PG 544: DoVinci

Dortmund Virtualized Networked Campus Infrastructure

http://ess.cs.tu-dortmund.de/DE/Teaching/PGs/dovinci

Endbericht

21. Oktober 2010

Teilnehmer:

Matthis Hainke, Nejla Karacan, Ingo Korb, Karsten Lettow, Dennis Nahberger, Maeva Obone Mba, Frederik Peiniger, Sven Radetzky, Mathias Rohde, Denijel Sakic, Matthias Wübbeling

Betreuer:

Prof. Dr.-Ing. Olaf Spinczyk, Dipl.-Inform. Jochen Streicher, Dipl.-Inform. Horst Schirmeier

Technische Universität Dortmund
Fachbereich Informatik
Lehrstuhl 12
Arbeitsgruppe Eingebettete Systemsoftware
Prof. Dr.-Ing. Olaf Spinczyk

Die Projektgruppe 544: DoVinci entwickelt ein Campus-Infrastruktur-System mit dem sich Personen auf dem Campus Anwendungen herunterladen und diese auf ihrem (mobilen) Endgerät ausführen können. Diese Anwendungen werden in Form von Appliances (Betriebssystem + Anwendung) bereitgestellt, was dazu führt, dass Installation und Wartung von der Rolle des Benutzers entkoppelt werden. Stattdessen wird auf dem Betriebssystem des Nutzers eine virtuelle Umgebung erzeugt, in welcher das Appliance-Betriebssystem, und damit auch die Anwendung, gestartet wird. Die Vorteile von Virtualisierung sind Isolation vom Nutzersystem (Schutz) sowie eine definierte Betriebssystemumgebung (Dediziertheit) für die Anwendung in der Appliance.

Dabei liegt ein Schwerpunkt bei der Verringerung des Datenvolumens sowohl für die erste heruntergeladene Appliance, als auch für weitere Appliances und Updates. Mittel hierzu sind Maßschneiderung (Weglassen nicht benötigter Dateien und Programme) und Sharing (gemeinsame Dateinutzung). Maßschneiderung optimiert Appliances bezüglich Platzbedarf und verringert so das Übertragungsvolumen des ersten Downloads. Sharing zwischen Appliances bedeutet, dass alle Appliances auf einen gemeinsamen Datenspeicher zugreifen. Beim Download einer weiteren Appliance müssen bereits vorhandene Daten nicht heruntergeladen werden. So wird das Übertragungsvolumen weiterer Appliances reduziert.

Um die oben genannten Ziele zu erreichen, werden im Rahmen von DoVinci verschiedene Techniken untersucht, verglichen und gegebenenfalls implementiert. Dazu gehören Wege zur Dienstfindung im WLAN, plattformunabhängiger Up-/Download mit Synchronisation, Maßschneiderung, Sharing und verschiedene Virtualisierungslösungen.

Inhaltsverzeichnis

1 Einführung

In der folgenden Einführung wird zunächst erklärt, was eine Projektgruppe ist. Anschließend werden die Betreuer und Mitglieder der Projektgruppe aufgeführt und die Motivation des Projekts sowie dessen wissenschaftlicher Kontext beleuchtet. Außerdem werden die daraus resultierenden Ziele angeführt und die Minimalziele erklärt, welche im Projektgruppenantrag vorgegebenen sind.

1.1 Projektgruppe

DoVinci wird entwickelt im Rahmen einer Projektgruppe (PG) der Technischen Universität Dortmund. Eine Projektgruppe dauert zwei aufeinander folgende Semester und umfasst Arbeiten im Umfang von acht Semesterwochenstunden pro Teilnehmer pro Semester.

Die Projektgruppenordnung trifft einige Aussagen über den Zweck von Projektgruppen. Demzufolge sollen die Teilnehmer in einer Projektgruppe lernen problemrelevante Methoden selbständig nachzuvollziehen und sie zu erweitern, beziehungsweise neue geeignete Methoden zu entwickeln. Außerdem lernen sie ein konkretes Problem in Teilprobleme zu zerlegen, die Teilprobleme einer koordinierten Bearbeitung zuzuführen und Lösungen von Teilproblemen zu einer Gesamtlösung zu integrieren. Darüber hinaus sollen die sachlichen und organisatorischen Schwierigkeiten bei der Analyse, Zerlegung und Bearbeitung umfangreicher Probleme deutlich werden und bewältigt werden.

Die *PG544 DoVinci*, die DoVinci entwickelt, besteht aus elf Teilnehmern sowie drei Betreuern. Die Teilnehmer sind Studierende mit Hauptfach Informatik im zweiten Studienabschnitt. Die Betreuer sind ein Professor sowie zwei Mitarbeiter der Fakultät Informatik. Die Projektgruppe erarbeitet unter dem Titel DoVinci eine Lösung zu einer im Projektgruppenantrag beschriebenen Problemstellung.

1.2 Mitglieder der Projektgruppe

Betreuer:

- Prof. Dr.-Ing. Olaf Spinczyk
- Dipl.-Inform. Jochen Streicher
- Dipl.-Inform. Horst Schirmeier

Teilnehmer:

- Matthis Hainke
- Nejla Karacan
- Ingo Korb
- Karsten Lettow

- Dennis Nahberger
- Maeva Obone Mba
- Frederik Peiniger
- Sven Radetzky
- Mathias Rohde
- Denijel Sakic
- Matthias Wübbeling

1.3 Motivation

Software für Mobilgeräte (wie Web-Tablets oder Mobiltelefone) leidet unter Beschränkungen. Oft ist der Zugriff auf spezielle Funktionalitäten eines Mobilgerätes (wie z.B. Kamera, Audio-Hardware, Bewegungssensoren oder GPS) von Laufzeitumgebungen wie Java oder .NET mobil nicht realisierbar. Um dieser Beschränkung zu entgehen, werden Anwendungen häufig an betriebssystemspezifische Schnittstellen und Treiber angebunden, wodurch sie auf eine spezifische Gerätetyp-Betriebssystem-Kombination eingeschränkt sind. Damit gestaltet sich die Entwicklung offener und flexibler Anwendungen für Mobilgeräte schwierig, da die Portierung von Software auf eine andere Plattform ein kostenintensiver und zeitaufwändiger Vorgang ist.

Ein neuer Typ von Dienstleistung für Mobilgeräte ist die verteilte, dezentrale Campus-Infrastruktur „DoVinci“, die im hier beschriebenen Projekt entwickelt wird. Das zu entwickelnde System ist offen, wodurch jeder potentielle Dienstanbieter (z.B. über einen eigenen kleinen Server mit WLAN-Access-Point) eigene Anwendungen zur Verfügung stellen kann. Die Offenheit des Systems, die Menge an unterschiedlichen Client-Systemen und die nicht vorhersehbare Vielfalt möglicher Anwendungen schließt eine Realisierung mit Java oder die Bindung an eine einzelne mobile Systemplattform aus. Ein Lösungsansatz, der die erforderliche Flexibilität und Offenheit verspricht, ist mit dem Einsatz von Virtualisierungstechnologie auf Systemebene verbunden. In Anbetracht der wachsenden Leistungsfähigkeit aktueller mobiler Geräte ist es mittlerweile auch auf kleinen, preiswerten mobilen Geräten möglich, Virtualisierungstechnologien einzusetzen. So können mehrere Betriebssystemumgebungen gleichzeitig betrieben und dabei die Isolation zwischen den einzelnen Systemen gewährleistet werden. Zudem wird von der spezifischen Gerätehardware in eine virtualisierte Hardware abstrahiert. So werden Kompatibilitätsprobleme ausgeschlossen und eine Basis für eine zukunftsfähige, offene, flexible und geräteunabhängige Informationsinfrastruktur entwickelt.

Die einzelne Applikation wird im DoVinci-System als maßgeschneiderte virtuelle Appliance (VA) implementiert und an das jeweilige mobile Endgerät ausgeliefert. Eine virtuelle Appliance beinhaltet ein Betriebssystem, benötigte Bibliotheken und Middleware sowie die eigentliche Applikation und deren Daten. Der Nutzer bekommt eine vordefinierte Umgebung bereitgestellt und hat so den Vorteil, dass er von jeder Installations- und Konfigurationsarbeit befreit ist.

1.4 Wissenschaftlicher Kontext

Virtualisierung ist im Bereich von Servern und Desktop-Systemen eine seit Jahren etablierte Technologie. Sie ermöglicht es, auf nur einer vorhandenen Hardware mehrere Betriebssystem-Umgebungen gleichzeitig und quasi-parallel, aber isoliert voneinander auszuführen. In diesen Bereichen ist Virtualisierung sowohl in kommerziellen Produkten (z.B. VMware, VirtualPC) als auch als Open Source-Lösung (Xen, KVM, VirtualBox) verfügbar.

Im Bereich eingebetteter Systeme und mobiler Geräte ist Virtualisierung ein aktuelles Forschungsthema [Hei08, BDB+08], das unter anderem Thema des IIES-Workshops ist [ES08, EN09]. Dieser Workshop wird von der Arbeitsgruppe ausrichtet, die die Projektgruppe betreut. Die Ressourcenbeschränktheit von mobilen eingebetteten Systemen stellt neuartige Anforderungen an Virtualisierungsinfrastrukturen, die in dieser Form bisher nicht existieren [Su08]. So ist beispielsweise die Größe von virtuellen Maschinen zu minimieren, um dem begrenzten Hintergrundspeicher und der vergleichsweise langsamen Netzanbindung Rechnung zu tragen. Hier können Techniken aus dem Bereich des Software-Engineering, insbesondere Ansätze zur Maßschneiderung (Weglassen nicht benötigter Dateien und Programme), zu einer deutlichen Reduktion der Größe der virtuellen Maschinen beitragen. Maßschneiderungstechniken werden von der betreuenden Arbeitsgruppe bereits erfolgreich im Bereich der Betriebssysteme [LHSP+09] und Datenhaltungssysteme [RSS+08] eingesetzt.

Im Kontext eines Campus-Informationssystems werden auch Provisioning-Infrastrukturen benötigt, also Bereitstellung, Übertragung, Verteilung und Lebenszyklus-Verwaltung maßgeschneiderter virtueller Appliances im großen Maßstab. Zugleich ermöglichen es die geplanten orts- und zeitabhängigen Dienste im Informationssystem, solche Ansätze wie push-Methoden oder publish-and-subscribe auf den Einsatz mit virtuellen Maschinen zu adaptieren.

1.5 Ziele des Projekts

Das übergeordnete Ziel des Projekts ist es eine Campus-Informationssystem-Infrastruktur zu schaffen, die auf Virtual Appliances basiert, sowie prototypische Anwendungen für das System als VA zu entwickeln. Dabei sind Ressourcenbeschränktheit (Energie [SLB07], Speicher, Rechenleistung, Netzanbindung) von Mobilgeräten [Mar07] zu beachten. Basierend auf dieser Infrastruktur werden geeignete Lösungen zur Verwaltung orts- und zeitabhängiger Anwendungen entworfen und implementiert. Deshalb sollen im Rahmen des Projekts folgende Einzelziele erreicht werden:

- Entwicklung einer Virtualisierungsumgebung für mobile, eingebettete Systeme basierend auf existierenden Technologien wie Xen [BDF+03] oder Linux/KVM [KKL+07, Su08].
- Entwicklung einer Infrastruktur für ein offenes, verteiltes Campus-Informationssystem.
- Implementierung und Evaluation von Methoden zur Maßschneiderung von virtuellen Maschinen für Mobilgeräte.

Minimalziele Die folgenden Minimalziele sind im Projektgruppenantrag festgeschrieben und stellen Mindestanforderungen an das Projektergebnis dar.

MZ1 Konzipierung und Entwicklung von Methoden zur Dienstlokalisierung in drahtlosen Netzen als Basis für ortsabhängige Dienste

MZ2 Entwicklung einer grundlegenden Virtualisierungsinfrastruktur, die VAs auf Anforderung laden kann, z.B. basierend auf Xen [BDF+03] oder Linux/KVM [KKL+07]

- MZ2a Entwicklung einer Infrastruktur zum Laden/Entladen von VMs im Hypervisor
- MZ2b Methoden zur gemeinsamen Benutzung von Ressourcen zwischen VMs (Bildschirm, Eingabegeräte etc.) unter Wahrung der Isolationsanforderungen

MZ3 Provisioning von VMs

- MZ3a Erstellung eines prototypischen Servers zur Bereitstellung und Maßschneiderung von VMs
- MZ3b Erstellung von mindestens zwei Beispielapplikationen und zugehörigen VMs

MZ4 Dokumentation von Ideen zur Maßschneiderung von VMs (z.B. in Hinblick auf Anwendungen, Dateisystem und Daten) und Implementierung mindestens einer Methode

1.6 Aufbau dieses Dokumentes

In den Abschnitten 2 bis 5 werden die Tätigkeiten die im ersten Semester stattgefunden haben beschrieben. Es wird eine Bestandsaufnahme der projektrelevanten Techniken, die von den Teilnehmern auf der Seminarfahrt untersucht wurden, aufgeführt. Weiterhin werden die Aufgabenstellungen in Form von Szenarien analysiert, aus denen sich unmittelbar die Anforderungen an die Systemkomponenten von Dovinci ergeben. Ein weiterer Abschnitt betrifft die Entwurfsentscheidungen, in dem die untersuchten, verwendeten und verworfenen Technologien aufgeführt werden. Letztlich folgt eine Beschreibung des Entwicklungsstands der Komponenten von DoVinci bei der Veröffentlichung des ersten Prototyps (MS1). Der Abschnitt schließt mit einem Zwischenfazit und einem Ausblick, in dem die zuvor gestellten Anforderungen mit den bis dahin erreichten Zielen abgeglichen werden. Darüber hinaus wird im Ausblick beschrieben welche Arbeiten für das zweite Semester eingeplant sind.

Die Entwicklungen während des zweiten Semesters werden in den Abschnitten 6 bis 12 beschrieben. Hier werden die Umsetzung der Maßschneiderung in DoVinci auf Paketebene, Dateiebene sowie Blockebene erklärt. Es wird auf die Implementierung von Sharing auf Dateiebene mithilfe eines Dateisystems namens DoVinciFS, basierend auf FUSE, sowie einem Objectstore zur gemeinsamen Datenhaltung eingegangen. Das integrieren der Teillösungen in die GUI des MS1 sowie die Verbesserungen des Clientprogramms in Sicht auf erhöhte Anwenderfreundlichkeit bildet den Abschluss der Abschnitte die die Weiterentwicklungen erläutern.

Zum Ende folgt noch ein Fazit, in dem die Entwicklungen und Evaluationen der einzelnen DoVinci-Systembestandteile zusammengefasst und bewertet werden, sowie ein Ausblick in dem Bereiche angesprochen werden, in denen im Laufe der Projektgruppe Entwicklungspotenziale hervorgetreten sind, die nicht im Rahmen des Projekts entwickelt werden konnten.

