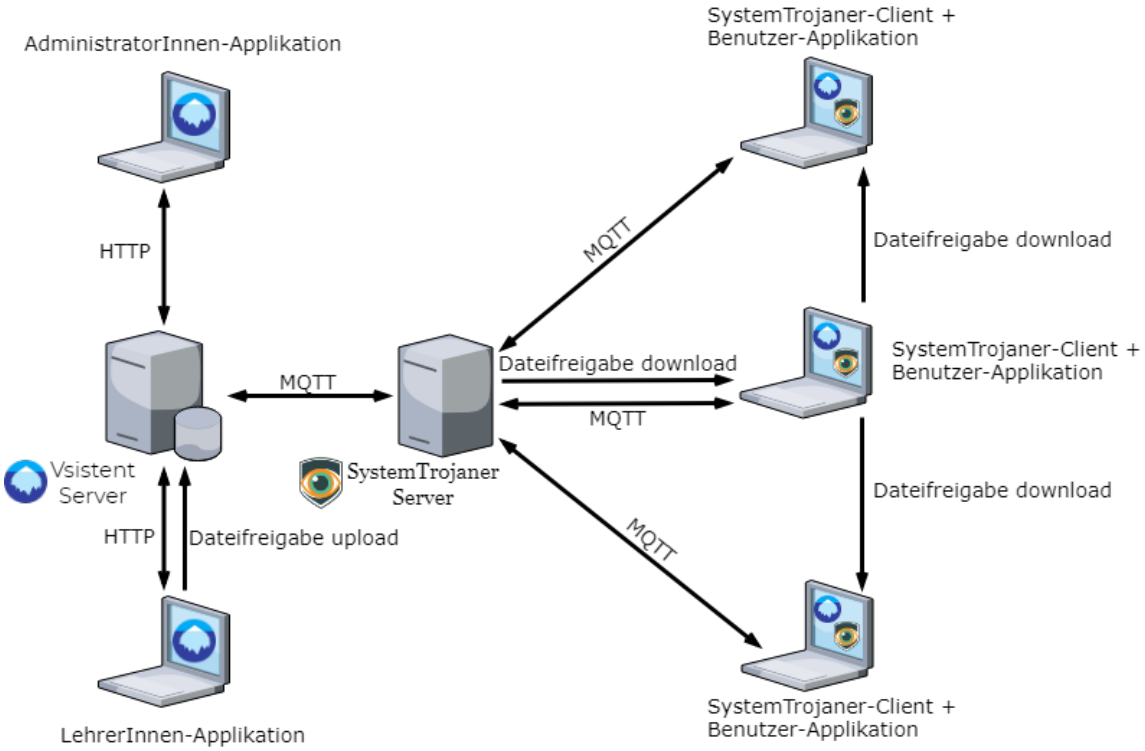


Abbildung 4.1: Diagramm der Kommunikationen der Vsistent-Lösung

Diese Grafik zeigt die einzelnen Kommunikationswege. Diese werden nun in den folgenden Kapiteln erklärt.

4.2.2 Protokolle

Im Backend wurden zwei Protokolle für den Informationsaustausch eingesetzt. Dies wären zum einen MQTT für die Kommunikation mit dem *„SystemTrojaner“*-Server. Mehr zu diesem Protokoll finden Sie unter [3]. Das andere Protokoll, dass eingesetzt wurde, ist HTTP. Dieses wurde für den Informationsaustausch zwischen den Vsistent-Applikationen verwendet. Hierbei ist anzumerken, dass keine Möglichkeit zur Authentifizierung eingebaut wurde. Des Weiteren ist dieser Informationskanal nicht verschlüsselt. Mehr zu dem Protokoll HTTP ist unter [12] zu finden.

4.2.3 Server – LehrerInnen-Applikation

Die LehrerInnen-Applikation wird für das Erstellen von den einzelnen Master-VMs benötigt. Diese muss mit dem Server für auf das Hochladen der virtuellen Maschinen kommunizieren. Des Weiteren muss die LehrerInnen-Applikation den Server verständigen, wenn sie mit dem Hochladen fertig ist.

Um dem Server mitzuteilen, dass die virtuelle Maschine fertig hochgeladen ist, sendet die LehrerInnen-Applikation einen HTTP-Request auf */teacher/submit/<Name der VM>*. Der Server reagiert auf einen solchen Request, indem er die eingereichte VM aus der Dateifreigabe in den „submitted"-Ordner verschiebt. Durch diesen Ordner können Administratorinnen und Administratoren auf einen Blick sehen, welche virtuellen Maschinen von Lehrerinnen und Lehrern eingereicht wurden. Jedoch sind in diesem Ordner nur jene Gast-Maschinen, die noch nicht verteilt oder zum Verteilen freigegeben wurden.

Nachdem die virtuelle Maschine verschoben wurde, wird an die Administratorinnen, Administratoren und die Lehrperson, die die Gast-Maschine eingereicht hat, jeweils eine Mail versandt. In dieser wird erstens der Lehrkraft mitgeteilt, dass ihre virtuelle Maschine nun am Server ist und demnächst von einer Administratorin oder einem Administratoren begutachtet wird. Zweitens bekommen die Administratorinnen und Administratoren eine Mail, in der steht, dass jene Lehrperson eine neue virtuelle Maschine eingereicht hat. Des Weiteren werden hier die gesamten Metadaten mitgesandt, die eine Lehrkraft eingetragen hat.

Wenn dies geschehen ist, antwortet der Server mit der Nachricht *„ok"* der LehrerInnen-Applikation. Daraufhin verschiebt jenes Programm die virtuelle Maschine lokal vom „clones"-Ordner, in dem die Maschinen enthalten sind, die das Lehrpersonal erstellt hat, in den „submitted"-Ordner, in dem jene sind, die bereits eingereicht wurden.

4.2.4 Server – AdministratorInnen-Applikation

Die AdministratorInnen-Applikation wird für das Management der virtuellen Maschinen auf dem Server und den einzelnen Clients benötigt. Diese Applikation ist im Grunde nur eine Oberfläche für den Server. Das heißt, dass man mit dieser den Server, aber nicht den lokalen Client, verwaltet.

Die erste Aktion, die implementiert wurde, fragt den Server nach dem aktuellen Stand der virtuellen Maschinen. Hierzu sendet die AdministratorInnen-Applikation ein HTTP-Request auf */admin*. Geantwortet wird auf diese Anfrage mit allen Gast-Maschinen, die momentan am Server vorhanden sind. Des Weiteren wird der Status der jeweiligen Maschinen mitgegeben. Das heißt, ob diese bereits verteilt ist, momentan vom *„SystemTrojaner"* verteilt wird, sich in der Warteschleife zum Verteilen befindet, oder erst von einer Lehrerin oder einem Lehrer eingereicht wurde.

Je nach Status der virtuellen Maschine gibt es unterschiedliche Aktionen, die ausgeführt werden können.

Der erste Fall wäre, dass eine Gast-Maschine bereits verteilt wurde. Hierzu gibt es die Aktion, die die Maschine löscht. Dazu gibt es den HTTP-Request */admin/un-*

ploy/<Name der VM>. Sollte der Server diesen Request erhalten, wird die virtuelle Maschine vom „deployed“-Ordner in den Papierkorb verschoben.

Der nächste Zustand, in dem eine virtuelle Maschine sein kann, ist jener, dass sie von einer Lehrkraft eingereicht wurde. Hier gibt es zwei Aktionen, die durchgeführt werden können. Einerseits kann sie mit dem Request */admin/deploy/<Name der VM>* in die Warteschlange zum Verteilen gereiht werden. Hierbei wird diese einfach aus dem „submitted“-Ordner in den „pending“-Ordner verschoben. Die andere Tätigkeit, die durchgeführt werden kann, ist, die virtuelle Maschine abzulehnen. Hierbei wird diese vom „submitted“-Ordner in den Papierkorb verschoben. Des Weiteren werden die Administratorinnen, Administratoren und die Lehrperson, die die Maschine erstellt hat, über diese Aktion über eine E-Mail benachrichtigt.

Wird eine virtuelle Maschine von einer Administratorin oder einem Administrator zum Verteilen freigegeben, kommt diese in die Warteschlange. Die Gast-Maschine, die als erstes der Warteschlange hinzugefügt wird, wird daraufhin als erstes verteilt. Die einzige Aktion, die in diesem Zustand getätigt werden kann, ist die virtuelle Maschine aus der Warteschlange zu entfernen, sodass diese nicht verteilt wird. Dies wird mit dem HTTP-Request */admin/unpending/<Name der VM>* realisiert. Hierbei wird die virtuelle Maschine aus dem „pending“-Ordner in den „submitted“-Ordner verschoben.