2 Seminarphase

Vor dem eigentlichen Start der Projektgruppe wurde eine Seminarphase durchgeführt. So wurden zu Beginn des Projekts einige elementar notwendige Technologien untersucht, um den aktuellen Stand der Technik zu ermitteln. Hierbei wurden insbesondere jene Technologien beleuchtet, welche für DoVinci von besonderer Relevanz sind. Zusätzlich wurden die Themen Zeit- und Projektmanagement als Grundlage einer reibungslosen Projektgruppenabwicklung untersucht. Diese Phase diente dazu die Projektteilnehmer auf einen gemeinsamen Wissensstand zu heben. Die Kurzfassungen der Seminarvorträge zu den untersuchten Technologien finden sich im folgenden Abschnitt.

2.1 Grundlagen der Virtualisierung

Virtualisierung ist die grundlegende Technologie, auf der das Projekt DoVinci aufbaut. Ihr Ziel ist es, mehrere unmodifizierte Betriebssysteme gleichzeitig auf einem Computer zu betreiben und dabei den Geschwindigkeitsverlust im Vergleich zu exklusivem Betrieb zu minimieren. Um dieses Ziel zu erreichen ist eine zusätzliche Software erforderlich, welche den gleichzeitig laufenden Systemen alleinigen Zugriff auf dem Computer vortäuscht. Tatsächlich wird diesen jedoch nur Zugriff auf die ihnen zugeteilten Ressourcen erlaubt, wie beispielsweise CPU-Zeit, Speicher oder Peripheriegeräte. Diese Software, normalerweise *Virtual Machine Monitor* oder auch *Hypervisor* genannt, lässt sich für ein beliebiges Computersystem konstruieren, sofern dieses die Anforderungen erfüllt, die in [PG74] ausgearbeitet wurden.

In diesem Fall erfolgt die Virtualisierung, indem der Hypervisor als „Über-Betriebssystem“ arbeitet und die auszuführenden Betriebssysteme ähnlich einem normalen Prozess im User-Modus des Prozessors ausführt. Dabei werden alle Befehle, die in diesem Modus nicht erlaubt sind, vom Hypervisor in Software nachgebildet. Damit der Hypervisor nicht selbst Treiber für sämtliche im System vorhandenen Geräte mitbringen muss, wird heutzutage häufig die sogenannte *Hosted Virtualization* verwendet, bei der der Hypervisor beim Gerätezugriff auf die Treiber eines der laufenden Betriebssysteme zurückgreift. Gerätezugriffe der anderen laufenden Betriebssysteme werden vom Hypervisor abgefangen und nachgebildet. Um einen konfliktfreien parallelen Betrieb ermöglichen zu können, werden diese Zugriffe auf getrennte Ressourcen abgebildet – beispielsweise das Zeichnen von Grafiken in ein Fenster, anstelle des Bildschirmspeichers oder Festplattenzugriffe in eine Datei, anstelle der Festplatte. Diese Isolation ermöglicht es DoVinci zusätzliche Software mit minimalen Auswirkungen auf das bestehende System beim Benutzer zu betreiben.

2.2 Virtualisierung in eingebetteten Systemen

Eingebettete Systeme sind informationsverarbeitende Systeme, die in ein größeres Produkt integriert sind. Sie müssen vielen Anforderungen gerecht werden, auch wenn sich diese oft gegenseitig ausschließen. Beispielsweise sind Leistungsfähigkeit und Energieeffizienz gegensätzlich, aufgrund der erforderlichen Chipfläche und dem daraus resultierenden Energieverbrauch [Mar07]. Derzeit wird untersucht, ob Virtualisierung eine mögliche Lösung ist, um

einige dieser gegensätzlichen Anforderungen zu vereinen. So untersuchen aktuell verschiedene Arbeitsgruppen den Einsatz von Virtualisierung, um das derzeitige Multichip-Design von Smartphones in ein Design mit nur einem Uniprozessor zu überführen (Konsolidierung). Die Isolation der virtuellen Maschinen soll dabei die Sicherheit des Gerätebetriebssystems gewährleisten. Des Weiteren kann dem Benutzer durch Virtualisierung erlaubt werden, ein eigenes Betriebssystem zu installieren.

Ein zentrales Problem stellt die Hardware dar, welche eingebetteten Systemen zugrunde liegt. Bisher unterstützt kein Prozessor für eingebettete Systeme Hardware-Virtualisierung. Die Performanzeinbußen, welche durch Emulation oder Vollvirtualisierung entstehen, sind für eingebettete Systeme nicht akzeptabel. Die verbleibende Alternative ist Paravirtualisierung. Bei der Paravirtualisierung werden im Gastbetriebssystem Befehle, welche die Hardware manipulieren oder auslesen möchten, durch so genannte Hypercalls ersetzt. Hypercalls rufen dann den Hypervisor, anstelle eines direkten Befehls an den Prozessor, auf. Der Hypervisor verwaltet dann für jede virtuelle Maschine einen virtuellen Hardwarezustand.

In diversen Versuchen zeigte sich für Paravirtualisierung ein Performanzverlust von nur 4-20% [HSH+08, BDB+08]. Virtualisierung ist somit ein viel versprechender Ansatz, um in eingebetteten Systemen Energie einzusparen, die Sicherheit für Nutzer und Anbieter zu verbessern, sowie die Systeme offener und flexibler zu gestalten, indem den Nutzern innerhalb der virtuellen Maschine Vollzugriff auf die Hardware gewährt wird.

2.3 Virtualisierung auf der x86-Plattform

Virtualisierung auf der x86-Plattform bereitet im Gegensatz zur Virtualisierung auf anderen Systemen einige Schwierigkeiten. Die x86-Architektur erfüllt das P&G - Theorem [PG74] nicht, welches besagt, dass alle sensitiven Befehle eines Befehlssatzes zur Gruppe der privilegierten Befehle gehören müssen. Nur dann sei eine Virtualisierung ohne Probleme möglich. Für dieses Problem gibt es unterschiedliche Lösungsansätze: Hardware-Traps und dynamische Binärübersetzung.

Hardware-Traps werden für jeden Befehl einzeln geschrieben und fangen bei einem sensitiven Befehl die Ausnahme ab und bearbeiten den Befehl, so dass keine Probleme entstehen. Bei dynamischer Binärübersetzung werden zur Laufzeit die sensitiven Befehle durch unproblematische Befehlsketten ersetzt. Der Vorteil dieser Methode ist, dass die Übersetzung einzelner Blöcke gecached werden kann. Eine weitere Lösung für das Problem besteht in hardwareunterstützter Virtualisierung. Dabei wird die Hardware so konfiguriert, dass eine Virtualisierung wieder möglich ist.

Das Fazit für die Projektgruppe ist also: In seiner Grundform ist der x86-Befehlssatz nicht zur Virtualisierung geeignet. Unter Verwendung einer der gegebenen Lösungen kann er jedoch auch virtualisiert eingesetzt werden.

2.4 Kernel-based Virtual Machine

Das Modul Kernel-based Virtual Machine (KVM) bietet auf Linux-Hostsystemen einen Hypervisor für virtuelle Maschinen an. Dieser ist als Kernelmodul in den laufenden Betriebssystemkern integriert. KVM basiert zu einem großen Teil auf dem x86-Emulator QEMU und

kann dementsprechend alle QEMU-kompatiblen virtuellen Maschinen betreiben. Sollte die Prozessorhardware des Hostsystems Hardwarevirtualisierungsunterstützung wie Intel-VT oder AMD-V anbieten, so wird diese von KVM genutzt. Andernfalls wird auf QEMU als Emulator zurückgegriffen. Bis vor kurzem kam noch das performantere KQEMU als Softwarevirtualisierung zum Einsatz. Da außer der CPU kein Gerät eine Virtualisierungsschicht anbietet, müssen alle zusätzlichen Geräte emuliert werden. Diese Emulationen werden von den QEMU-Bibliotheken zur Verfügung gestellt. Die Geschwindigkeit von Hardwarezugriffen innerhalb einer VM kann mit paravirtualisierten Treibern erhöht werden. Die Ziele der Projektgruppe können mit KVM erreicht werden, wenn Linux als Betriebssystem auf den Client-PCs vorausgesetzt wird. Im Fall von nicht unterstützten Host-Betriebssystemen oder nicht vorhandener Hardwarevirtualisierung hätte KQEMU eine akzeptable Geschwindigkeit bieten können. Der Emulator QEMU bietet diese leider nicht [Jer07].

Die direkte Integration in den Kernel erlaubt eine schlanke Codebasis des KVM-Hypervisors und eine transparente Prozessintegration in den Linux-Userspace. Somit können alle vorhandenen Kernelfunktionen (Scheduler, Speicherverwaltung, ...) genutzt werden. Seit der Aufnahme in den Vanilla-Kernelzweig (2.6.20) sind Zukunftssicherheit und Stabilität gewährleistet.

2.5 Virtual Appliances und deren Management

Unter Virtual Appliances (Kurzform: VA) versteht man eine auf einen bestimmten Anwendungsfall zugeschnittene, vorinstallierte und vorkonfigurierte Anwendung einschließlich Betriebssystem, welche gemeinsam in ein Image gekapselt sind. Der große Vorteil von VAs besteht darin, dass das Image auf jedem System läuft, welches über die entsprechende Virtualisierungslösung (z.B. KVM oder VirtualBox) zum Starten der VA verfügt [WR07]. Für DoVinci ist sowohl die Sicht ins Innere als auch das Management von VAs zentral. Deshalb sind hier grundlegende Probleme aus beiden Bereichen zu lösen. Zu den wichtigsten zählen:

- Maßschneiderung vs. Sharing, inkrementelle Updates, Nutzerdaten im Kontext von VAs, Parametrisierung und zeitlich begrenzte Ausführung (für die VA und die ausführende Virtualisierungslösung).
- Dienstfindung via WLAN, effiziente Auslieferung, Authentifizierung für Nutzer und Publisher, zeitlich begrenzte Auslieferung, Updates verbreiten, Einhaltung von Datensicherheit/Datenschutz (für das Management und die Auslieferung der VAs).

2.6 Maßgeschneiderte Distributionen

Für DoVinci wurden verschiedene Formen der Maßschneiderung untersucht. Die Bottom-Up-Maßschneiderung, bei der ein Betriebssystem von unten her aufgebaut wird, bietet Vorteile bei der Anpassung an eine bestimmte Architektur, kostet aber viel Zeit bei der Erstellung. Der Top-Down-Ansatz, bei dem eine vorhandene Distribution angepasst wird, ist oftmals nicht sehr effizient, da einzusparende Daten oftmals übersehen werden.

Jede Form der Maßschneiderung bietet gewisse Vorteile, verbunden mit Nachteilen. Es ist ein Optimierungsproblem, die Vorteile von Maßschneiderungslösungen zu verbinden und dabei die Nachteile gering zu halten. JeOSs (Just enough Operating System) sind sehr gut geeignet für die Erstellung von Appliances, jedoch bieten auch diese Systeme noch Minimierungspotential. Eine Idee ist es mit Hilfe der „Linux from Scratch“-Handbücher ein eigenes JeOS zu erstellen, welches auf den Betrieb in virtuellen Maschinen spezialisiert ist. Dieses könnte die Grundlage für künftige Appliances liefern.

SuSE Studio bietet eine sehr benutzerfreundliche Oberfläche, in der ein Image für eine virtuelle Maschine mit frei wählbaren Applikationen erzeugt werden kann. Allerdings sind die so erstellten SuSE-Distributions-Images im Allgemeinen zu groß für den Transfer via WLAN. Eine eigene Version dieses Systems könnte für Autoren von Appliances sehr interessant sein. Eine hybride Lösung der Maßschneiderung von Distributionen könnte ein gangbarer Weg sein.

2.7 Paketmanagement

Mit Hilfe von Paketmanagementsystemen lässt sich die verfügbare Software auf einem Computersystem einfach verwalten. Die Software liegt in Paketform vor, die entweder kompilierte Programme oder reinen Quellcode enthält.

Es gibt grob unterteilt in zwei Arten von Paketmanagern. Die eine implementiert nur elementare Funktionen, wie das Installieren, das Deinstallieren und die Auflistung von Softwarepaketen. Die andere implementiert weitere Funktionen wie das Nachladen von Softwarepaketen und das automatische Auflösen von Abhängigkeiten und Konflikten. Diese Paketmanager installieren bei der Installation eines Softwarepakets automatisch weitere benötigte Pakete von lokalen Installationsmedien oder dem Distributionsserver. Paketmanager garantieren die Systemstabilität dadurch, dass Pakete erst entfernt werden können, wenn sie von keinem anderen Paket mehr benötigt werden, sowie dass Pakete erst installiert werden können, wenn alle benötigten Pakete zuvor installiert wurden [EGT03]. Darüber hinaus enthalten Pakete Prüfsummen und digitale Signaturen, so dass nachgeprüft werden kann, ob ein Paket den richtigen Inhalt hat und tatsächlich vom Distributor stammt. Ein großer Nachteil der meisten Paketmanagementsysteme ist, dass die vor-kompilierten Programme nicht weiter an die Hardware angepasst werden können.

Innerhalb von DoVinci soll eine Maßschneiderung der einzelnen Softwarepakete erfolgen. So soll immer nur die notwendige Software in einer Appliance vorhanden sein. Dies kann dadurch erreicht werden, dass man bei der Installation von Software nur die unbedingt für das Paket notwendigen Pakete mitinstalliert. Weitere Pakete, die für eine Software empfohlen werden, jedoch nicht zwingend notwendig sind, werden nicht installiert. Bei der Deinstallation von Paketen werden alle Abhängigkeiten überprüft und nicht mehr benötigte Pakete aus dem System entfernt.

Paketmanagementsysteme, die Abhängigkeiten bei Installation und Deinstallation überprüfen und automatisch auflösen, sind für DoVinci somit unbedingt notwendig. Eine andere Einsatzmöglichkeit ist die Bildung von eigenen Softwarepaketen, die später vom Paketmanager verwaltet werden. Das Advanced Packaging Tool *Apt* bietet hierzu eine gute Lösung. Es ist ein Debian-basiertes Paketmanagementsystem, welches Abhängigkeiten und Konflik-

te automatisch auflösen kann. *Apt* besteht aus einer Programmbibliothek und verschiedenen Kommandozeilen-Programmen, die diese Bibliothek benutzen (z.B. *Apt-get* und *dpkg*).