Sollte eine virtuelle Maschine verteilt worden sein, wird die nächste aus der Warteschlange genommen. Dies wird realisiert, indem diese in den „deploying“-Ordner verschoben wird. Sobald die virtuelle Maschine sich in diesem Ordner befindet, können die Administratorinnen und Administratoren den Status nicht mehr verändern.

Die einzige Möglichkeit, die diese haben, um die Gast-Maschine aus dem „deploying“-Ordner zu entfernen, ist der Reset-Knopf. Mit diesem werden alle virtuellen Maschinen, die sich entweder im „deploying“-Ordner oder im „pending“-Ordner befinden, zurück in den „submitted“-Ordner verschoben. Der Sinn hinter dieser Funktion ist, dass man, falls es Fehler gibt oder die Kommunikation mit dem *„SystemTrojaner“* Probleme bereitet, wieder auf einen konsistenten Zustand kommen kann. Von diesem sicheren Zustand kann das Verteilen von Neuem gestartet werden. Um dies zu erledigen, muss der Request */admin/reset* an den Server gesendet werden.

4.2.5 Server – SystemTrojaner-Server

Das Verteilen der virtuellen Maschinen wurde auf die Diplomarbeit *„SystemTrojaner“* ausgelagert, da dieser Vorgang die Hauptaufgabe der Software ist. Die Kommunikation zwischen dem Vsistent-Server und dem *„SystemTrojaner“*-Server wird mit dem Protokoll MQTT verwirklicht. Des Weiteren müssen beide Serveranwendungen auf dem selben Gerät laufen, da der *„SystemTrojaner“* nur von dem Server aus verteilen kann, auf dem dieser läuft.

Konkret wird der Nachrichtenaustausch über den TCP/IP Port 6666 abgehandelt. Auf der Seite von MQTT wird der Kanal „deployAH“ verwendet. In diesem werden von beiden Serverapplikationen Nachrichten geschrieben und mitgehört. Die konkreten Nachrichten, welche ausgetauscht werden, werden in der folgenden Tabelle kurz erklärt:

Von wem	Nachricht	Erklärung
Vsistent	<Pfad von>@ <Pfad nach>@all	Sagt dem *„SystemTrojaner“*, dass er eine VM an alle Clients verteilen soll.
Vsistent	<Pfad von>@ <Pfad nach>@<mac>@...	Sagt dem *„SystemTrojaner“*, dass er eine VM an bestimmte Clients (Die, die mit ihrer MAC-Adresse genannt werden) verteilen soll.
SystemTrojaner	1@Deployment already running	Die Fehlermeldung, dass momentan schon Dateien verteilt werden. => Man muss das Deployment neu starten, da der *„SystemTrojaner“* das Deployment somit nicht verarbeiten kann.
SystemTrojaner	2@Error@<mac>@...	Sagt, dass das Deployment auf den Clients mit den genannten MAC-Adressen nicht funktioniert hat. => Man muss es auf diesen Clients neu starten.
SystemTrojaner	3@Deployment starting	Meldung, dass das gewünschte Deployment nun startet.
SystemTrojaner	4@Deployment done	Meldung, dass das gewünschte Deployment ohne Fehlermeldungen vollendet wurde.

Tabelle 4.1: Nachrichten zwischen dem Vsistent-Server und dem *„SystemTrojaner“*-Server

Wird eine virtuelle Maschine aus der Warteschlange genommen, wird mit der entsprechenden Nachricht dem *„SystemTrojaner“*-Server mitgeteilt, dass diese auf alle Clients im Netzwerk verteilt werden soll. Sollte eine Fehlermeldung auftreten, wird das Ausrollen der virtuellen Maschine erneut gestartet. Wird die Nachricht empfangen, dass die Gast-Maschine auf allen Clients verteilt wurde, bekommen die Administratorinnen, Administratoren und das Lehrpersonal, das die virtuelle Maschine eingereicht hat, eine E-Mail, dass diese nun erfolgreich verteilt wurde. Nun wird die nächste VM aus der Warteschleife genommen und der gleiche Ablauf wiederholt.

4.2.6 SystemTrojaner-Server – SystemTrojaner-Client & Benutzeroberfläche

Die Benutzeroberfläche hat keine Rolle in der Kommunikation und dem Verteilen der virtuellen Maschinen. Clientseitig wird dies von der Diplomarbeit *„SystemTrojaner“* verwaltet. Die Benutzeroberfläche übernimmt nur die Verwaltung der sich auf dem Rechner befindlichen Gast-Maschinen. Eine Beschreibung, wie das Verteilen der virtuellen Maschinen funktioniert, findet sich im Diplomarbeitsbuch der Diplomarbeit *„SystemTrojaner“*.

A Anhang

A.1 Übergangslösungen

A.1.1 Allgemeines

Um Erfahrungen mit den Kommandozeilenwerkzeugen der Hypervisoren zu sammeln, wurden einfachere Lösungen entwickelt, bevor an der finalen Applikation gearbeitet wurde. Diese dienten als Test für das Arbeiten mit den untersuchten Virtual-Machine-Monitoren in Skripten. Durch den produktiven Einsatz von zwei dieser Lösungen konnte das Feedback zu ihnen während der Entwicklung der endgültigen Anwendungen in diese eingearbeitet werden.

A.1.2 Windows

A.1.2.1 Grund dieser Übergangslösung

Diese Übergangslösung wurde erstellt, um mit dem Hypervisor *„Hyper-V“* und der Schnittstelle *„Hyper-V PowerShell Module“* Erfahrungen zu sammeln. Dies geschah im Rahmen der Evaluierung.

A.1.2.2 Programmiersprache

Als Programmiersprache wurde C# gewählt. Dies hat einerseits den Grund, dass schon Erfahrung mit dieser gesammelt werden durfte. Andererseits ist diese perfekt an die Windows-Umgebung angepasst. Des Weiteren kann man mit dieser Sprache auch ganz einfach windowseigene Bibliotheken importieren und verwenden. Dies ist sehr nützlich in Bezug auf die Schnittstelle mit dem Hypervisor *„Microsoft Hyper-V“*.

A.1.2.3 Das Programm

Allgemein Die Übergangslösung wurde so programmiert, dass dieses so wenige Interaktionen mit den einzelnen Hypervisoren wie möglich beherbergt. Die meisten Interaktionen wurden in *PowerShell*-Skripte ausgelagert. Die einzige, bei der dies nicht der Fall ist, wird im Kapitel A.1.2.3 genauer beschrieben. Dies hat einerseits den Vorteil, dass man ohne den Code groß zu ändern, mit diesem Programm größtenteils Hypervisor-unabhängig ist. Andererseits ist dadurch der Quelltext nicht zu aufgeblasen und bei Weitem verständlicher, als wenn man es für jeden Hypervisor einzeln programmieren. Noch dazu gibt es einige Funktionen, die gar nicht per C# lösbar gewesen sind und dadurch ist die Auslagerung der Skripte nicht nur sehr praktisch, sondern auch teilweise unabdingbar.

Die Klassen

Allgemein Da die Übergangslösung objektorientiert programmiert wurde, wurden verschiedene Funktionen und Methoden in unterschiedliche Klassen unterteilt. Hier ein kurzer Überblick, welche Funktionen die einzelnen Klassen erfüllen:

Klasse	Beschreibung
Program	Der Einstiegspunkt in die Übergangslösung. Liest außerdem die Konfiguration ein.
Mainmenu	Die grafische Benutzeroberfläche.
ScriptHandler	Führt in einer PowerShell-Instanz die Skripte aus und bearbeitet den Output.
Connection	Eine Klasse, die nur benötigt wird, wenn man den Hypervisor Hyper-V verwendet. Startet eine Remote-Desktop Connection zum Hypervisor und dieser reicht das Bild der VM durch.

Tabelle A.1: Die Klassen der Windows-Übergangslösung

Program Beim Programmstart wird die Konfigurationsdatei *„App.config“*, welche es in jedem C#-Programm automatisch gibt, ausgelesen. Eine Konfigurationszeile in dieser XML-Datei schaut wie folgt aus:

```
1 <add key="hypervisor" value="Hyper-V" />
```

Listing A.1: App.config

Daran sieht man, dass es sich hier um eine Map handelt. Eine Map ist ein Speichertyp, bei dem ein Key, in dem Fall „hypervisor“ auf einen Value, in diesem Beispiel „Hyper-V“, verlinkt wird. Dieser Wert ist möglich mit folgendem C#-Code auslesbar:

```
1 String hypervisor = ConfigurationManager.AppSettings["
    hypervisor"];
```

Listing A.2: Program.cs

Hier wird die C# eigene Klasse „ConfigurationManager“ verwendet, um den Wert auszulesen. Und wie beschrieben benötigt man nur den Key „hypervisor“ und diese Klasse holt sich dann automatisch aus der Konfigurationsdatei den konfigurierten Wert. Des Weiteren wird ein neuer Thread gestartet, in dem die *Mainmenu*-Klasse aufgerufen wird.