2.8 Projektmanagement

Um Projekte erfolgreich managen zu können, stehen verschiedene Methoden des Projektmanagements zur Verfügung. Für DoVinci wurden verschiedene Methoden diskutiert und untersucht, um herauszufinden, welche davon für die Projektgruppe geeignet sein könnten. Der Projektmanagementzyklus unterteilt den Ablauf eines Projektes in die vier Phasen Projektstart, Planung, Realisierung und Abschluss. In der Phase Projektstart sollte ein Projektziel formuliert und das Team zusammengesetzt werden. Um ein Projektziel zu formulieren, gilt es, alle Fragen bezüglich Inhalt, Zeit und Kosten des Projekts zu beantworten. Das Teamrollenmodell von Belbin verdeutlicht, dass ein Team auch gemäß informeller Teamrollen zusammengesetzt werden kann und nicht ausschließlich gemäß funktionaler Rollen (Programmierer).

In der Phase der Planung ist ein Projektablaufplan zu erstellen und mögliche Projektrisiken sind zu untersuchen. Ein Gantt-Diagramm ist ein hilfreiches Tool zur Erstellung eines solchen Projektablaufplans. Um Projektrisiken sinnvoll zu analysieren ist eine Risikomatrix ein geeignetes Werkzeug, denn mit Hilfe dieser Matrix können alle Risiken in einem Schaubild dargestellt werden [Sei06a]. Um während eines laufenden Projekts einschätzen zu können, ob Aufgaben fristgerecht erledigt werden, ist in der Phase der Realisierung die Erstellung einer Meilenstein-Trend-Analyse eine adäquate Möglichkeit. Sollte der Fall eintreten, dass ein Problem entsteht, welches dem rechtzeitigen Abschluss einer Aufgabe im Wege steht, besteht Bedarf an einem einfachem Tool zur Problemdiskussion. Hervorragend geeignet hierfür ist die Zwei-Felder-Tafel. In der letzten Phase, dem Abschluss, die nach erfolgreichem Projektabschluss stattfindet, ist es ratsam eine Reflektionsphase durchzuführen, da diese dabei helfen kann, in zukünftigen Projekten aus den Fehlern dieses Projekts zu lernen.

2.9 Zeitmanagement

Unter Zeitmanagement versteht man die effektive Organisation und Koordinierung von Aufgaben oder Terminen in begrenzter Zeit. Das Ziel ist es, komplexe Aufgaben erfolgreich zu erledigen und gleichzeitig Belastung und Druck zu vermeiden [Sei06b]. Der Erfolg von Zeitmanagement ist stark abhängig von der eigenen Einstellungen. Neben einem Mindestmaß notwendiger persönlicher Selbstdisziplin können verschiedene Methoden zu Hilfe genommen werden, um bei der Planung und Durchführung systematisch vorzugehen.

Das Ziel ist der Maßstab, an dem sich jede Aktion messen lässt. So ist es unverzichtbar vor jeder Aktivität Ziele zu definieren. Außerdem ist es wichtig die Zeit schriftlich zu planen, da schriftlich festgehaltene Pläne das Gedächtnis entlasten und den psychologischen Effekt der Selbstmotivation zur Arbeit bewirken. Ein wesentlicher Schritt zur Gesamtplanung der Zeit ist die Planung jedes einzelnen Tages. Der Tag ist die kleinste noch überschaubare Einheit einer ausdauernden Zeitplanung. Mit der übersichtlichen ALPEN-Methode kann ein Tag in nur acht Minuten leicht vorbereitet werden.

Darüber hinaus ist es unumgänglich sämtliche Aufgaben mit Prioritäten zu versehen. Zu entscheiden ist, welche Aufgaben erstrangig, zweitrangig und nachrangig zu bearbeiten sind. Danach muss sich gezielt, in einer festgelegten Zeit, einer definierten Aufgabe gewidmet werden.

Auch Delegation ist ein erhebliches Mittel zu wirksamer Arbeitstechnik und Zeitgewinn. Bei jeder Aufgabe wird beschlossen, ob sie unbedingt selbst zu erledigen ist oder ob sie auch von jemand anderem ausgeführt werden kann. Dabei ist zu beachten, dass die Person, an die delegiert wird, die Aufgabe genau verstanden haben muss. Das Entscheidungsraster von Dwight D. Eisenhower ist ein einfaches und schnelles Hilfsmittel zur Delegation. Hier werden Prioritätsentscheidungen nach zwei Merkmalen getroffen, nach Wichtigkeit und Dringlichkeit.

Um die genannten Techniken praktisch anwenden zu können sind unterschiedliche Hilfsmittel gegeben, die man sich zu Nutze machen kann. Es gibt eine Auswahl von Zeitplanbüchern, über elektronische Organizer oder elektronische Zeitplansoftware bis hin zu Web-Organizern. Diese unterscheiden sich in Kosten und Anwendungsart, weswegen die Auswahl nach persönlichen Vorlieben und Liquidität zu treffen ist.

2.10 UPnP

Universal Plug and Play basiert auf einer Reihe von standardisierten Netzwerkprotokollen und Datenformaten. Es dient zur herstellerübergreifenden Ansteuerung von Geräten (Stereoanlagen, Router, Drucker, Haussteuerungen) über ein IP-basierendes Netzwerk. Dabei spielt es keine Rolle, ob das Netzwerk zentral verwaltet wird oder nicht. Die verwendeten Protokolle handhaben die Adressierung, die Dienstlokalisierung sowie die Ansteuerung von Geräten in einem Netzwerk.

UPnP wurde ursprünglich von einem Firmenkonsortium um Microsoft konzipiert. Der Standard wurde jedoch freigegeben und wird nun vom UPnP-Forum verwaltet. Das UPnP-Forum[1] ist eine von Microsoft gegründete Gruppe von Konzernen, Entwicklern und Firmen und umfasst aktuell knapp 900 Teilnehmer. Das Simple Service Discovery Protocol (SSDP) ist für DoVinci von besonderem Interesse, da es einfach verwendet werden kann, um nach UPnP-Geräten und ihren Diensten im Netzwerk zu suchen.

2.11 Zeroconf

Zeroconf[2] ist eine Sammlung von Protokollen, welche konfigurationsfreie Netzwerkkommunikation in Computernetzen ermöglichen sollen. Diese Protokolle bieten Lösungen für die Zuweisung von Netzwerkadressen, dezentrale Domain Name System-Konfiguration und Dienstlokalisierung in Netzen. Diese Lösungen kommen komplett ohne Benutzerkonfiguration aus. Die Protokolle sind offene und frei zugängliche Internet-Standards. Die grundlegende Entwicklung dieser Protokolle liegt insbesondere bei der Firma Apple, welche in den letzten 10 Jahren die treibende Kraft dieser Protokolle war.

[1] UPnP-Forum: `http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf`
[2] `http://www.zeroconf.org`

Besonders interessant für DoVinci ist das zur Dienstlokalisierung verwendete Domain Name System Service Discovery Protokoll [3]. Es stellt eine einfache und dennoch mächtige Lösung für die Dienstlokalisierung bereit, ohne dabei den Anwendungsentwickler in anderen Bereichen einzuschränken. Weiterhin sind von allen Protokollen plattformunabhängige Open-Source-Implementationen schon vorhanden, welche eine Implementation in neuen Projekten noch weiter vereinfachen.

[3] `http://www.dns-sd.org`

3 Analyse der Aufgabenstellung

Die Analyse der Aufgabenstellung wird in Form einer Anwendungsfallanalyse und einer anschließenden Anforderungsanalyse durchgeführt. Aus den Anforderungen ergibt sich ein Grobentwurf, der später zu mehreren Detailentwürfen verfeinert wird.

3.1 Anwendungsfallanalyse

Die erste Analyse des zu entwickelnden Systems wird in Form einer Szenarienbeschreibung durchgeführt. Es werden hierzu relevante Situationen erdacht und in den Kontext des zu entwickelnden Systems gesetzt. Diese Situationen werden als Szenarien festgehalten und die darin enthaltenen Anwendungsfälle in Diagramme überführt. Als erstes wird eine Studi-Appliance beschrieben, die von Professoren für Studenten semesterbegleitend angeboten werden könnte. Dann wird eine Werbe-Appliance vorgestellt, die von einem werbenden Unternehmen für potentielle Kunden angeboten wird und ihnen einen verlockenden Mehrwertdienst bietet. Darauf hin wird eine Kommunikations-Appliance umrissen, welche eine auf den Campus zugeschnittene, plattformunabhängige und konfigurationsfreie Kommunikation anbietet. Weiterhin wird eine Vorlesungappliance besprochen, die es Veranstaltern erlaubt vorlesungsbegleitend Software und Material für die Studenten gesammelt bereit zu stellen. Schließlich wird eine Klausur-Appliance erklärt, die es erlaubt von der Anmeldung bis zur Notenvergabe alle Vorgänge der Klausurstellung und des Ablegens einer Klausur abzubilden.

3.1.1 Studi-Appliance

Die Studi-Appliance nimmt Studenten die Arbeit ab, sich für jede Lehrveranstaltung und jedes Semester die benötigte Software zu installieren und einzurichten. Stattdessen laden sie zum Ende des vorangegangen Semesters, während einer Vorlesung, die Appliance für das kommende Semester herunter. Diese enthält voreingestellte Softwarepakete und Dateien für alle Vorlesungen und Seminare, die im kommenden Semester eingeplant sind. Zudem ist ein Paket installiert, welches die Daten der UniCard eines Studenten in der Appliance speichert. Dadurch Prüfungsanmeldungen oder Bücherausleihe erheblich vereinfacht. Damit die Integrität des Host-Systems gewahrt wird, ist die virtuelle Hardware, auf der die Appliance betrieben wird, vom Host-System isoliert.

Student Die Rolle Student agiert in diesem Szenario als Client und somit Nutzer der Appliance. Er kann die Appliance in ihrer maßgeschneiderten Form (vorkonfiguriert und installiert) vom Server herunterladen, lokal verwalten und in Betrieb nehmen. Falls eine aktualisierte Version einer Appliance auf dem Server vorliegt, kann die lokale Version aktualisiert werden, ohne dabei die eigenen Nutzerdaten zu verlieren.

Der Host-PC hat Zugriff auf die UniCard(-Daten) des Studenten, diese werden in die Appliance weitergeleitet. So kann die Appliance nur dann in Betrieb genommen werden, wenn sie von dem entsprechenden Schlüssel der UniCard freischaltet wird. Auf diese Weise ist eine Authentifizierung des Nutzers einer Appliance möglich. Der Schlüssel der UniCard kann

über ein Terminal per Netzwerkkabel oder per Cardreader in den Host-PC gelangen. Falls die UniCard abläuft (aktuell bei allen UniCards 2013 der Fall) und beim ITMC reaktiviert wird, kann der neue Authentifizierungscode eingesetzt werden, um die Appliance wieder zu verwenden. Ferner kann die UniCard-Authentifizierung auch als kryptographischer Schlüssel zur Sicherung persönlicher Daten dienen. Innerhalb der Studi-Appliance können Emails versendet werden, welche mit dem durch das ITMC bereitgestellten Schlüssel signiert sind. Außerdem ist mit der UniCard-Authentifizierung eine weitere Authentifizierung bei Übungsanmeldungen oder Prüfungsangelegenheiten denkbar (jedoch aus rechtlicher Sicht fraglich). Das auf der UniCard enthaltene Guthaben für Dienstleistungen an der eigenen Universität kann innerhalb der VA ausgelesen werden.

Die Appliance kann den aktuellen Aufenthaltsort des Host-PCs wenigstens grob bestimmen und anzeigen. Folglich ist eine Navigation vom aktuellen Aufenthaltsort zu wichtigen Anlaufstellen auf dem Campus möglich.

Studiengang- und Veranstaltungsspezifische Applikationen wie „Dave“, „Dia“ oder „Borland Together“ werden bereitgestellt. Lizenzpflichtige Software steht nur dann zur Verfügung, wenn sich der Host-PC im richtigen Hörsaal befindet. Die Lizenzfreischaltung ist also ortsabhängig.

Anwendungen innerhalb der Appliance haben Zugriff auf die Benutzerdaten. Dabei wird sichergestellt, dass die Daten nicht gelöscht werden (Persistenz) und weiterhin nicht von außen abgerufen werden können (Sicherheit), auch wenn die Appliance deaktiviert oder deinstalliert wird.

Professor / Fachschaft In diesem Szenario können verschiedene Personen die Rolle des Publishers einnehmen. Wie in Abbildung 1 zu sehen hat der Publisher die Möglichkeit, die von ihm erstellte Studi-Appliance mithilfe des Client-Programms auf den Server hochzuladen. Eine neue, aktualisierte Version derselben Appliance kann er mit geringer Netzlast auf dem Server zur Verfügung stellen. Die ältere Version steht dann gegebenenfalls nicht mehr zur Verfügung.

3.1.2 Werbe-Appliance

Die Werbe-Appliance wird als Werbegeschenk bereitgestellt und ermöglicht es einem definierten Personenkreis (Zielgruppe) kommerzielle Anwendungen für einen begrenzten Zeitraum zur Verfügung zu stellen. In dieser Appliance darf es keine Kommunikations- oder Ausführungsprobleme geben, um die der Benutzer sich kümmern müsste. Hier bietet sich der Betrieb in einer virtualisierten Umgebung an. Nach Ablauf der Nutzungsdauer kann die Appliance nicht mehr gestartet werden.