Mainmenu In dieser Klasse wird die grafische Oberfläche generiert. Im Grunde ermöglicht diese dem User mit einfachen Klicks *PowerShell*-Skripte, die bestimmte Funktionen auf gewünschte virtuelle Maschinen, auszuführen. Diese GUI wurde mit Windows-Forms umgesetzt. Hier ist ein Beispiel dieser GUI:

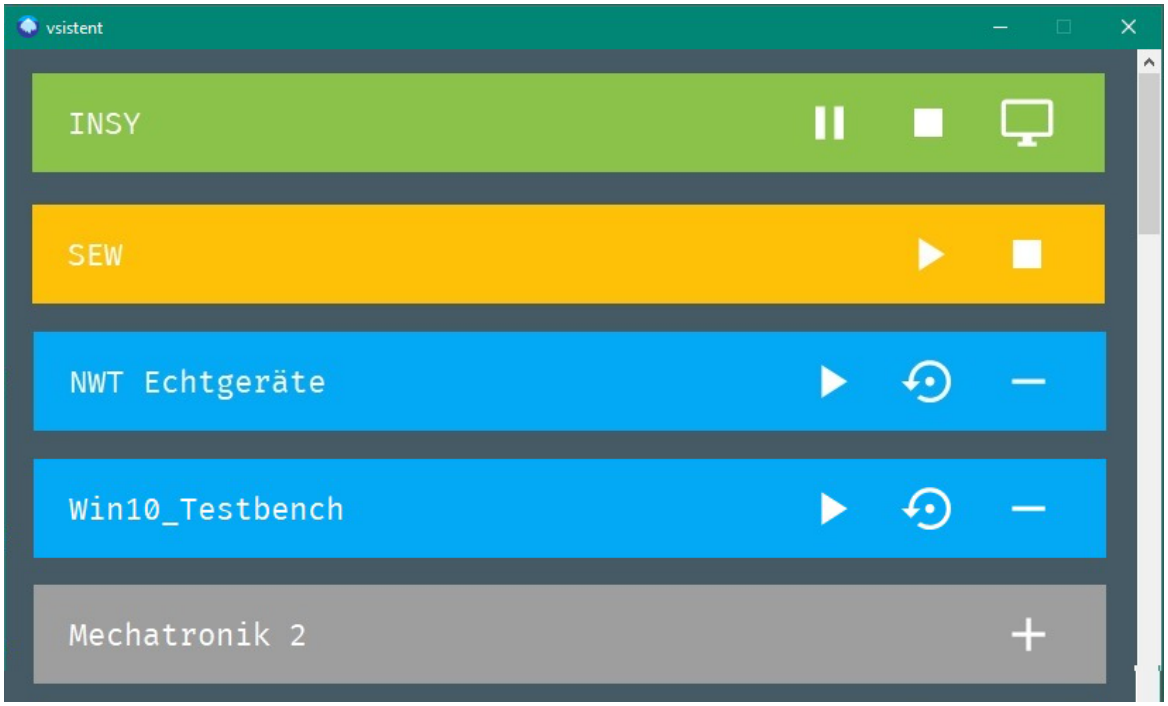


Abbildung A.1: GUI der Windows Testlösung

An diesem Bild kann man erkennen, dass verschiedene virtuelle Maschinen auf diesem PC vorhanden sind. Anzumerken ist hierbei, dass Gast-Maschinen mit grünem Hintergrund momentan laufen. Die drei Icons auf der rechten Seite sind Knöpfe, die verschiedene Funktionen ausführen. Bei laufenden virtuellen Maschinen wäre das zum einen das Einfrieren von der Maschine mit dem Pause-Symbol. Dann kann man die Gast-Maschinen auch noch beenden mit dem Stop-Symbol und eine Remote-Desktop-Verbindung starten mit dem Bildschirm-Symbol. Virtuelle Maschinen mit blauem Hintergrund sind bereites geklont und können jederzeit mit dem Start-Symbol ausgeführt werden. Mit dem Replay-Symbol kann man erzwingen, dass dieser Klon gelöscht und ein neuer erstellt wird. Schlussendlich kann man mit dem Minus-Symbol den Klon komplett löschen.

Des Weiteren gibt es noch zwei weitere Möglichkeiten, welche Hintergrundfarbe eine virtuelle Maschine haben kann. Ist die Hintergrundfarbe orange, heißt das, dass die Gast-Maschine momentan eingefroren ist. Die Funktionen auf eine virtuelle Maschine in diesem Zustand sind einerseits das Auftauen der Maschine mit dem Play-Symbol und andererseits das Herunterfahren dieser mit dem Stop-Symbol. Die letzte Möglichkeit, die es gibt, ist, dass der Hintergrund grau ist. Dies bedeutet, dass es von der Maschine nur einen Master gibt und noch keinen Klon. Die einzige Funktion, die durchgeführt werden kann, wenn es nur den Master gibt, ist mit dem Plus-Symbol einen neuen Klon zu erstellen.

ScriptHandler In der Klasse *ScriptHandler* gibt es drei verschiedene Arten an Methoden. Die Arten differenzieren sich anhand der Funktionen, die die Methoden beinhalten. Zuerst gibt es die Methoden, die richtigen Skripte zu den Userinputs ausführen. Das wären die Folgenden:

Methode	Beschreibung
AddVm()	Klont die VM (Plus-Symbol)
StartVm()	Startet die VM (Play-Symbol)
ResetVm()	Setzt die VM auf Werkseinstellungen zurück (Replay-Symbol)
RemoveVm()	Löscht den Klon (Minus-Symbol)
StopVm()	Beendet die VM (Stop-Symbol)
SuspendVm()	Friert die VM ein (Pause-Symbol)
ResumeVm()	„Taut“ die VM wieder auf (Play-Symbol)
ConnectVm()	Zeigt das Bild der VM an (Bildschirm-Symbol)

Tabelle A.2: Methoden, die Userinputs ausführen

Dann gibt es noch die Methoden, die regelmäßig aufgerufen werden, um den Status der virtuellen Maschinen zu erfahren. Dies wird realisiert, indem Skripte aufgerufen werden und den Output dieser analysiert und verarbeitet wird. Methoden dieser Art sind die Folgenden:

Methode	Beschreibung
GetRunningVms()	Gibt alle VMs zurück die gerade laufen.
GetClones()	Gibt alle VMs zurück die geklont wurden.
GetMasters()	Gibt alle Master-VMs zurück.
GetSuspendedVms()	Gibt alle VMs zurück die momentan eingefroren sind.
CheckVMs()	Ruft alle Get-Methoden auf.

Tabelle A.3: Methoden, die den Status von VMs ausgeben

Und die letzte Methode ist jene, die eine *PowerShell*-Instanz startet und in dieser dann das Skript ausführt, welches ihr mitgegeben wurde:

- RunPSscript(string script)

Damit man dieses *PowerShell*-Skript, über ein C#-Programm ausführen kann, benötigt man zuerst einen *„Process“* dem man mitgibt, dass man eine *PowerShell* öffnen will und andere Informationen, wie zum Beispiely dass man das *PowerShell*-Fenster nicht sieht:

```
1  Process p = new Process();
2  p.StartInfo.FileName = "Powershell.exe";
3
4  p.StartInfo.CreateNoWindow = true;
5  p.StartInfo.RedirectStandardOutput = true;
6  p.StartInfo.UseShellExecute = false;
7  p.StartInfo.Verb = "runas";
8  p.StartInfo.StandardOutputEncoding = Encoding.UTF8;
9  p.StartInfo.Arguments = "-Exec Bypass -File " + Program.
       scripts + script;
10 p.Start();
```

Listing A.3: ScriptHandler.cs

In Zeile neun werden der *PowerShell* nun die Parameter mitgeben. Wenn man allerdings nur das Skript mitgeben würde, würde es zu Fehlermeldungen wegen der Execution-Policy geben. Denn standardgemäß werden Skripte geblockt, die nicht direkt vom User in der *PowerShell* ausgeführt werden. Dies kann umgangen werden, indem man den Parameter *„-Exec Bypass“* mitgibt.

Connection In dieser Klasse wird mithilfe der Bibliothek *„mstscax.dll“*, die automatisch auf jedem Windows installiert ist, eine RDP-Session zu den einzelnen virtuellen Maschinen aufgebaut. Diese DLL muss man dann nur als Verweis importieren. Mit

dieser kann mit dem C#-Programm eine Remote-Desktop-Verbindung aufgebaut werden. Dazu muss die Komponente *„Microsoft RDP Client Control – version 9“* in seiner Microsoft-Forms-Applikation hinzugefügt werden. Diesen muss man dann im Quelltext die notwendigen Einstellungen mitgeben. Mithilfe dieser weiß das Programm mit welchem PC, in diesem Fall dem *localhost*, eine Verbindung aufgebaut werden soll. Eine weitere Option, die konfiguriert werden muss ist, welche Gast-Maschine durchgereicht werden soll. Hier der notwendige Code:

```
1 vm.Server = "localhost";
2 [...]
3 vm.AdvancedSettings2.RDPPort = 2179;
4 vm.AdvancedSettings7.PCB = this.vmid;
5 [...]
6 vm.Connect();
```

Listing A.4: Connection.cs

In Zeile eins wird hier dem *„Microsoft RDP Client Control“* mitgegeben, auf welchen Server er sich verbinden soll. In diesem Fall in dies *localhost*, da sich der *Hyper-V* lokal befindet. In der dritten Zeile wird dann der Port definiert, welcher die RDP-Verbindung zulässt. In diesem Fall muss der Port 2179, über den Hyper-V das Bild der Maschinen weitergibt, verwendet werden. Von welcher Maschine das Bild übertragen wird, wird in Zeile vier über die VMID konfiguriert. Diese Kennung entspricht genau einer einzigen virtuellen Maschine. In Zeile sechs wird dann, sobald die Konfiguration abgeschlossen ist, eine Anfrage an den *Hyper-V* Server geschickt. Sollte dieser die Anfrage annehmen und ein Bild zurücksenden, sieht man in der *Windows-Forms*-Applikation das Bild der gewünschten virtuellen Maschine.

Konfiguration Da innerhalb des Quelltextes keine Konfiguration durchgeführt werden soll, wird die in C# implementierte Konfigurationsdatei verwendet, um die notwendigen Konfigurationen vorzunehmen:

```
1 <!-- Allgemeine Konfig -->
2 <add key="version" value="1.0" />
3 <add key="hypervisor" value="Hyper-V" />
4 <add key="scripts" value=".\\scripts\\" />
5 <add key="icons" value=".\\icons\\" />
6 <!-- Hyper-V Konfig -->
7 <add key="masters" value="D:\\vsistent\\masters\\" />
8 <add key="clones" value="D:\\vsistent\\clones\\" />
```

Listing A.5: App.config

In diesem Ausschnitt der Konfigurationsdatei sieht man, wie eine solche aussehen kann. Da die Datei im XML-Format geschrieben ist, ist diese auch ganz leicht für einen Menschen lesbar. Im „key“-Feld wird hier der Name der Einstellung definiert und im „value“-Feld der Wert, den diese Einstellung annehmen soll.

A.1.2.4 Skripte

Skripte Allgemein Die Skripte, die benötigt werden sind:

Skript	Erklärung
clone.ps1	Klont eine VM.
remove.ps1	Entfernt eine VM.
reset.ps1	Löscht eine VM und erstellt danach einen neuen Klon von dieser.
start.ps1	Startet eine VM.
stop.ps1	Stoppt eine VM
suspend.ps1	Friert eine VM ein.
resume.ps1	Taut eine VM auf.
running.ps1	Gibt alle VMs zurück die momentan laufen.
suspended.ps1	Gibt alle VMs zurück die momentan eingefroren sind.
getClones.ps1	Gibt alle VMs zurück, von denen es einen Klon gibt.
getMasters.ps1	Gibt alle VMs zurück, von denen es einen Master gibt.

Tabelle A.4: Benötigte Skripte und die Funktionen dieser

Sind diese Skripte für einen bestimmten Hypervisor vorhanden, kann man diese in den Skripte-Ordner des Programmes kopieren und dann mit der Übergangslösung diesen Hypervisor bedienen.

Skripte Hyper-V Für den Hypervisor Hyper-V werden all diese Skripte auch benötigt wie sie oben beschrieben sind. Dieser Hypervisor hat die Eigenschaft, dass man für die meisten Funktionen die VMID benötigt. Die VMID ist die eindeutige Kennung einer virtuellen Maschine. Da man diese braucht, benötigt man für Hyper-V ein zusätzliches Skript:

Skript	Erklärung
getVmid.ps1	Gibt die VMID zu einem VM-Namen zurück.

Tabelle A.5: Skripte, die speziell benötigt werden, wenn Hyper-V im Einsatz ist

Diesem Skript wird nur der Name einer VM mitgegeben und der Output ist in Folge die VMID, sofern der Name der virtuellen Maschine eindeutig ist.

A.1.3 Linuxskripte

A.1.3.1 Verwendung

Diese Lösung wurde in den Sälen 072 und 079 der HTL Rennweg eingesetzt, um die Schülerinnen und Schüler mit der Diplomarbeit und den Verbesserungen durch das Projekt vertraut zu machen.

A.1.3.2 Verwendete Technologien

Shell-Skripte Der Kern dieser Lösung ist das Shell-Skript *startvm.sh*. Dieses Skript führt alle Interaktionen mit der virtuellen Maschine aus und ist somit für den gesamten Klon- und Startprozess verantwortlich. Die einzige ausgelagerte Funktion ist das Einhängen eines Shared-Folders bei Linux-VMs, da dies mit einem Skript am Gastsystem geschieht. Dafür wird das Shell-Skript *mnt.sh* verwendet.

VMware Workstation Pro Diese Lösung wurde für den Einsatz in der Schule entwickelt, weshalb sie mit dem Hypervisor VMware Workstation Pro und dem Kommandozeilentool *vmrun* arbeitet. Da das *startvm.sh*-Skript nicht modular aufgebaut ist, würde ein Wechsel des Virtual-Machine-Monitors zu einer Änderung des gesamten Skriptes führen.

.desktop-Dateien Damit das *startvm.sh*-Skript den Benutzern auf der Benutzeroberfläche angezeigt wird, wurden .desktop-Dateien erstellt, welche das Skript ausführen. Außerdem stellen diese Dateien sicher, dass das Skript mit den richtigen Parametern aufgerufen wird. Deshalb gibt es für jede virtuelle Maschine eigene .desktop-Dateien.

Python-Skript und .ini-Dateien Um das Erstellen der zahlreichen .desktop-Dateien zu automatisieren, wurde dafür ein Python-Skript geschrieben. Dieses Skript bezieht die notwendigen Informationen zum Generieren dieser Verknüpfungen von einer Initialisierungsdatei.

A.1.3.3 Konkrete Umsetzung

startvm.sh Damit die gewünschte virtuelle Maschine geklont und gestartet wird, nimmt dieses Skript folgende Argumente:

Parameter	Beschreibung
VM	Der Name der zu startenden VM
vmpath	Der Pfad zu der Master-VM
clonepath	Der Pfad zu dem Klon der VM
newclone	Soll ein neuer Klon erstellt werden (boolean)
sharedfolder	Soll ein Shared-Folder eingebunden werden (boolean)
os	Das Gast-Betriebssystem
username	Der Gast-Benutzername
password	Das Gast-Passwort
mntscript_host	Der Pfad zum Mountskript für den Shared-Folder am Host
mntscript_guest	Der Pfad zum Mountskript für den Shared-Folder am Gast
guestshare	Das Einbindeverzeichnis des Shared-Folders

Tabelle A.6: Parameter des startvm.sh-Skriptes

Bei den Parametern ist zu beachten, dass das Betriebssystem, die Anmeldedaten, die Speicherorte des Mount-Skriptes und das Einbindeverzeichnis nicht angegeben werden müssen, wenn man keinen Shared-Folder auf einem Linux-Gast erstellen will. Wenn es sich um ein Windows-Gastsystem handelt und man einen Shared-Folder verwenden will, muss man neben dem Parameter *sharedfolder* nur das Betriebssystem angeben.