Am Beispiel eines möglichen IKEA-Planers bedeutet dies, dass ein verkleinertes Windows-Betriebssystem zusammen mit dem IKEA-Planer eine Appliance bilden und ausgeliefert werden. Der IKEA-Planer aktualisiert sich automatisch auf den neuesten Katalog und verweigert den Dienst, wenn zu lange keine Katalog-Aktualitätsprüfung vollzogen werden konnte. Der Planer kann erkennen, ob er innerhalb eines IKEA-Geschäfts betrieben wird und passt seine Funktionalität dementsprechend ortsabhängig an. Innerhalb eines IKEA-

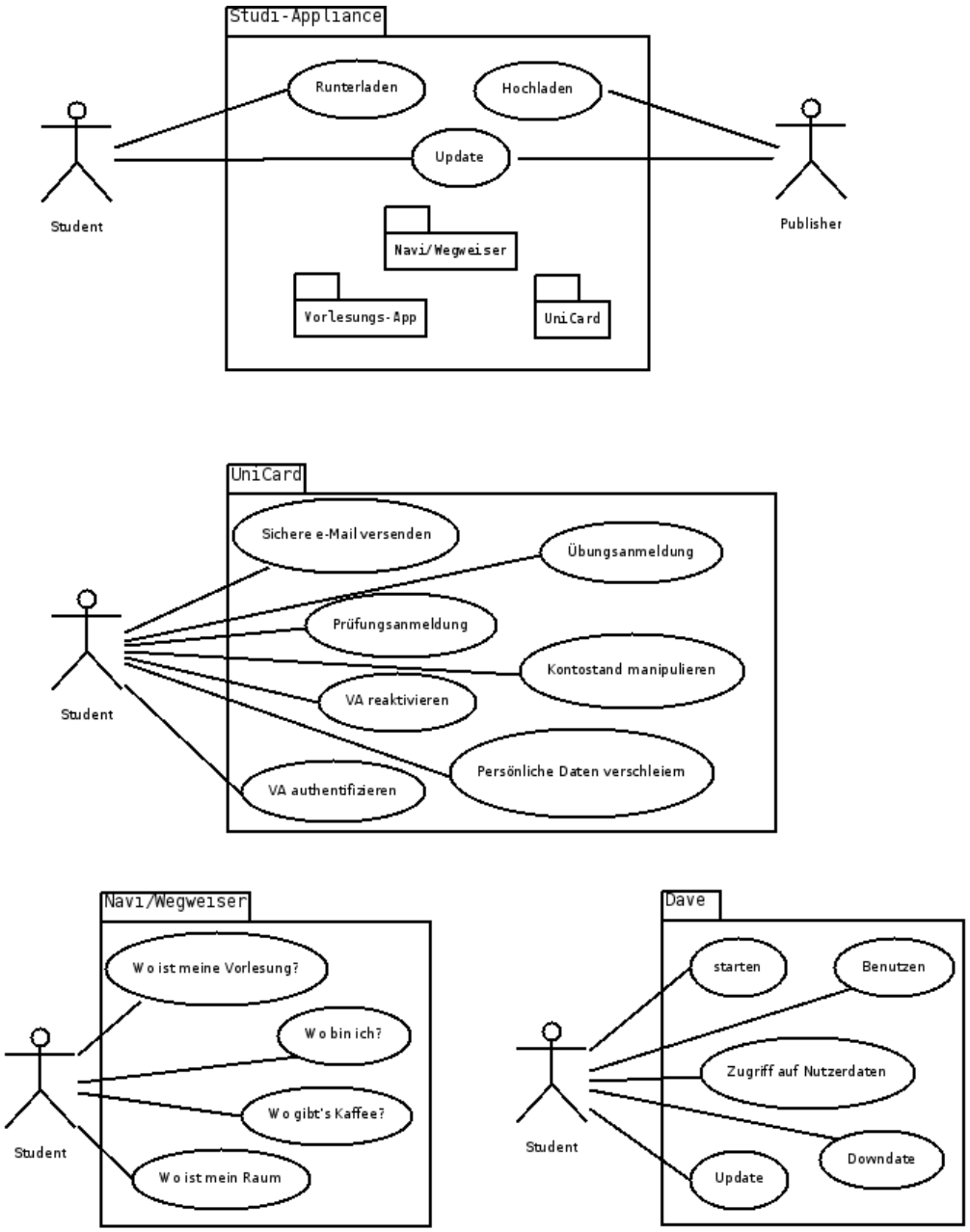

Abbildung 1: Anwendungsfalldiagramm der Studi-Appliance (siehe 3.1.1)

Hauses dient er als Navigationssystem und hilft die Waren zu finden, außerhalb fungiert er als Katalog, Einrichtungsplaner und Einkaufszettel. Das Anwendungsfalldiagramm das sich hieraus ergibt findet sich in Abbildung 2.

3.1.3 Kommunikations-Appliance

Die Kommunikations-Appliance gibt allen Universitätsangehörigen und Gästen der Universität die Möglichkeit zur direkten Kommunikation rund um den Campus. Von der vorhandenen Hardware wird bei diesem System insofern abstrahiert, als für VoIP nur die virtualisierte Audiohardware und die virtualisierte Netzwerkhardware benötigt wird. Die Entwickler von Mehrwertdiensten auf der Kommunikationsinfrastruktur müssen sich somit nicht um die Vielfalt der Endgeräte kümmern, mit denen die Kommunikation letztlich stattfindet. Die Appliance ermöglicht auch ressourcenlastige Kommunikation, wie VoIP oder Dateiaustausch.

Zusätzlich zur Telekommunikation kann der Nutzer auf Wunsch Informationen über sich auf einem eigenen Profil bereitstellen. Über diese Informationen kann z.B. der eigene Terminplan veröffentlicht und damit das Finden von Übungsgruppen oder Sitzungen vereinfacht werden. Wenn Lokalisierungsinformationen zur Verfügung stehen, kann Kommunikationspartnern auch die eigene Position angezeigt werden, wodurch Telekommunikation leicht in ein reales Gespräch münden kann. Die gerade beschriebenen Anwendungsfälle sind in Abbildung 3 abgebildet.

3.1.4 Vorlesungs-Appliance

Student Studenten können sich während der Vorlesung mit dem Server verbinden und eine vorlesungsbegleitende Appliance herunterladen, beziehungsweise eine bereits heruntergeladene Appliance updaten. Die lokale Appliance kann dann gegebenenfalls auch außerhalb des Hörsaals gestartet und die darin bereitgestellte Software benutzt werden. Diese Anwendungsfälle sind in Abbildung 4 grafisch dargestellt. Den Studenten stehen auf diese Weise vorkonfigurierte Software und vorlesungsrelevante Dateien zur Verfügung, wodurch das mobile Endgerät im Hörsaal erst wirklich sinnvoll wird. Insbesondere wird den Studenten auf diese Weise abgenommen, sich um die Lizenzierung der mitunter sehr teuren Software zu kümmern, die in vielen Vorlesungen vorgestellt oder benutzt werden. Auch ganze Betriebssysteme und deren Eigenschaften können so in der Vorlesung ausprobiert werden, ohne dass eine Installation oder Konfiguration durch den Hörer notwendig ist.

Professor / **Mitarbeiter** Der Publisher wird wohl in den meisten Fällen von einem Professor oder seinen Mitarbeitern repräsentiert. Mit Hilfe des Publisher-Tools kann er eine Appliance mit vorlesungsbegleitender Software ausstatten und Lizenzeinstellungen als Metadaten der VA auswählen, wie z.B. die Gültigkeitsdauer oder die maximale Anzahl gleichzeitig laufender Kopien der VA. Die Einhaltung dieser Lizenzeinstellungen wird durch das Client-Tool sichergestellt. Innerhalb der VA kann der Publisher daraufhin noch weitere Veränderungen und Lizenzbeschränkungen manuell einpflegen.

Die fertige VA kann schließlich zusammen mit den zugehörigen Metadaten auf den Server geladen und bereitgestellt werden. Hierbei sind weitere Daten bezüglich Zeit und Ort, wann

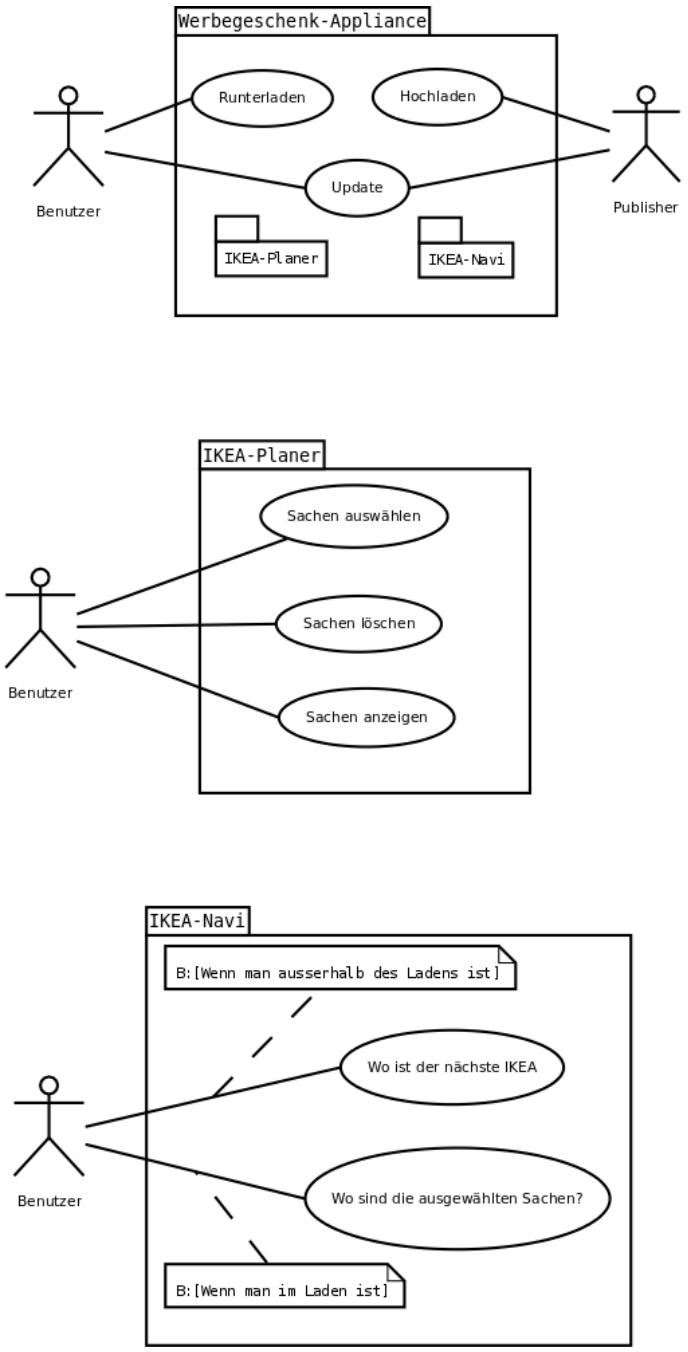

Abbildung 2: Anwendungsfalldiagramm der Werbe-Appliance (siehe 3.1.2)

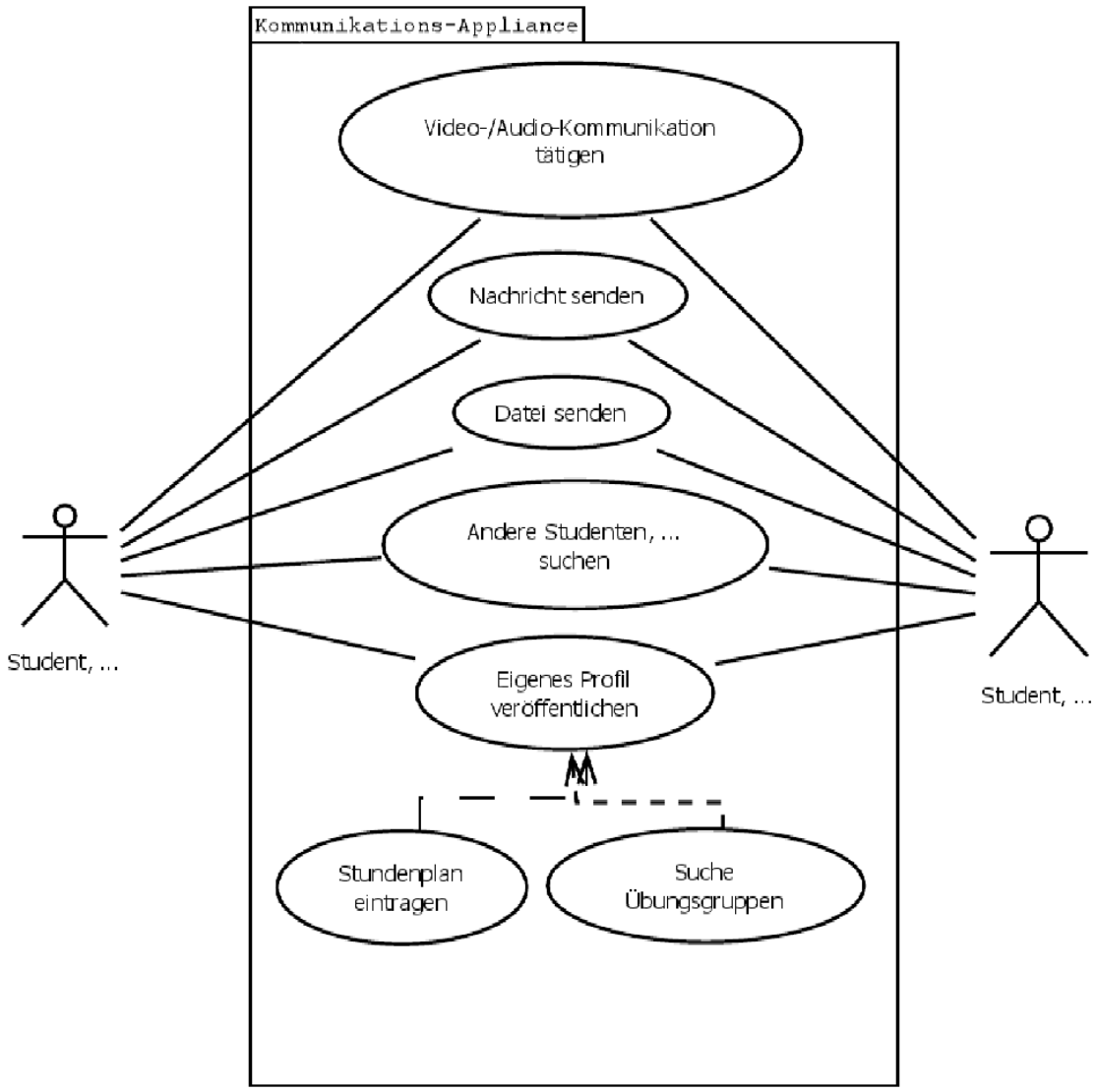

Abbildung 3: Anwendungsfalldiagramm der Kommunikations-Appliance (siehe 3.1.3)

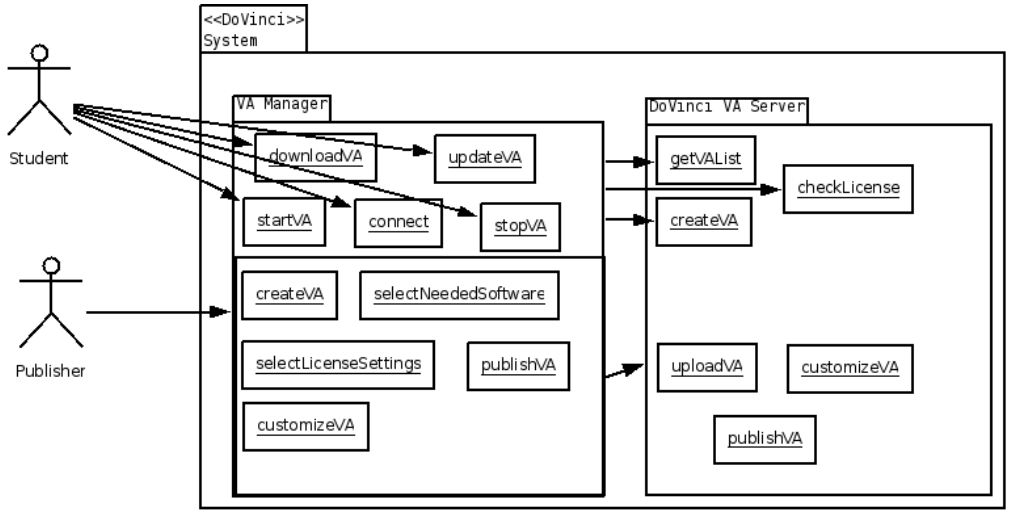

Abbildung 4: Anwendungsfalldiagramm der Vorlesungs-Appliance (siehe 3.1.4)

und wo die VA verfügbar ist, einstellbar.