Shared-Folder Um das Teilen von Dateien zwischen dem Hostsystem und den virtuellen Maschinen zu vereinfachen, wird beim Starten einer virtuellen Maschine über das Skript automatisch ein Shared-Folder erzeugt. Dieser Ordner wird nach dem Starten der virtuellen Maschine durch das *mnt.sh*-Skript eingehängt.

```
1 #!/bin/bash
2
3 path=$1
4
5 vmhgfs-fuse -o nonempty .host:/Share $path
```

Listing A.6: mnt.sh

Der Befehl *vmhgfs-fuse* wird verwendet, um den geteilten Ordner einzuhängen.[11] Der Befehl steht Gastsystemen zur Verfügung, in welchen die VMware-Tools installiert sind.

Damit das Skript *mnt.sh* auch ausgeführt werden kann, muss es zuerst auf das Gastsystem kopiert werden und dann auf diesem gestartet werden. Bei dem Starten eines Skriptes in der Gastmaschine ist zu beachten, dass man die Anmeldedaten eines Be-

nutzers benötigt und dieser auch angemeldet sein muss, wenn das Skript ausgeführt werden soll.

Unter Windows wird der Shared-Folder als Netzwerklaufwerk angezeigt, wenn der Parameter bei dem Startskript auf *true* gesetzt wurde. Auf dem Hostsystem befinden sich die geteilten Verzeichnisse unter dem Pfad *~/vsistent/sharedFolder*.

.desktop-Dateien Es werden für jede virtuelle Maschine zwei .desktop-Dateien erstellt. Eine startet einen vorhandenen Klon und erstellt nur einen Neuen, wenn es noch keinen gibt. Die andere Datei, welche man durch den Zusatz „(neue Maschine)“ beim Namen erkennen kann, erstellt immer einen neuen Klon und löscht den aktuellen Klon, wenn es einen gibt. Diese Dateien unterscheiden sich aber nur bei dem Argument *newclone* des *startvm.sh*-Skriptes. Den Aufbau einer solchen Verknüpfung kann man gut anhand eines Beispiels sehen. In diesem Fall ist es ein eine .desktop-Datei für eine Windows 10-VM.

```
1 [Desktop Entry]
2 Name=Windows_10_x64_1809_Basic
3 Comment=Startet die VM Windows_10_x64_1809_Basic
4 Exec=/usr/local/share/bin/startvm.sh [...]
5 Terminal=true
6 Type=Application
7 Icon=/usr/share/icons/vsistent.png
8 Categories=Office
```

Listing A.7: Desktopentry

Es wurden dabei die Parameter des *startvm.sh*-Skriptes weggelassen. Es wurde das Element *Terminal* auf den Wert *true* gesetzt, damit der Benutzer Ausgaben über die durchgeführten Aktionen bekommt, während die virtuelle Maschine gestartet wird. Dies wurde so eingestellt, damit die Wartezeit für den Benutzer überbrückt wird und er auch sehen kann, dass tatsächlich etwas passiert.

Dem Benutzer werden diese Dateien mit dem angegebenen Namen in der grafischen Oberfläche angezeigt und nicht mit dem Dateinamen. Damit diese Dateien im Startmenü angezeigt werden, werden sie im Verzeichnis */usr/local/share/applications* gespeichert. Sie werden in der Kategorie „Büro“ gruppiert und besitzen vorerst alle das Vsistent-Logo als Icon.

Erstellen der Verknüpfungen Damit die Verknüpfungen automatisch erstellt werden können, werden einige Parameter benötigt, welche in einer .ini-Datei definiert werden.

Die Initialisierungsdatei für die Anwendung in der Schule sieht folgendermaßen aus:

```
[default]
# The delimiter between multiple paths
delimiter = :
# The directories in which the VMs can be found
vmlocations = /masters/VM/Linux:/masters/VM/Windows
# The directory in which the script can be found
scriptlocation = /usr/local/share/bin
# The directory in which the .desktop-files will be
   created
filefolder = /usr/local/share/applications
# The path to the icon of the .desktop-files
icon = /usr/share/icons/vsistent.png
# The hypervisor
hypervisor = vmware

[vmware]
clonelocation = vsistent/clones
```

Listing A.8: config_mult.ini

Der Hypervisor wird angegeben, damit man für jeden Hypervisor eigene Parameter hinzufügen kann, wenn spezifische Konfigurationen nur für einen Virtual-Machine-Monitor benötigt werden.

Um Änderungen bei der Verknüpfung von einer bestimmten virtuellen Maschine vorzunehmen, kann man eine Initialisierungsdatei für diese Maschine erstellen. Diese Datei wird dann in den Ordner der virtuellen Maschine gespeichert. Eine .ini-Datei für eine VM sieht wie folgt aus.

```
[info]
# The name of the VM
name = Linux Mint 19
# A short description of the VM
comment = Startet eine Linux Mint 19 VM
# Specifies, if a shared folder is needed
sharedfolder = true
# The OS oft the VM (Linux or Windows)
os = linux
# The username of the user, for whom the shared folder
   will be created
```

```
username = junioradmin
# The password of the user, for whom the shared folder
   will be created
password = junioradmin
# The location of the mountscript on the hostmachine
mntscript_host = /usr/local/share/bin/mnt.sh
# The location of the mountscript on the guestmachine
mntscript_guest = /home/junioradmin/mnt.sh
# The location of the shared folder in the VM
guestshare = ~/Dokumente
```

Listing A.9: vmconfig.ini

Diese Konfigurationen können getroffen werden, wenn man den .desktop-Dateien einen kürzeren oder aussagekräftigeren Namen geben will. Außerdem wird diese Datei benötigt, wenn man einen Shared-Folder bei einer virtuellen Maschine verwenden will.

Um mit den definierten Parametern in Python zu arbeiten, werden diese Konfigurationsdateien mit dem Modul „configparser“ eingelesen. Das Einlesen eines Wertes aus der Initialisierungsdatei geschieht folgendermaßen:

```
import configparser
[...]
configfile = configparser.ConfigParser()
configfile.read(args.config)
delimiter = configfile['default']['delimiter']
```

Listing A.10: createDesktopFiles.py: Einlesen von Werten

Mit den eingelesenen Werten kann anschließend eine .desktop-Datei generiert werden. Diese Aufgabe wird von zwei Methoden durchgeführt, *handlevmconfig* und *writedesktopfile*.

Die Methode *handlevmconfig* wird für das Auslesen der Konfigurationsdateien von einzelnen virtuellen Maschinen benötigt. Sollte es keine Initialisierungsdatei für eine spezifische virtuelle Maschine geben, werden die benötigten Variablen mit Standardwerten befüllt. Diese Methode gibt dann eine Liste mit den Parametern zurück. Die Methode *writedesktopfile* verwendet diese Werte anschließend, um die .desktop-Datei zu erstellen.

A.1.3.4 Untersuchte Alternativen

Desktopumgebungen Zu Beginn wurde angestrebt, dass die Skripte als Desktopumgebung integriert werden, also dass der Benutzer beim Anmelden die gewünschte virtuelle Maschine auswählen kann und diese automatisch gestartet wird. Diese Variante wurde auch im Laufe der Diplomarbeit ausgiebig getestet. Damit ein Skript als Umgebung ausgewählt werden kann, muss die .desktop-Datei, welche das Skript aufruft, im Ordner */usr/share/xsessions* gespeichert werden.

Da bei dieser Umsetzung jedoch kein Weg gefunden wurde, um ein Skript mit Argumenten in der .desktop-Datei aufzurufen, müssten für jede virtuelle Maschine zwei Skripte erstellt werden, welche die Maschine einfach starten oder einen neuen Klon erstellen. Aufgrund der Schwierigkeiten beim Erstellen und Verwalten von vielen Skripten wurde dieser Ansatz nicht produktiv eingesetzt.

Auslagerung von Funktionen Weil das *startvm.sh*-Skript eine virtuelle Maschine klont, diesen Klon konfiguriert, ihn startet und einen Shared-Folder einhängt, wurde überlegt, ob diese Funktionen in andere Skripte aufgeteilt werden sollen. Diese Methode ist vorteilhaft, da die einzelnen Skripte dadurch kurzgehalten werden und sie dadurch sehr übersichtlich sind. Der Ansatz wurde aber nicht verfolgt, da diese Testlösung sehr früh entstanden ist und deshalb vorhersehbar war, dass sie oft umstrukturiert werden würde. Dies wurde vereinfacht, da die Argumente nur für ein Skript angepasst werden mussten. Außerdem wusste man bei Tests dadurch, in welchem Skript sich Fehler befinden und musste nicht Zeit aufwenden um zu überprüfen, ob das Erstellen der Argumente oder das Verwenden dieser zu den Fehlersituationen führten. Abschließend ist auch zu sagen, dass das einfache Verteilen eines Skriptes auch ein wichtiger Vorteil ist.