3.1.5 Klausur-Appliance

Student Der Student kann mit Hilfe der Klausur-Appliance alle Aufgaben rund um das Thema Klausuren erledigen. Dazu gehört auch die Möglichkeit eine Prüfung abzulegen. Da eine sichere Authentifizierung vor dem Rechner von zu Hause aus nicht gewährleistet werden kann, muss der Student die Prüfung nach wie vor im Hörsaal ablegen. Obwohl der Student hierzu weiterhin vor Ort sein muss, ergeben sich dennoch Vorteile. Es können Aufgaben am PC gelöst werden (z.B. direktes Kompilieren in einer Entwicklungsumgebung) oder Multiple-Choice-Fragen mit zeitlicher Begrenzung gestellt werden. Beginn und Ende der Prüfung sind hierdurch für alle Teilnehmer exakt gleich. Eine rechtssichere Unterschrift wird den Ergebnissen in Form der digitalen Signatur der UniCard hinzugefügt. Die digitale Erfassung der Antworten ermöglicht zudem eine einfachere und schnellere Auswertung der Prüfungsergebnisse als bisher.

Jeder Student kann nur die eigenen Klausurergebnisse einsehen. Dies führt zu verbessertem Datenschutz für die Studenten. Weiterhin wäre es unter Umständen möglich, die Klausureinsicht von zu Hause aus zu ermöglichen. Der Zeitraum für die Klausureinsicht (heute meist nur an einem festen Tag) könnte damit beliebig ausgeweitet oder eine Kopie der Klausur jederzeit eingesehen werden.

Prüfungsprotokolle liegen erstmals in digitaler Form vor und können für einen bestimmten Zeitraum innerhalb der Appliance heruntergeladen werden (z.B. nach dem 2. Prüfungstermin der jeweiligen Prüfung). Die Prüfungsprotokolle stehen nur Studenten zur Verfügung, die für die Prüfung angemeldet sind.

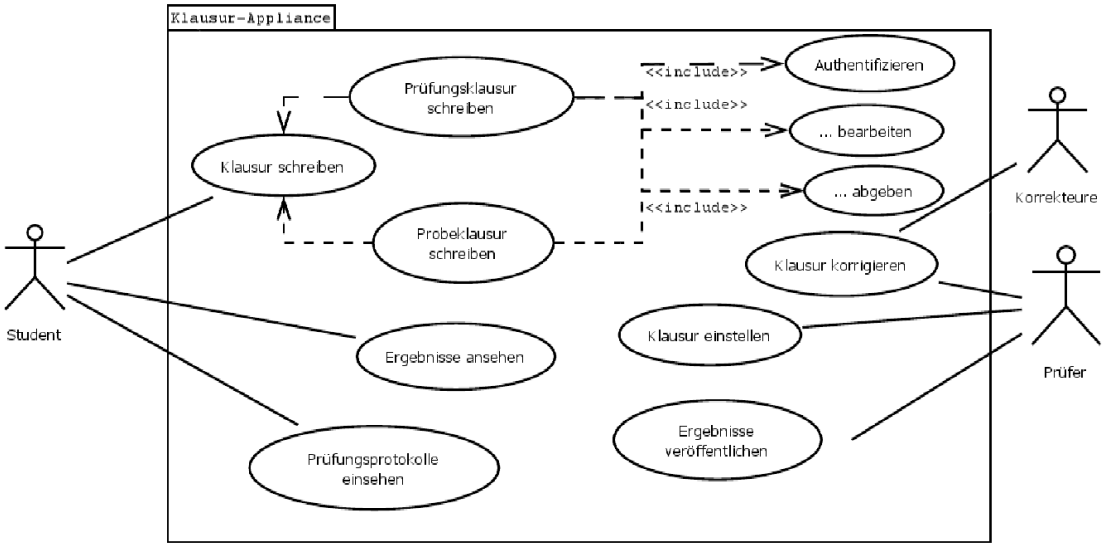

Abbildung 5: Anwendungsfalldiagramm der Klausur-Appliance (siehe 3.1.5)

Prüfer und Korrekteure Für das Stellen und Korrigieren der Klausuren ergeben sich diverse Vorteile. Die Verwendung standardisierter Klausurvorlagen ermöglicht es Klausuren einfach zu erstellen oder nachträglich abzuändern. Die Prüfungsappliance erhält erst zu Beginn der Vorlesung virtualisierte Netzwerkhardware, wodurch die digitale Vervielfältigung erst vor Ort im Hörsaal stattfindet. Dadurch sind die Prüfungsfragen bis zum Prüfungstermin vor fremdem Zugriff geschützt. Nach der Prüfung ist die Korrektur und Aufgabenverteilung unter den Korrekteuren einfacher. Klausurergebnisse lassen sich gegebenenfalls automatisch berechnen und anschließend bekanntgeben.

Ein Anwendungsfalldiagramm des soeben vorgestellten Szenarios ist in Abbildung 5 zu finden.

3.2 Grobentwurf

Aus den Szenarien ergibt sich ein Grobenwurf für die Infrastruktur von DoVinci. Der erste Grobentwurf des Systems sieht eine Aufteilung in Server, Client und Publisher-Tool vor. Der Server stellt die Appliances per Dienstfindung zum Download bereit. Der Client leistet sowohl den Download und Upload, als auch die lokale Verwaltung von Appliances. Das Publisher-Tool unterstützt letztlich die Erzeugung der Virtual Appliances mit Techniken zur Maßschneiderung (siehe Abbildung 6).

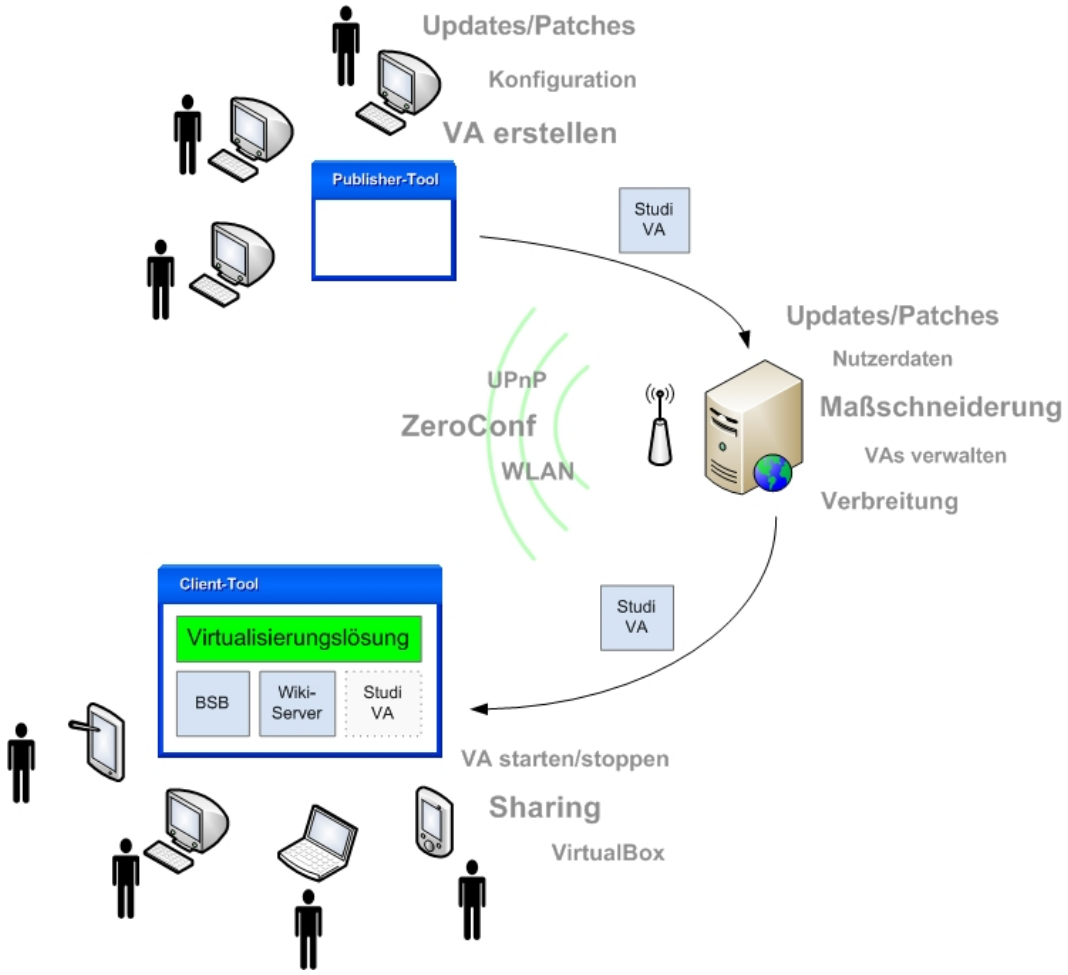

Abbildung 6: Grobentwurf des zu entwickelnden Systems

Die Anforderungsanalyse wird sich an den im Grobentwurf vorkommenden Komponenten maßgeblich orientieren.

3.3 Anforderungsanalyse

In diesem Abschnitt werden die Anforderungen an die einzelnen Systemkomponenten sowie an die Appliances mit Bezug auf die Anwendungsszenarien beschrieben. An den Anforderungen stehen in Klammern jeweils die Szenarien aus denen die Anforderung hervorgeht.

3.3.1 Anforderungen an den Client

C01 **Dienstfindung** **Priorität: 1**

Der Client soll automatisch die aktuell verfügbaren DoVinci-Server im Netzwerk finden und die dort verfügbaren Appliances auflisten. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.4)

C02 **Integration der Virtualisierungslösung** **Priorität: 1**

Für den Benutzer sollte die Einbindung der Virtualisierungslösung nicht sichtbar geschehen. Die Appliances sollen einfach gestartet und beendet werden können. (siehe Anwendungsszenario: 3.1.4)

C03 **VA Verwaltung** **Priorität: 1**

Die Verwaltung von VAs erfordert, dass der Client dem Benutzer einige Standardoptionen bietet. So soll es die Möglichkeit geben eine VA zu installieren und zu löschen. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.4)

C04 **Zeitbeschränkte Laufzeit von VAs** **Priorität: 1**

Der Client soll Appliances mit zeitlich beschränkter Laufzeit unterstützen. Dies kann z.B. im Falle lizenzierter Komponenten innerhalb einer VA auftreten. Im Falle einer Zeitbeschränkung sollte der Benutzer rechtzeitig darauf hingewiesen werden. (Siehe Anwendungsszenarien: 3.1.2, 3.1.4, 3.1.5)

C05 **VA Updates** **Priorität: 1**

Der Client soll das Updaten von VAs unterstützen. Das beinhaltet auch das regelmäßige Nachfragen am Server, ob Updates für lokale VAs vorhanden sind. Weiterhin muss besondere Rücksicht auf Sharing genommen werden, falls z.B. zwei VAs sich Dateien teilen, diese aber nur bei einer davon ein Update erhalten. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5)

C06 **Sharing** **Priorität: 1**

Die Anwendung soll das Sharing von Daten zwischen einzelnen VAs unterstützen, um die Übertragungsgrößen zwischen Client und Server zu minimieren und Ressourcen (wie z.B. die Festplatte) zu schonen, jedoch ohne dabei die Performance des Client-Rechners maßgeblich negativ zu beeinflussen. Die zeitgleiche Nutzung von mehreren Appliances soll möglich sein. Die Art und Weise des Sharings ist noch im Laufe der Entwicklung zu bestimmen und zu evaluieren. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5)

C07 **Authentifizierung von Benutzern** **Priorität: 1**

Der Client soll die Authentifizierung von Benutzern am Server unterstützen, dieses gegebenenfalls in verschiedenen Varianten (z.B. per Passwort, Benutzername + Passwort oder Lizenzschlüssel). (Siehe Anwendungsszenarien: 3.1.1, 3.1.4)

C08 **Nutzerdaten lokal speichern** **Priorität: 1**

Es ist wünschenswert, dass eine Appliance Nutzerdaten auf der lokalen Festplatte hinterlegen kann. Der Benutzer sollte in der Lage sein lokale Verzeichnisse zur Speicherung benutzerspezifischer Daten aus der VA anzulegen und zu verwalten. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.4)

C09 **Nutzerdaten zentral speichern** **Priorität: 3**

Nutzerdaten sollten optional auch VA-übergreifend gespeichert werden können, so dass diese auch nach Deinstallation einer VA weiterhin verfügbar sind. Zusätzlich soll es möglich sein, diese Daten auch extern (z.B. auf einem externen Server) abzulegen. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.4)

C10 **Benutzerinterface** **Priorität: 3**

Um dem Benutzer den Umgang mit dem DoVinci-Client zu vereinfachen, sollte das Benutzerinterface den gängigen Standards des jeweiligen Betriebssystems entsprechen, auf dem es läuft. Eine intuitive Bedienung soll gewährleistet sein.

C11 **Plattformunabhängigkeit** **Priorität: 3**

Die Client-Anwendung sowie die darin enthaltene Virtualisierungslösung sollte plattformunabhängig sein.

C12 **USB-Boot** **Priorität: 3**

Das Client-Tool soll über USB gebootet werden können. Hiermit soll erreicht werden, dass das Gast-System überhaupt nicht beeinflusst wird. Ein Modus wie er von Live-CDs bekannt ist, wäre denkbar.