A.1.4 Windowsskripte

A.1.5 Grund der Übergangslösung

Diese Übergangslösung wurde in EDV-Sälen mit Windows-Hostrechner etabliert, um den Schülerinnen und Schülern eine sofortige Erleichterung im Umgang mit virtuellen Maschinen zu bieten.

A.1.6 Verwendete Technologien

A.1.6.1 Bash Skripte

Im Wesentlichen werden bei dieser Lösung zwei Batch Skripte je virtueller Maschine generiert. Eines dieser Skripte dient dazu einen bereits vorhandenen Klon der virtuellen Maschine, für welche dieses Skript geschrieben ist, zu starten. Sollte noch kein Klon vorhanden sein wird mit dem Skript zusätzlich ein neuer Klon erstellt. Das zweite Skripte kontrolliert ebenfalls, ob bereits ein Klon der virtuellen Maschine vorhanden ist. Im Unterschied zum ersten Skript wird hier jedoch der bestehende Klon entfernt, ein neuer Klon erstellt und dieser gestartet.

A.1.6.2 VMware Workstation Pro

Als Hypervisor wurde hier auf VMware Workstation Pro gesetzt, da dieser bereits in den EDV-Sälen in Verwendung war und ein großes Augenmerk daraufgelegt wurde, dass das bereits bestehende System uneingeschränkt und unverändert verfügbar bleibt. Aus diesem Grund wurde in diesem vorläufigen Lösungsansatz das Kommandozeilentool *vmrun*, von *VMware*, eingesetzt.

A.1.7 Konkrete Umsetzung

A.1.7.1 Batchskripte

Folgende Variablen werden in beiden der Skripte definiert:

Variable	Beschreibung
pathToMasterVMX	Pfad zu Konfigurationsdatei der Master-VM
cloneName	Name des Klons
clones	Pfad zu dem Klon-Verzeichnis des Benutzers
clone	Pfad zu Konfigurationsdatei der Klon-VM
pathToClone	Pfad zum Ordner der Konfigurationsdatei der Klon-VM
vmrun	Pfad zur *vmrun.exe*
share	Verzeichnis in dem der Shared Folder eingebunden wird

Tabelle A.7: Definierte Variablen

A.1.7.2 Startskript

Das Startskript ist ein Batchskript, welches nach der jeweiligen Master-VM benannt wird. Exemplarisch das Startskript für die virtuelle Maschine *SEW20181116*:

```
1  @echo off
2  echo starting...
3  SET pathToMasterVMX=C:\VM\Softwareentwicklung V2\
     SEW20181116.vmx
4  SET cloneName=SEW20181116-Clone
5  SET clones=%userprofile%\Clones
6
7  SET clone=%userprofile%\Clones\SEW20181116-Clone\
     SEW20181116-Clone.vmx
8  SET pathToClone=%userprofile%\Clones\SEW20181116-Clone
9  SET vmrun=c:\"Program Files (x86)"\VMware\"VMware
     Workstation"\vmrun.exe
10 SET share=%userprofile%\Share
```

Listing A.11: SEW20181116.bat: 1-10

Zuerst werden in diesem Skript die Variablen festgelegt. Für *clones*, *vmrun* und *share* werden, unabhängig der Master-VM für diese dieses Skript geschrieben ist, die selben Werte als default Parameter gesetzt. Die Werte der restlichen Variablen unterscheiden sich je nach Master-VM. In diesem konkret genannten Beispiel *SEW20181116*.

```
1  FOR /F "tokens=*" %%F IN ('%vmrun% -T ws listSnapshots "%
     pathToMasterVMX%"') DO (
2  SET snapshot=%%F
3  )
4  echo Snapshot: "%snapshot%" wird zum clonen verwendet
```

Listing A.12: SEW20181116.bat: 12-15

Hier wird mit dem *listSnapshots* Befehl von *vmrun* der neuste Snapshot der Master-VM gesetzt, da dieser als Grundlage zum Klonen der virtuellen Maschine verwendet wird.

```
if exist %pathToClone% (
echo "VM wird gestartet..."
%vmrun% -T ws start "%clone%" gui
)
```

Listing A.13: SEW20181116.bat: 17-20

Hier wird überprüft, ob bereits ein Klon der Master-VM vorhanden ist, falls dies der Fall ist, wird dieser mit dem *vmrun* Befehl *start* gestartet.

```
if not exist %share% mkdir %share%

%vmrun% -T ws clone "%pathToMasterVMX%" "%clone%" linked -
   cloneName="%cloneName%" "%snapshot%"
echo "VM wurde geklont"
echo gui.fullScreenAtPowerOn = "TRUE" >> %clone%
echo gui.viewModeAtPowerOn = "fullscreen" >> %clone%
echo msg.autoAnswer = "TRUE" >> %clone%
echo "VM wird gestartet..."

%vmrun% -T ws start "%clone%" gui

%vmrun% -T ws enableSharedFolders "%clone%"

%vmrun% -T ws addSharedFolder "%clone%" Share %share%
```

Listing A.14: SEW20181116.bat: 21-37

Falls noch kein Klon für die virtuelle Maschine dieses Skriptes vorhanden ist, wird hier überprüft ob bereits ein Verzeichnis für einen Shared Folder vorhanden ist. Falls noch kein Verzeichnis erstellt wurde, wird es hier generiert. Dadurch, dass alle virtuelle Maschinen, welche mit dieser Lösung von Vsistent gestartet, denselben Ordner für den Share nutzen ist es möglich, dass dieser bereits vorhanden ist.

Danach wird mit dem *clone* Befehl von *vmrun* ein neuer Klon erstellt. Zur komfortablen Nutzung werden folgende Einträge in Konfigurationsdatei des Klons geschrieben:

```
gui.fullScreenAtPowerOn = "TRUE"
gui.viewModeAtPowerOn = "fullscreen"
msg.autoAnswer = "TRUE"
```

Listing A.15: SEW20181116.bat: 25-27

Diese Einträge ermöglichen ein automatisches Starten im Fullscreen Modus der virtuellen Maschine und die standardmäßigen Abfragen von *VMware Workstation Pro* automatisch zu beantworten. Das Kommandozeilentool von *VMware* bietet, im Gegensatz zu anderen Hypervisoren, keine eigenen Parameter beim Starten einer virtuellen Maschine, um diese Funktionen zu ermöglichen.

Die letzten drei *vmrun*-Befehle dienen dazu, den eben erstellten Klon zu starten und einen *Shared Folder* einzubinden.

A.1.7.3 Startskript_newVM

Das Startskript_newVM wird wie das *gewöhnliche* Startskript nach der jeweiligen Master-VM benannt. Zusätzlich wird hier die Erweiterung _newVM dem Namen der Batch Datei hinzugefügt, um die zwei Skripte voneinander unterscheiden zu können.

Die Variablen des *Startskript_newVM* sind identisch zu denen des gewöhnlichen *Startskripts*. Der wesentliche Unterschied zum einfachen *Startskript*:

```
if exist %pathToClone% rmdir /S /Q %pathToClone%
```

Listing A.16: SEW20181116_newVM.bat: 17

Hier wird ebenfalls überprüft, ob bereits ein Klon der Master-VM vorhanden ist, jedoch wird hier, falls dies der Fall ist, dieser gelöscht und ein neuer Klon erstellt und gestartet.

A.2 Benutzerhandbücher

A.2.1 Linuxskripte

Andreas Himmler
Julian Kern
Raffael Zbiral

Visistent Benutzerhandbuch: Linuxskripte

Starten einer VM

Zum Starten von virtuellen Maschinen wurden Dateien erstellt, welche dieselben Namen wie die VMs besitzen. Diese Dateien findet man im Startmenü unter der Kategorie „Büro“ oder über die Suche im Startmenü.

Es gibt zwei Dateien zu jeder virtuellen Maschine:

- Name der VM ohne Zusatz:
  - Falls ein Klon im Klonverzeichnis (*~/vsistent/clones*) vorhanden ist, wird dieser gestartet. Sonst wird im Klonverzeichnis ein neuer Klon erstellt und dieser wird gestartet.
- Name der VM mit dem Zusatz „(neue Maschine)“:
  - Falls ein Klon im Klonverzeichnis vorhanden ist, wird dieser gelöscht. Anschließend wird im Klonverzeichnis ein neuer Klon erstellt und dieser wird gestartet. Dies geschieht auch, wenn kein Klon vorhanden war.

Shared-Folder

Beim Starten der VMs wird auch automatisch ein Shared-Folder eingebunden. Dieser befindet sich am Hostsystem unter dem Verzeichnis *~/vsistent/sharedfolder*. Auf dem Gastbetriebssystem wird dieser als Netzwerklaufwerk angezeigt. Mit diesem Verzeichnis kann man Dateien zwischen dem physischen und dem virtuellen Computer teilen. Dadurch sind Dateien der VM in diesem auch gesichert, wenn die VM abstürzt.

Bei Problemen können die virtuellen Maschinen auch wie gewohnt mit VMware Workstation gestartet werden. Wir würden uns freuen, wenn Sie Feedback an office@vsistent.at senden.

1

Andreas Himmler
Julian Kern
Raffael Zbiral

Benötigte Dateien

Desktopdateien: /usr/share/applications/*Desktopfilename*.desktop

Startskript: /usr/local/share/bin/startvm.sh

Icon: /usr/share/icons/vsistent.png

Pfad der Klone: ~/vsistent/clones

Pfad des Shared-Folders: ~/vsistent/sharedfolder

A.2.2 Windowsskripte

Andreas Himmler
Julian Kern
Raffael Zbiral

Vsistent Benutzerhandbuch: Windowsskripte

Starten einer VM

Zum Starten von virtuellen Maschinen wurden Batchskripte mit dem Namen der jeweiligen VM erstellt. Ausfuhr dieser dient dazu den Vorgang des Klonens und des Startens zu automatisieren.

Es gibt zwei Dateien zu jeder virtuellen Maschine:

- Name der VM ohne Zusatz:
  - Falls ein Klon im Klonverzeichnis (*C:|Users|username|share*) vorhanden ist, wird dieser gestartet. Sonst wird im Klonverzeichnis ein neuer Klon erstellt und dieser wird gestartet.
- Name der VM mit dem Zusatz „_newVM“
  - Falls ein Klon im Klonverzeichnis vorhanden ist, wird dieser gelöscht. Anschließend wird im Klonverzeichnis ein neuer Klon erstellt und dieser wird gestartet. Dies geschieht auch, wenn kein Klon vorhanden war.

Shared-Folder

Beim Starten der VMs wird auch automatisch ein Shared-Folder eingebunden. Dieser befindet sich am Hostsystem unter dem Verzeichnis *C:|Users|username|share*. Auf einer Windows-VM wird dieser Ordner als Netzwerklaufwerk angezeigt und auf einer Linux-VM wird der Ordner unter */mnt/hgfs/Share* erstellt. Mit diesem Verzeichnis kann man Dateien zwischen dem physischen und dem virtuellen Computer teilen. Dadurch sind Dateien der VM in diesem auch gesichert, wenn die VM abstürzt. Sollte der Shared Folder nicht in ihrer Guest VM erscheinen muss dieser händisch über **VM > Settings > Options > Shared Folders** aktiviert werden.

Bei Problemen können die virtuellen Maschinen auch wie gewohnt mit VMware Workstation gestartet werden.

Wir würden uns freuen, wenn Sie Feedback an office@vsistent.at senden.

A.2.3 Applikation

A.2.3.1 User-Oberfläche

Vsistent

Benutzerhandbuch

Verwende Vsistent für angenehmes Arbeiten mit virtuellen Maschinen

Andreas Himmler
Julian Kern
Raffael Zbiral

1

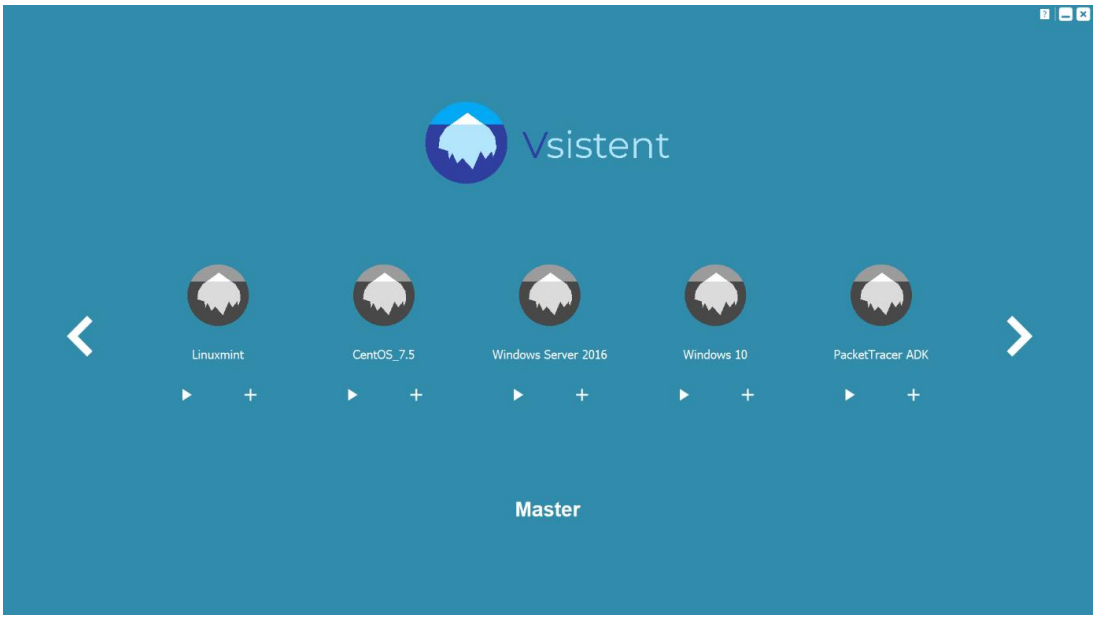
Vsistent Benutzerhandbuch

Was ist Vsistent?

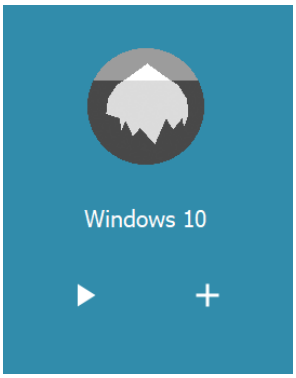
Vsistent ist eine Software, die Ihnen einen einfachen und unkomplizierten Umgang mit virtuellen Maschinen ermöglicht. Mit Vsistent sehen Sie auf den ersten Blick, welche Master-VMs auf Ihrem Rechner vorhanden sind und können diese bequem mit einem Klick starten. Nicht nur das Klonen und Starten wird in dieser Benutzeroberfläche ermöglicht, es lassen sich auch einfach Snapshots ihrer geklonten virtuellen Maschinen erstellen. Falls ihre virtuelle Maschine einmal defekt werden sollte, können Sie mit Vsistent sogar ganz einfach den letzten Snapshot ihrer virtuellen Maschine wiederherstellen.

Master-Bildschirm

Beim ersten Starten von Vsistent öffnet sich dir direkt eine Oberfläche auf der alle, auf dem Rechner, vorhandenen Master-VMs angezeigt werden.



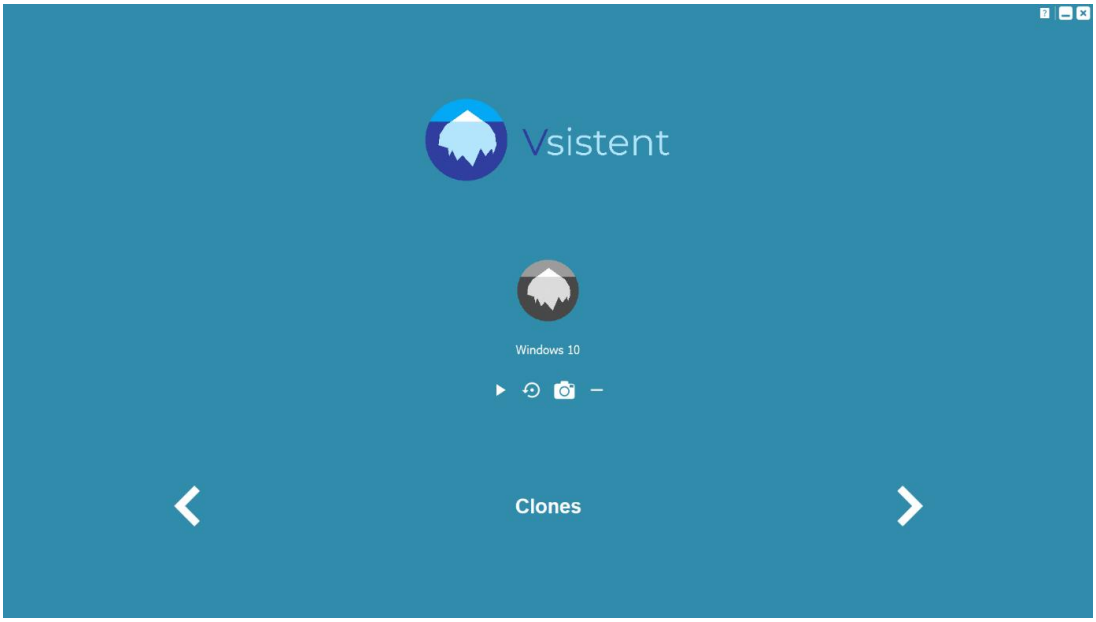
Sollten mehr als fünf virtuelle Maschinen zur Verfügung stehen können Sie mithilfe der Pfeiltasten alle bereitgestellten Master-VMs anschauen.