3.3.2 Anforderungen an das Publisher-Tool

P01 **VA-Verwaltung** **Priorität: 1**

Das Publisher-Tool soll in der Lage sein mehrere VAs zu verwalten. (siehe Anwendungsszenarien: 3.1.1, 3.1.5)

P02 **VA-Konfiguration** **Priorität: 1**

Damit die VA nicht nur Basisinstallationen, sondern auch fertig konfigurierte Programme enthält, soll es möglich sein die VA zu starten, sie zu konfigurieren und sie wieder abzuspeichern. Es könnte auch ein Konfigurationsskript angeboten werden, welches formularartig ausgefüllt wird. (siehe Anwendungsszenarien: 3.1.1, 3.1.4, 3.1.5)

P03 **Performanz** **Priorität: 1**

Das Bauen und Uploaden von VAs soll auch auf leistungsschwächeren Rechnern in akzeptabler Zeit möglich sein. (Siehe Anwendungsszenarien: 3.1.1, 3.1.4)

P04 **Kosten** **Priorität: 1**

Es sollen dem Publisher durch die Nutzung der DoVinci-Infrastruktur keine monetären Kosten entstehen. (siehe Anwendungsszenario: 3.1.4)

P05 **Usability** **Priorität: 1**

Es soll für den Publisher sehr einfach sein eine erste lauffähige VA anzubieten. Auch für Laien soll es möglich sein, eine VA zu erzeugen, welche alle Vorteile der DoVinci-Infrastruktur nutzt. (siehe Anwendungsszenario: 3.1.4)

P06 **VA-Update** **Priorität: 1**

Es soll die Möglichkeit geben, eine VA geringfügig abzuändern und die aktualisierte Version in Form eines Patches (nur die Änderungen zur Vorgängerversion werden gespeichert) bereitzustellen. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.4, 3.1.5)

P07 **VA-Zusammenstellung** **Priorität: 2**

Das Publisher-Tool soll die Möglichkeit bieten, eine VA aus einer Programmliste „zusammenklicken“ zu können. (Siehe Anwendungsszenarien: 3.1.3, 3.1.4, 3.1.5)

P08 **Plattformunabhängigkeit** **Priorität: 2**

Das Publisher-Tool soll für mehrere Plattformen verfügbar sein. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5)

P09 **Plattformunabhängigkeit VA** **Priorität: 2**

Das innerhalb einer VA verwendete Betriebssystem soll frei wählbar sein. (siehe Anwendungsszenario: 3.1.4)

P10 **Zeitliche Begrenzung** **Priorität: 2**

Es soll möglich sein, die Laufzeit einer VA bei den Benutzern zeitlich einzuschränken. (siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.4, 3.1.5)

P11 **Kapselung** **Priorität: 2**

Der Betrieb des Publisher-Tools soll das vorhandene System nicht gefährden oder schädigen.

P12 **Automatisches Update** **Priorität: 2**

Das Publisher-Tool soll sich selbstständig updaten, wenn der Server ein Update bereithält.

P13 **Zielgruppe** **Priorität: 3**

Es soll möglich sein, die Zielgruppe einer VA einzugrenzen. Dies kann bestimmte Hörsäle, Uhrzeiten oder authentifizierbare Personenkreise betreffen. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5)

P14 **VA-Statistiken** **Priorität: 3**

Es soll möglich sein, Statistiken über die Nutzung der eigenen VAs abzufragen (z.B. Anzahl Zugriffe, Anzahl der Benutzer, Aufenthaltsorte der Benutzer). (Siehe Anwendungsszenarien: 3.1.3, 3.1.4)

3.3.3 Anforderungen an den Server

MW1 **Dienstangebot** **Priorität: 1**

Der Server muss Dienste (VAs) bereitstellen, katalogisieren, nach Kriterien sortieren, speichern und archivieren können. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5)

MW2 **Erreichbarkeit** **Priorität: 1**

Der Server sollte dauerhaft erreichbar sein und die Dienste anbieten können. Eine korrekte Datenübertragung zum Client sollte über ein geeignetes Kommunikationsprotokoll (z.B. TCP) mit Checksummenprüfung (z.B. per MD5-Hash) sichergestellt werden. (Siehe Anwendungsszenarien: 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5)

MW3 **Maßschneiderung** **Priorität: 1**

Der Server soll die Optimierung der Größe der zu übertragenden VA-Images und deren Updates durch die Nutzung von Maßschneiderungsverfahren unterstützen. (Siehe Anwendungsszenarien: 3.1.1, 3.1.3, 3.1.4)

MW4 **Sharing** **Priorität: 1**

Der Server sollte, soweit möglich, auf dem Client schon existierende Daten nicht erneut übertragen müssen. Zu diesem Zweck muss er bereits vorhandene Komponenten ermitteln, die Differenzen berechnen und ausschließlich diese übermitteln. (Siehe Anwendungsszenarien: 3.1.1, 3.1.3, 3.1.4)

MW5 **VA-Verwaltung** **Priorität: 2**

Die vorhandenen und angebotenen VAs sollten verwaltet werden können. Hierzu gehören insbesondere eine mögliche Lizenzierung einzelner Komponenten, Zugangsbeschränkungen sowie die nachträgliche Änderung (Hinzufügen oder Entfernen von installierter Software) einer Appliance. Des Weiteren sollten die Maßschneiderung und das Sharing konfiguriert und komplette VAs aus dem System entfernt werden können. (Siehe Anwendungsszenarien: 3.1.1, 3.1.3, 3.1.4, 3.1.5)

MW6 **Datenintegrität** **Priorität: 2**

Die Herkunft einer Appliance sollte offensichtlich dargestellt werden und möglicherweise mit Hilfe von Zertifikaten signiert werden können. (Siehe Anwendungsszenarien: 3.1.1, 3.1.4, 3.1.5)

MW7 **Datensicherheit** **Priorität: 2**

Die Daten innerhalb einer VA sollten entsprechend der Möglichkeiten vor Datendiebstählen geschützt werden. Hierzu gehören zum einen die Benutzerdaten des Client, zum anderen Lizenzierungsdaten von Software, welche vor dem Auslesen zu schützen sind. (Siehe Anwendungsszenarien: 3.1.1, 3.1.3, 3.1.4, 3.1.5)

MW8 **Nutzerdaten-Verwaltung** **Priorität: 3**

Anfallende Benutzerdaten bei der Arbeit mit einer VA sollten an zentraler Stelle gespeichert werden können, um eine Verwendung in einer weiteren VA oder über die Lizenzdauer einer VA hinaus zu erlauben. Ebenfalls kann es sinnvoll sein, Informationen über das vom Benutzer verwendete System zu sammeln, um VAs und deren Verwendung optimieren zu können. Zusätzlich könnte eine direkte Übermittlung von Nutzerdaten (Statistiken, Arbeiten, etc.) an den Publisher ermöglicht werden. (Siehe Anwendungsszenarien: 3.1.1, 3.1.3, 3.1.4, 3.1.5)

3.3.4 Anforderungen an die VA

A01 **Universeller Einsatz** **Priorität: 1**

Die VA soll bei allen Nutzern mit x86-PCs und wenigstens 1GHz-Prozessortakt sowie 512MB Arbeitsspeicher funktionieren. (Siehe Anwendungsszenarien: 3.1.1, 3.1.4, 3.1.5, 3.1.2)

A02 **Peripheriezugriff** **Priorität: 1**

Es soll möglich sein aus der VA heraus auf PC-Peripheriegeräte zugreifen zu können. Insbesondere USB-Sticks und Drucker sollen in der VA verfügbar sein. (Siehe Anwendungsszenarien: 3.1.1, 3.1.3)

A03 **Performanz, Stabilität und Ressourcennutzung** **Priorität: 1**

VAs sollten mit ähnlicher Geschwindigkeit, Ressourcennutzung und Stabilität betrieben werden können, wie dies ohne Virtualisierung möglich wäre. Der Einfluss der VA auf das Host-System, z.B. durch Festplatten-, Arbeitsspeicher- oder Prozessor-Overhead durch Virtualisierung, soll möglichst gering sein. (Nichtfunktional, nicht von einem expliziten Szenario gefordert)

A04 **Isolation** **Priorität: 1**

Die VAs sollen komplett voneinander isoliert laufen. Insbesondere das Gast-Betriebssystem, aber auch andere VAs, dürfen nicht in ihrem Betrieb gestört werden. (Nichtfunktional, nicht von einem expliziten Szenario gefordert)

A05 **Systemverträglichkeit** **Priorität: 1**

Die VA soll andere Dienste im Netzwerk nicht stören und auch sonst kein von außen sichtbares schädliches Verhalten zeigen (z.B. Bildschirmflackern, USB-Sticks formatieren). (Nichtfunktional, nicht von einem expliziten Szenario gefordert)

A06 **Zugriffsschutz** **Priorität: 2**

Damit die VA nicht von unbefugten Nutzern verwendet werden kann, soll ein Passwortschutz optional integriert werden können. (Siehe Anwendungsszenarien: 3.1.1, 3.1.4, 3.1.5)

A07 **Safety & Security** **Priorität: 2**

Nutzerdaten, die innerhalb der VA erzeugt oder verwendet werden, sollen vor unbefugtem Zugriff geschützt werden können. (Siehe Anwendungsszenarien: 3.1.1, 3.1.5)

A08 **Fehlerbenachrichtigung** **Priorität: 3**

Die VA soll Fehler protokollieren und diese an den VA-Server übertragen können. (Generelle Forderung, nicht von einem expliziten Szenario gefordert)

4 Entwurfsentscheidungen

Eine Vielfalt an Hardwarearchitekturen, Virtualisierungslösungen und Betriebssystemen bietet eine Vielzahl an Kombinationsmöglichkeiten. Eine genauere Beleuchtung einzelner Komponenten ermöglicht eine engere Auswahl und somit im späteren Vorgehen eine stärkere Fokussierung auf Technologien mit einem für das Projekt optimalen Nutzen.

4.1 Projektrelevante Rechnerarchitekturen

Virtualisierung im Bereich der mobilen Computer (z.B. Laptops und Netbooks) und der eingebetteten Systeme (z.B. Mobiltelefone) ist nur auf bestimmten Architekturen möglich, da besondere Anforderungen erfüllt sein müssen. Als weit verbreitete Architekturen kommen hier für den oben genannten Anwendungsbereich nur x86, ARM, PowerPC und MIPS in Frage. Im Folgenden wird die Frage geklärt, welche Architekturen für das Projekt DoVinci besonders zu beachten sind.

4.1.1 x86

Aufgrund der großen Verbreitung der x86-Architektur bei Laptops und Netbooks sowie zunehmend im Bereich von eingebetteten Systemen ist die Unterstützung dieser Architektur für DoVinci enorm wichtig und stellt somit die entscheidende Architektur dar, die in jedem Fall unterstützt werden muss. Für die vollständige Virtualisierung stehen mehrere Lösungen zur Verfügung, wie KVM, VirtualBox, VMware und Xen.

4.1.2 ARM

Die ARM-Architektur ist weit verbreitet im Bereich der eingebetteten Systeme (z.B. Mobiltelefone, PDAs und Netbooks) und ist mit Hinblick auf eine zunehmende Nutzung dieser Endgeräte in naher Zukunft von besonderer Bedeutung. Mittlerweile gibt es eine lauffähige Lösung des Hypervisors Xen auf ARM, welche auf der ARM 9 Architektur getestet ist[4]. Xen hat jedoch den entscheidenden Nachteil keine Hosted-Virtualisierungslösung zu sein. Es kann also nicht in einem vorhandenen Betriebssystem installiert werden, sondern liegt als Basis unter allen Gast-Betriebssystemen, von denen eines die Steuerung von Xen übernehmen darf. Auf Mobiltelefonen oder vergleichbaren Endgeräten, welche das DoVinci System einsetzen sollen, würde es folglich eine Neuinstallation des Betriebssystems nach sich ziehen. Es ist jedoch nicht auszuschließen, dass sich dies in Zukunft ändert.

4.1.3 PowerPC

Die Verbreitung der PowerPC-Architektur ist unter den für DoVinci relevanten Endgeräten (Laptops, Netbooks und Mobiltelefone) eher gering, somit ist PowerPC-Unterstützung weniger interessant für das Projekt. Durch vorhandene Lösungen von KVM und Xen für PowerPC ist eine spätere Ergänzung um diese Architektur jedoch nicht ausgeschlossen.

[4] http://www.xen.org/files/xensummitboston08/SecureXenARM_XenSummitNorthAmerica2008.pdf

4.1.4 MIPS

Ähnlich wie PowerPC hat MIPS nur eine sehr geringe Verbreitung auf den relevanten Endgeräten und konnte sich hier insbesondere gegen ARM nicht durchsetzen. Für DoVinci ist diese Architektur somit vorerst nicht interessant.

4.2 Virtualisierungslösungen

Virtualisierungslösungen können sich in verschiedenen Aspekten stark unterscheiden. KVM und QEMU sind quelloffene Projekte, von VirtualBox sind Teile des Codes closed source, und VMware ist ein kommerzielles Produkt, dessen Quellen gänzlich verschlossen bleiben[5]. Je nach Plattform müssen unterschiedliche Hypercalls in den Hypervisor eingefügt werden, um kritische Operationen auf virtualisierter Hardware auszuführen. Ein weiteres Merkmal ist die Bedienbarkeit. So ist VMware (Player 3.0) mit seiner GUI sehr benutzerfreundlich, jedoch nur schwer auf der Kommandozeile zu bedienen. Dagegen kann eine KVM-Maschine in einem einzigen Kommandozeilenbefehl konfiguriert und gestartet werden. VirtualBox bietet beide Wege an und verwaltet Maschinen und Hintergrundspeicher in Datenbanken über eine GUI oder wahlweise in einem Tool *VBoxManage*. Eine Gemeinsamkeit der drei betrachteten Virtualisierungslösungen ist, dass sie ausschließlich virtuelle x86-Hardware anbieten. Eine Portierung auf ARM hat bisher nicht stattgefunden, wodurch Virtualisierung auf dem Markt der eingebetteten Systeme bisher kaum Aufmerksamkeit erfährt. Der Frage, ob auch starke Unterschiede in der erbrachten Performanz oder der Leistungsaufnahme bestehen, wird in diesem Projekt nachgegangen und ist in der Evaluation der Test-Appliances zu finden.

4.2.1 KVM / QEMU / KQEMU

Die KVM-Suite hat den Nachteil nur unter Linux betrieben werden zu können, da kritische Teile in dessen Kernel integriert sind. Dennoch ist dieses gleichermaßen auch ein Vorteil, da nahezu jeder Linux-basierte PC bereits über eine installierte Version dieser Virtualisierungslösung verfügt. Ein gravierender Nachteil hingegen ist, dass KVM ausschließlich in Verbindung mit Hardware-Virtualisierungsuntertützung betrieben werden kann. Somit ist der Betrieb von KVM auf solche Rechner beschränkt, deren Prozessor Virtualisierung unterstützt. Bei den Prozessoren der Firma AMD ist die Unterstützung seit 2006 in Prozessoren für den Consumerbereich und darüber gegeben. CPUs von Intel unterstützen Virtualisierung seit 2005, jedoch nicht durchgängig in allen Produkten. Allerdings bieten die leistungsstärkeren, in Netbooks installierten Intel Atom-Prozessoren Virtualisierungsunterstützung an. Diese ist jedoch oft im BIOS nicht freischaltbar. Darüber hinaus bietet die Firma VIA eine Virtualisierung unterstützende (ab Stepping 3) Prozessorreihe namens Nano an.