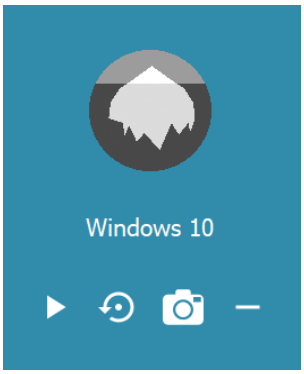
Für jede Master-VM haben Sie jetzt zwei Optionen. Mit dem Icon klont sich Ihre ausgewählte virtuelle Maschine und starten automatisch. Mit dem Icon erstellen Sie einen Klon für die gewählte virtuelle Maschine und Sie kommen automatisch auf die Ansicht aller vorhanden geklonten virtuellen Maschinen. Sollte eine virtuelle Maschine als Klon gelistet sein, wird diese nicht mehr bei den Master-VMs angezeigt.

Klon-Bildschirm

Sollten bereits mit Vsistent geklonte virtuelle Maschinen beim Starten von Vsistent vorhanden gewesen sein, öffnet sich direkt der Klon-Bildschirm.



Um zurück auf den Master-Bildschirm zu kommen verwenden Sie die Pfeiltasten am unteren Bildschirmrand.



Für jeden vorhanden Klon gibt es die folgenden Optionen:

 Startet den ausgewählten Klon

 Erstellt einen Snapshot vom momentanen Status der virtuellen Maschine

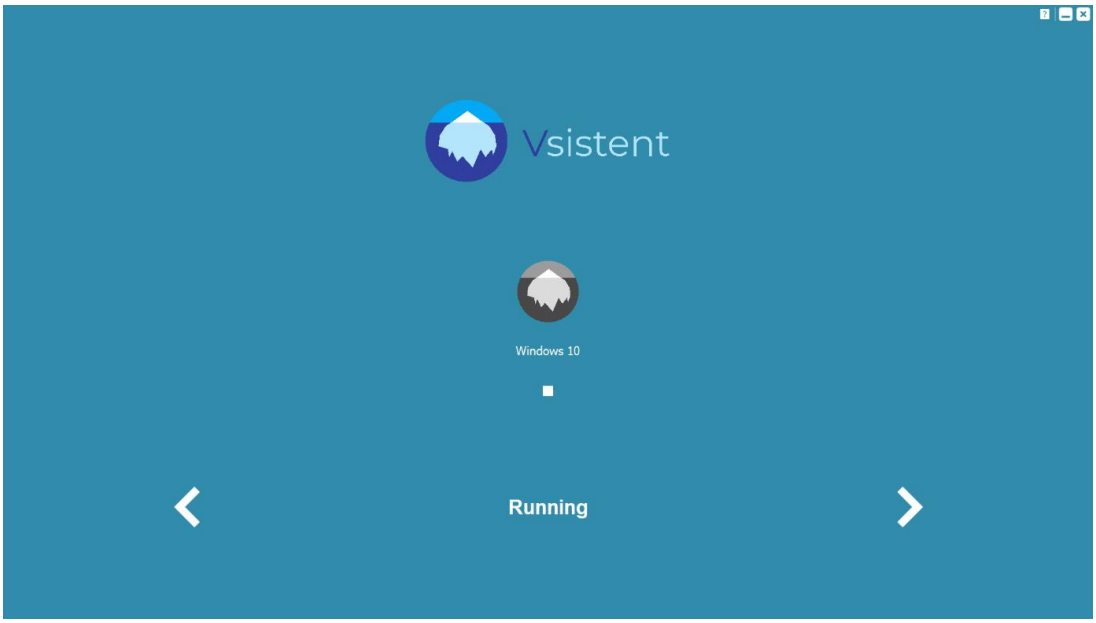
 Setzt den Status der virtuellen Maschine auf den des letzten Snapshots zurück

 Entfernt den Klon der virtuellen Maschine

Sollte der letzte Klon aus der Liste entfernt worden sein kommen Sie automatisch zurück auf den Master-Bildschirm.

Durch das Starten des Klons kommen Sie automatisch auf die Sicht der laufenden virtuellen Maschinen.

Running-Bildschirm



Laufende virtuelle Maschinen können mit dem  Icon heruntergefahren werden.

4

Literaturverzeichnis

[1] *About Qt.* https://wiki.qt.io/About_Qt, Abruf: 2019-03-22

[2] *Microsoft Docs – Hyper-V – PowerShell-Module.* https://docs.microsoft.com/en-us/powershell/module/hyper-v/?view=win10-ps, Abruf: 2019-03-22

[3] *MQTT – Frequently Asked Questions (inkl. Beschreibung des Protokolls).* http://mqtt.org/faq, Abruf: 2019-04-03

[4] *Qt: Signals and Slots.* https://doc.qt.io/qt-5/signalsandslots.html, Abruf: 2019-03-11

[5] *This is Qt.* https://www.qt.io/what-is-qt/, Abruf: 2019-03-22

[6] *Using vmrun to Control Virtual Machines.* https://www.vmware.com/pdf/vix160_vmrun_command.pdf, Abruf: 2019-03-22

[7] *VBoxManage Manual.* https://www.virtualbox.org/manual/ch08.html, Abruf: 2019-03-20

[8] *Vergleich zwischen Typ 1 und Typ 2 Hypervisor.* https://www.computerweekly.com/de/tipp/Vergleich-zwischen-Typ-1-und-Typ-2-Den-richtigen-Hypervisor-auswaehlen, Abruf: 2019-04-03

[9] *VirtualBox Webseite.* https://www.virtualbox.org/, Abruf: 2019-03-25

[10] *VMware.* https://www.vmware.com/at.html, Abruf: 2019-03-22

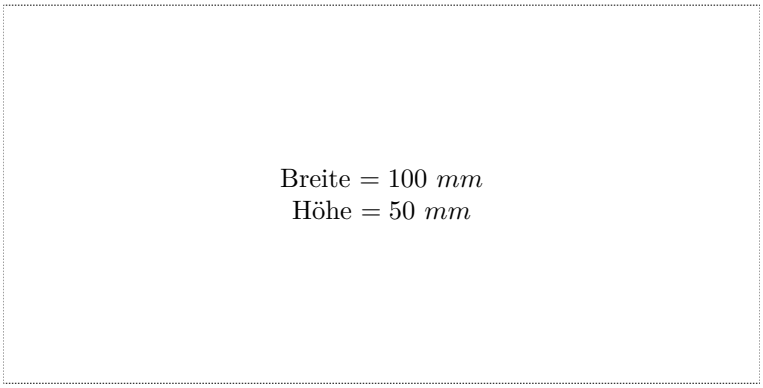
[11] *VMware: Shared-Folder einhängen.* https://pubs.vmware.com/workstation-9/index.jsp?topic=%2Fcom.vmware.ws.using.doc%2FGUID-AB5C80FE-9B8A-4899-8186-3DB8201B1758.html, Abruf: 2019-03-27

[12] *Was ist HTTP (Hypertext Transfer Protocol)?* https://www.ip-insider.de/was-ist-http-hypertext-transfer-protocol-a-691181/, Abruf: 2019-04-03

[13] Sommergut, Wolfgang: *VMware-Dateien im Überblick*. https://www.windowspro.de/wolfgang-sommergut/vmware-dateien-im-ueberblick-vmd-vmx-vmem-vmsn-vmsd-vmss, Abruf: 2019-03-22

[14] Steele, Phil ; Poggemeyer, Liza: *Microsoft Docs – Hyper-V Architecture*. https://docs.microsoft.com/en-us/windows-server/administration/performance-tuning/role/hyper-v-server/architecture, Abruf: 2019-03-22

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —

Diese Seite nach dem Druck entfernen!