Steht auf einem Host-PC keine Virtualisierungsunterstützung zur Verfügung, wird der Emulator *QEMU* eingesetzt, dessen enorme Performanzeinbußen nicht hinnehmbar sind. Das Projekt *KQEMU* welches *QEMU* beschleunigt, ist leider beendet und wird nicht weiter unterstützt. Das Projekt *KQEMU* empfiehlt den Wechsel zu VirtualBox.

[5] `http://www.vmware.com/de/`

4.2.2 VirtualBox

Die Virtualisierungslösung *VirtualBox* ist auf vielen Host-Betriebssystemen (unter anderem Windows, Linux und Mac OS X) lauffähig und erlaubt beliebige Gast-Betriebssysteme auf x86-Maschinen. *VirtualBox* wird von der Firma innotek entwickelt und seit dem Januar 2007 ist der Großteil der Quellen (unter der GPL) Open Source verfügbar[6]. Die Teile der Quellen, die nicht unter der GPL stehen, sind erweiterte Features, welche, laut Aussage von innotek, vor allem für Enterprise Kunden interessant sind. Darunter fallen paravirtualisierte Treiber für USB-Controller, Unterstützung von *Remote Display Protocol* und die USB-über-RDP-Unterstützung.

Die Open-Source-Variante von *VirtualBox* enthält dennoch eine sehr große Anzahl von Features, welche diese für DoVinci interessant machen. Volle Unterstützung von Hardware-Virtualisierung sowie eine große Anzahl an Management-Funktionen für virtuelle Maschinen inklusive Unterstützung so genannter Snapshot Bäume. Das heißt, selbst mehrere Overlays über einer virtuellen Maschine werden direkt von der vorhandenen GUI und dem Hintergrunddienst von *VirtualBox* unterstützt. Die Existenz einer gut funktionierenden plattformunabhängigen GUI ist ein weiterer Punkt, welcher *VirtualBox* für DoVinci interessant macht.

4.2.3 VMware

Die Firma VMware bietet eine ganze Produktreihe von Virtualisierungslösungen an. Von High-End-Virtualisierung für Server-Rechenzentren bis hin zur Virtualisierung auf ARM-basierten mobilen Geräten im Alphastadium sind alle Kundenkreise abgedeckt. Leider verbreitet VMware seine Produkte nicht quelloffen und die meisten Produkte nur gegen Lizenzgebühren. Einige kostenlose Produkte (Player, Server) konnten im Rahmen dieses Projekts jedoch getestet werden.

4.3 Gastbetriebssysteme

Die Wahl des Gastsystems ist für DoVinci sehr wichtig, da die Größe des Gastsystems die Größe der VA maßgeblich beeinflusst. Es ist folglich wichtig, dass sich das Gastsystem maßschneidern lässt und dass der Betrieb in einer virtuellen Maschine gegebenenfalls mit paravirtualisierten Treibern möglich ist.

4.3.1 Windows

Bei der Installation von Windows werden sehr viele für eine Appliance unnötige Funktionen mitinstalliert. Das führt dazu, dass ein relativ umfangreiches System entsteht, welches für den Transport per WLAN übertragen werden muss. Leider lässt sich Windows nur schwer maßschneidern und individuell einrichten. Tools wie *nlite* (für *Windows XP*) und *vlite* (für *Windows Vista*) erzeugen nur Installationsmedien, die eine schlankere Installation erlauben.

[6] Oracle-Webseite: `http://www.virtualbox.org/wiki/VirtualBox`

Windows 7 Das neueste Produkt von Microsoft auf dem Betriebssystemmarkt ist schlanker als der Vorgänger *Windows Vista*. Die Grundinstallation mit 5,3GB steht der von *Windows Vista* mit ca. 6GB gegenüber, allerdings kommen noch Auslagerungs- und Ruhezustandsdateien hinzu, welche etwa 1,5GB zusätzlichen Speicherplatz verbrauchen[7][8]. Es gibt zu *Windows 7* noch kein Tool wie *nlite* und *vlite*.

Windows XP Die älteste unter den modernen Windows, aber auch die kleinste (Grundinstallation 1,5GB + Auslagerungsdatei), unter welcher die meisten Programme noch funktionieren, ist Windows XP. Viele Programme setzen allerdings Service Pack 2 oder eine aktuelle .NET-Laufzeitumgebung voraus. Konservativ geschätzt sollte also mit ungefähr 2GB gerechnet werden. Diese 2GB stellen allerdings die Basisinstallation ohne zusätzliche Applikationen dar. Durch *nlite* ist es möglich eine *Windows XP* Installation mit einer Größe von 537 MB[9] zu erhalten. Die Größe einer Installation von *Windows Vista* mit *vlite* steht diesem im Vergleich mit 1,4GB gegenüber, also etwas mehr als dem doppelten eines mit *nlite* installierten *Windows XP*.

Dies führt zu dem Schluss, dass Windows als Gastsystem für DoVinci ungeeignet ist und nicht weiter betrachtet werden sollte. Jedoch kann der Betrieb eines Windows-Gastes wegen der großen Verbreitung des Betriebssystems nicht ausgeschlossen werden.

4.3.2 Debian / Ubuntu / JeOS

Ubuntu JeOS Die auf Debian basierende Distribution Ubuntu beinhaltet in ihrer Serverversion die Möglichkeit bei der Installation die Option *minimale virtuelle Maschine* zu wählen. In der Version *hardy* ist das so erzeugte Image sehr klein und beinhaltet kaum unnötige Pakete (Dokumentation zu Perl und Python). Allerdings verwendet *hardy* die Kernelversion 2.4 und erfüllt damit nicht die Anforderungen, die an ein modernes Betriebssystem gestellt werden. Dieselbe Installation mit der aktuelleren Version *karmic* erzeugt hingegen ein weitaus größeres Image.

Gnome in Debian Die grafische Oberfläche KDE ist nicht sehr ressourcensparend und die Basislibraries sind sehr groß. Mit der Distribution Ubuntu wird die grafische Oberfläche Gnome mitgeliefert. Bei der Installation wird allerdings ein Paket mitinstalliert, welches für den Download und die Darstellung von Online-Wetterdaten verantwortlich ist. Diese unumgehbare, zwingende Abhängigkeit bringt 20MB unnötige Festplattenbelegung mit sich.

DamnSmallLinux (DSL) *DSL* ist eine auf Debian basierende Distribution, deren Hauptparadigma eine möglichst geringe Größe bei größtmöglichem Funktionsumfang ist. Auf nur 50MB haben die Entwickler es geschafft, eine fast vollständige Desktopumgebung zu lie-

[7] http://www.compdigitec.com/labs/2009/03/17/windows-7-clean-install-size/
[8] http://www.tomshardware.com/de/windows-vista-xp-speicherplatz-verbrauch,testberichte-238565-22.html
[9] http://www.i64x.com/eeexp.php

fern[10]. Die Distribution enthält Webbrowser, E-Mail Client, Tabellenkalkulation, MP3-Player, Treiber für USB, PCMCIA und gängige WLAN Karten und noch einiges mehr. Das Betriebssystem ist außerdem derart ressourcenschonend, dass es vollständig in eine 128 Megabyte große Ramdisk passt. *DSL* kann nach einer Installation auf eine Festplatte sogar in eine vollständige Version von Debian Linux umgewandelt werden.

Aptitude Debian bringt den Paketmanager Aptitude mit, welcher bei der Installation eines Pakets dessen Abhängigkeiten (dependencies) betrachtet und notwendige Pakete zusätzlich zur Installation vorsieht. Allerdings gibt es für die Distributoren die Möglichkeit den Abhängigkeiten noch eine Wichtigkeit zuzuordnen. Es lassen sich so auch Pakete empfehlen, die aber nicht notwendig sind. Auf diese Weise installiert der Paketmanager oft viel zu viele Pakete. Dies lässt sich jedoch in den Programmoptionen abschalten, sodass nur noch Pakete installiert werden, die als wirklich notwendig erachtet werden.

4.3.3 SuSE / SuSE Studio

OpenSuSE-Studio ist ein über ein Webinterface steuerbares Maßschneiderungstoolkit. Mit ihm lassen sich, von einer Basisdistribution ausgehend, eigene Appliances erzeugen. Dabei umfasst das Maßschneidern sowohl die zu installierende Software als auch das Aussehen, auch *Branding* genannt. Weiterhin kann die Appliance als LiveCD, Installationsimage oder Festplattenimage heruntergeladen werden. Als weiteres Feature ist die erstellte Appliance auf dem Server testbar. Die Usability dieses Toolkits kann als Vorbild für den DoVinci-Publisher dienen.

4.4 Webserver

Die Verwendung eines Webservers in DoVinci, zur Verwaltung und Veröffentlichung von Appliances innerhalb der Infrastruktur, führt zu einer Evaluation vorhandener Web-Server Software.

Apache 2 ist eines der Vorzeigeprojekte der Open Source-Gemeinde, weltweit laufen 53,84% aller Webserver mit Apache[11]. Aus diesem Grund bietet sich die Verwendung von Apache als Webserver ohne weitere Überlegungen an. Allerdings hat Apache 2 auch Nachteile, weswegen im Folgenden mögliche Alternativen untersucht werden.

4.4.1 Apache 2

Der Webserver *Apache 2* ist ein mittlerweile sehr umfangreiches Projekt mit einer großen Zahl an Features und Erweiterungsmodulen. Die Anzahl der Kern-Funktionen alleine führt zu einem sehr großen Footprint der Applikation und macht das Arbeiten mit Apache mitunter unnötig komplex.

[10] `http://www.damnsmalllinux.org/index_de.html`
[11] `http://news.netcraft.com/archives/web_server_survey.html`

Apache 2 ist also für kleine Projekte mit einer geringen Anzahl benötigter Funktionen durchaus zu umfangreich. Weiterhin benötigt *Apache 2* Wartung und Pflege, wie jeder andere Webserver, allerdings gibt es Lösungen, die weitaus weniger Zeit für diese Aufgaben fordern und somit für kleine Projekte effizienter sind.

4.4.2 Lighttpd

Der Webserver *Lighttpd* ist Anfang 2003 im Rahmen einer Diplomarbeit entstanden[12]. Das grundsätzliche Problem, welches gelöst werden sollte, war, dass ein einzelner Server 10.000 parallele Verbindungen effizient handhaben sollte. Viele Systeme werden schon bei viel geringerer Last unbrauchbar, bei *Apache 2* reichen schon 100 parallele Verbindungen zu Überlastung.

Der *Lighttpd* wurde mit dieser grundsätzlichen Forderung im Hintergrund entwickelt und stellt mittlerweile eine Alternative gegenüber dem *Apache 2* dar. Der Code des Lighttpd und so auch sein Speicherabdruck sind sehr viel kleiner als der des Apache, und die gesamte Entwicklung wurde auf Effizienz ausgerichtet. Darüber hinaus gibt es weitere Features, die für DoVinci interessant sind. So gibt es ein spezielles Modul für *Lighttpd*, welches es ermöglicht große Dateien für den Down-/Upload in Blöcke von 500 Kilobyte zu teilen. Bei diesen Operationen wird dann jeweils nur der aktuell übertragene Block im Speicher gehalten. Dies verringert den Ressourcenverbrauch bei vielen parallelen Operationen ungemein und soll es ermöglichen die Anforderungen an den DoVinci-Server mit Standardhardware zu erfüllen.

4.4.3 Fazit

Der Hauptgrund für die Verwendung von *Lighttpd* ist die geringe Einarbeitungszeit. Apache-Konfigurationen sind komplexe Dateien, dasselbe gilt für die Wartung eines *Apache-2*-Servers. *Lighttpd* hingegen hat kleine unkomplizierte Konfigurationsdateien, effizienten Code, einen kleinen Speicherabdruck und benötigt nur geringen Wartungsaufwand.

4.5 Qt

Qt stellt mächtige und intuitive Klassenbibliotheken für C++ zur Verfügung. Der in Qt geschriebene Code lässt sich auf alle von Qt unterstützten Desktop- und Embedded-Betriebssysteme portieren. Somit lässt sich mit Qt plattformübergreifend implementieren. Es ist nicht nur mit C++, sondern auch mit anderen Programmiersprachen kombinierbar, so zum Beispiel mit Perl, Pyhton, OpenGL und SQL. Es hinterlässt zudem nur einen kleinen Speicherabdruck auf eingebetteten Systemen. Des Weiteren bietet die Firma Nokia eine sehr gute *Online Reference Documentation*[13]. Somit ist Qt geeignet im weiteren Verlauf der Entwicklung von DoVinci eingesetzt zu werden.

[12] `http://www.lighttpd.net/story`
[13] `http://doc.trolltech.com/`

5 Prototyp MS1

Der erste Milestone (MS1) besteht aus drei Komponenten. Der erste Grobentwurf für den ersten Prototypen sieht eine Aufteilung in einen Server und einen Client vor. Der Server stellt Appliances zum Download bereit. Das Client-Tool beinhaltet sowohl die Funktionen für Benutzer, die Appliances herunterladen und verwenden wollen, als auch die Funktionen die von Publishern verwendet werden, um Appliances auf den Server hochzuladen. Der Fokus bei diesem Milestone liegt darauf, mit vertretbarem Aufwand funktionsfähige Komponenten zur Demonstration des Gesamtsystems zu erstellen. Daher ist für die Client-Seite die x86-Architektur festgelegt, für die bereits eine Vielzahl von Virtualisierungslösungen existiert. Als Betriebssystem fällt die Wahl sowohl für das virtualisierte System als auch für den Client auf Linux. Als System innerhalb der Appliance ermöglicht es eine relativ einfache Maßschneiderung mit den bereits vorhandenen Paketmanagern der Distributionen, als System für den Client ermöglicht es die Nutzung von KVM, welches sehr einfach durch den Client gestartet und beendet werden kann. Der Client selbst ist unter Verwendung von Qt4 geschrieben, um eine spätere Portierung auf andere Betriebssysteme zu vereinfachen. Die Maßschneiderung im ersten Prototypen beschränkt sich auf manuelle Maßschneiderung unter Verwendung des Paketsystems der Linux-Distribution mit dem Ziel eine möglichst kleine Appliance zu erzeugen.

5.1 Client

Als *DoVinci-Client* wird die Applikation bezeichnet, welche einem Benutzer für die Verwendung und Verwaltung virtueller Appliances zur Verfügung gestellt wird. Der DoVinci-Client ist als Benutzerschnittstelle ein Hauptbestandteil der DoVinci-Infrastruktur. Das Programm bildet die Schnittstelle zwischen dem Endanwender und vorhandenen DoVinci-Servern. Abhängig von der Benutzerrolle des Anwenders lassen sich verschiedene Aktionen des Programms benutzen. Eine grafische Oberfläche erleichtert dem Benutzer den Umgang mit lokalen Appliances und entfernten Servern. Aus Sicht des Benutzers (hier insbesondere Appliance-Benutzer als auch Appliance-Anbieter) lässt sich der gesamte Ablauf über diese Applikation steuern. Der erste Prototyp des Clients soll die rudimentäre Funktionalität und eine erste Benutzeroberfläche bereitstellen. Durch die Verwendung etablierter plattformunabhängiger Bibliotheken wird versucht die Plattformunabhängigkeit und leichte spätere Erweiterung der Funktionalität zu gewährleisten.

5.1.1 Anforderungen

Der Client stellt eine Schnittstelle zur Verfügung, mit deren Hilfe der Benutzer die vom Server angebotenen Appliances herunterladen und verwalten kann. Nimmt der User die Publisher-Rolle ein, ist ein Upload auch möglich. Da das Client-Tool die User- und die Publisherfunktionen enthält, werden die Anforderungen an den Client 3.3.1 und die Anforderungen an das Publisher-Tool 3.3.2 beide an das Client-Tool gestellt.

Dienstfindung Die Dienstfindung soll im ersten Milestone noch über direkte Eingabe einer Adresse laufen. Es genügt, wenn eine Liste mit den Metadaten der vom Server zum Download angebotenen Appliances heruntergeladen wird.

Verwaltung der Appliances Vom Server angebotene Appliances sollen heruntergeladen lokal verwaltet, gestartet und gestoppt werden können. Darüber hinaus soll der Upload von lokal veränderten Appliances implementiert werden.

Plattformunabhängigkeit Der Client sollte auf möglichst vielen Endgeräten unabhängig von den dort installierten Betriebssystemen laufen.

5.1.2 Entwurf

Um dem Anspruch der Plattformunabhängigkeit des Clients gerecht zu werden, wurde die Entscheidung getroffen, den Client unter Verwendung von Qt zu implementieren. Runtergeladene Appliances sollen im MS1 zunächst in einer lokal vorhandenen Datei verwaltet werden. Ein Übergang zu der Verwaltung durch eine Datenbank ist in nachfolgenden Schritten vorgesehen, im Rahmen der Implementierung des MS1 jedoch nicht. Die Dienstfindung soll im MS1 über eine feste IP erfolgen. Über diese Adresse soll eine Liste angebotener VAs runtergeladen werden können. Die Liste soll die Metadaten der zum Download angebotenen Appliances verwalten.

5.1.3 Durchführung

Plattformunabhängigkeit Zunächst wurde nur die GUI mit Qt implementiert. Das Networking wurde in einem ersten Versuch mit den Bibliotheken *Boost* und der C++-Standardbibliothek (STL) implementiert. Jedoch war der Code nur unter Linux kompilierbar. Im Laufe der Implementierung wurde die Entscheidung getroffen, das Networking mit Qt neu zu programmieren, um die Systemunabhängigkeit des Clients zu erreichen. Bei der Umsetzung des Networkings mit WebDav und SSL-Verschlüsselung traten Probleme durch einen Fehler in der verwendeten Version von Qt auf, die jedoch mit einem Workaround umgangen werden konnten.

Im Laufe der Implementierung des ersten Milestones wurde die Entscheidung getroffen, die Anforderung an die Plattformunabhängigkeit des Clients aufzugeben und den Fokus auf Linux-Distributionen zu setzen.

GUI Die graphische Benutzeroberfläche des Clients (siehe Abbildung 7) präsentiert dem Benutzer primär zwei Auswahllisten. Die linke davon ist eine Liste der auf dem Server verfügbaren Appliances und die rechte zeigt die lokal installierten Appliances. Zur Auswahl des Servers, dessen Appliances angezeigt werden sollen, verwendet dieser erste Prototyp lediglich ein einfaches Eingabefeld für eine URL einer Appliance-Liste, die durch Druck auf den Button „Serverliste Aktualisieren“ heruntergeladen und dargestellt wird. Buttons zwischen den beiden Listen ermöglichen das Hoch- und Herunterladen von VAs vom Server

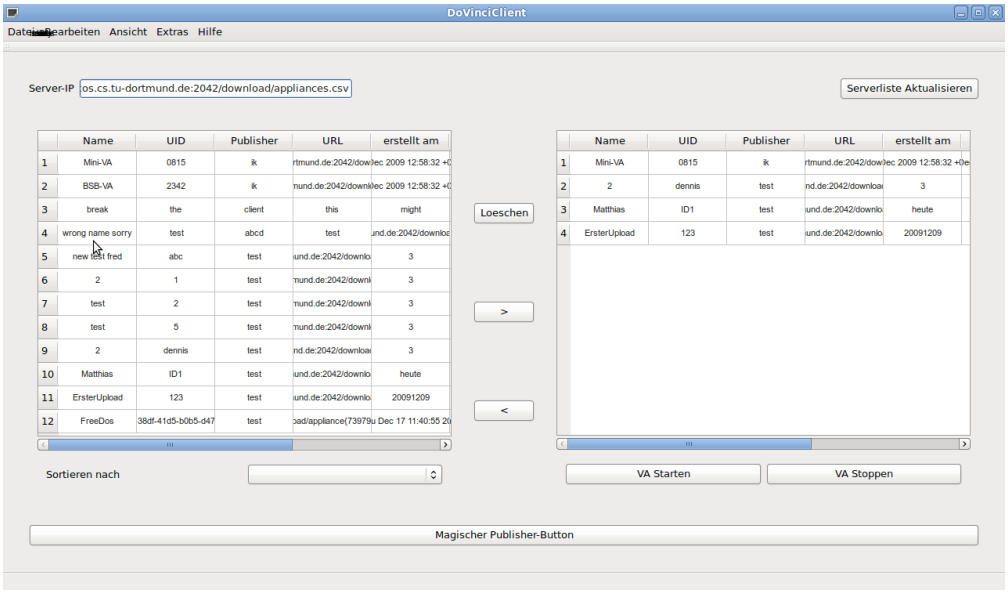

Abbildung 7: GUI des Client-Tools

sowie das Löschen lokal vorhandener VAs. Zwei weitere Buttons unterhalb der Liste der lokal vorliegenden Appliances dienen zum Starten und Stoppen selbiger. Um rudimentäre Publisher-Features bieten zu können gibt es zudem den vorläufig so genannten „Magischer Publisher-Button“, welcher in einem getrennten Dialog die minimal notwendigen Daten für eine Appliance (Name, Beschreibung, Ersteller, Dateiname und Name des Servers) abfragt und anhand dieser Daten eine VA in die lokale Liste einträgt.

Download Die Dienstfindung erfolgt im MS1 über eine feste IP, unter der eine Liste angebotener VAs runtergeladen werden kann. In der Liste befinden sich Pfade, unter denen die einzelnen vom Server angebotenen VAs heruntergeladen werden können. Hierbei ist die Aktualität der Dienstfindung nicht berücksichtigt, diese wird nur manuell durch Userinteraktion erreicht (Button „Aktualisieren“). Der Client lädt dann mit einem HTTP-GET eine vom Server zur Verfügung gestellte CSV-Datei runter. Die Metadaten der VAs werden in der GUI in einer Tabelle dargestellt und können dort angeklickt werden, um die gewünschte Appliance per WebDAV-GET runterzuladen. Der Client fügt die Appliance in die lokale Tabelle ein und speichert den Pfad, wo die Appliance abgelegt wurde.

Upload Der Upload der Appliance wird mit Hilfe eines Workarounds realisiert. Da serverseitig der Upload einer VA nur mit Angabe von Benutzername und Passwort möglich sein soll, müssen diese beim Upload an den Server übermittelt werden. Die für diesen Zweck gewählte Methode ist die übliche HTTP-Authentifizierung, die auch von Qt unterstützt wird. Jedoch hat die verwendete Qt-Version in diesem Zusammenhang einen Fehler, durch den der Upload stets ohne Übermittlung der Benutzerdaten erfolgt und die dann eigentlich vorge-

sehene Signalisierung an die Applikation, dass Benutzerdaten übergeben werden müssen, nicht stattfindet. Das derzeit implementierte Workaround besteht darin, dass das Feld für die Benutzerdaten, in der zum Server geschickten Anfrage vom Client, manuell gesetzt wird und so bereits die erste Kommunikation mit dem Server erfolgreich ist. Der Nachteil dieses Workarounds liegt jedoch darin, dass diese Daten nun immer mit gesendet werden, auch wenn der Server eigentlich keine Benutzeridentifikation verlangt. Dies wird als vertretbares Risiko angesehen, da die Kommunikation mittels SSL gesichert ist und die Wahrscheinlichkeit eines unbefugten Mitschneidens der Benutzerkennung bei Einsatz dieses Workarounds sehr gering ist.

Plattformunabhängigkeit Während der Implementierungsphase wurde die Entscheidung getroffen, dass der Client im MS1 nur auf Linux laufen soll. Das Starten und Stoppen von Appliances geschieht mittels Kommandozeilenaufruf von KVM, welches nur unter Linux läuft. Somit ist die Anforderung für die Plattformunabhängigkeit verletzt worden, die Anforderung an die Verwaltung der Appliances aber vollständig erfüllt.

5.1.4 Evaluation

Der Client verhält sich den Anforderungen entsprechend. Der Download der Liste der vom Server angebotenen Appliances sowie der Download und Upload selbiger funktioniert einwandfrei. Der Client läuft plattformunabhängig, jedoch ist das Starten und Stoppen der Appliances nur unter Linux möglich, da KVM nur unter Linux läuft.

Fazit Der Client verfügt über die Grundfunktionalitäten, um die angebotenen Appliances herunterzuladen, sie zu verwalten und sie hochzuladen. Die Bedienung ist intuitiv und einfach aufgebaut. Bei der Implementierung des Clients wurde insbesondere auf Modularität und Erweiterbarkeit geachtet. Er stellt somit ein gutes Grundgerüst für den MS2 dar, insbesondere im Hinblick auf Erweiterbarkeit. Der Client stellt eine solide Basis für das Provisioning von Virtual Appliances dar.

5.2 Server

Der Server hat die Aufgabe die *Virtual Appliances* zu verwalten, zu organisieren und sie der Client-Applikation des Endbenutzers zur Verfügung zu stellen. Er ist essentieller Bestandteil der DoVinci-Infrastruktur. Der erste Prototyp dieses Servers implementiert dessen grundlegende Funktionen. Er erlaubt durch seine modulare Entwicklung die Erweiterung um weitere Funktionen späterer Stadien, um auch die Anforderungen jenseits des MS1 erfüllen zu können. Der Server verfügt außerdem über Sicherheitsfunktionen, die ein Kompromittieren der Infrastruktur durch Dritte verhindern sollen, ohne dabei den Komfort der legitimen Endbenutzer einzuschränken. Der Server nimmt die zentrale Rolle zwischen Client und Publisher ein, da er die gesamte Auslieferung und Verwaltung von Virtual Appliances steuert. Dabei bietet er weitergehende Services, wie Nutzerverwaltung sowohl auf Client- als auch auf Publisherseite, an und garantiert die nötige Datensicherheit sowie den Datenschutz.

Basierend auf noch zu treffenden Designentscheidungen kann es sein, dass der Server um Maßschneiderung und/oder Sharing für die einzelnen Virtual Appliances erweitert wird.

Im Folgenden wird beschrieben, welche Anforderungen in diesem Prototyp erfüllt sein sollen und wie diese realisiert werden.

5.2.1 Anforderungen

Konfigurationsfreie Dienstfindung Die Liste der Virtual Appliances wird dem Client vom Server aus über WLAN bereitgestellt. Die vorhergehende Dienstfindung soll möglichst einfach, später sogar konfigurationsfrei erfolgen.

Upload Es wird die Möglichkeit des Uploads neuer und der Aktualisierung vorhandener Appliances durch einen Publisher zur Verfügung gestellt.

Maßschneiderung und/oder Sharing Maßschneiderung und/oder Sharing sind noch kein Bestandteil dieses Prototyps.

VA-Verwaltung Die Informationen zu den Virtual Appliances werden vom Server verwaltet. Jede virtuelle Appliance sollte anhand einer eindeutigen Kennung identifizierbar sein, einen menschenlesbaren Namen tragen und mit einem Uploaddatum versehen sein. Der Zeitpunkt des letzten Updates, der Name des Autors, die Beschreibung der Appliance sowie eine Prüfsumme zur Überprüfung des Downloads sollten darüber hinaus verwaltet werden.

Datenintegrität Der Upload neuer Appliances wird ausschließlich autorisierten Publishern angeboten. Die Zugangsdaten werden vom Serverbetreiber vergeben.

Datensicherheit Die Datenübertragung zwischen Client und Server wird über eine sichere, verschlüsselte Verbindung abgewickelt.

Nutzerdaten-Verwaltung Mit Ausnahme der Publisher-Zugangsdaten gibt es keine weiteren nutzerbezogenen Daten im Rahmen dieses Prototypen.

5.2.2 Entwurf

Der Grundgedanke beim Entwurf ist, dass dem Prototyp schnellstmöglich grundlegende Funktionen zur Verfügung stehen sollen, damit er vom parallel entwickelten Client in Tests verwendet werden kann.

Grundlegende Überlegungen Zu diesem Zweck werden, wo immer möglich, schon vorhandene Standards verwendet. Ein Webserver eignet sich hervorragend für die Verwaltung und Bereitstellung der benötigten Dateien. Ein Datei-Server könnte auch verwendet werden, allerdings wurde bewusst die Nutzung von FTP ausgeschlossen, da dieses Protokoll aus historischen Gründen eine deutlich höhere Komplexität als HTTP aufweist. Für den Upload der Dateien kommt die HTTP-Erweiterung WebDAV zum Einsatz, da eine Übertragung mittels normalem HTTP-Post bei den üblichen Implementierungen in Webservern erst die komplette Datei im RAM des Servers zwischenspeichert und somit zu viele Ressourcen erfordert. Weiterhin wird Zeroconf (*Apple Bonjour*, *Avahi*) zur Realisierung der Dienstfindungs-Anforderungen genutzt. Die Verwendung von Zeroconf ist weniger kompliziert als die Alternative des Simple Service Discovery Protokolls aus UPnP, was sich mit den Ergebnissen der Seminarphase deckt. Dies führt dazu, dass auch ein DNS-Server in dem Netzwerk vorhanden sein muss. Wobei noch erwähnt sei, dass Zeroconf auch ohne Vollzugriff auf einen statischen DNS-Server zur Dienstfindung vollständig nutzbar ist.

Überlegungen bezüglich des Webservers Die Webserver-Software für den Server muss eine Reihe von Anforderungen erfüllen und bestimmte Eigenschaften aufweisen. Es soll die Möglichkeit geben vorhandene, getestete Module zu benutzen, insbesondere die zentrale Funktionalität eines Webservers sollte durch schon vorhandene und getesteten Module realisierbar sein. Auch eine Implementierung optimierter Datenzugriffsverfahren für den Up- und Download sollte bereits existieren. Weiterhin sollen Standards wie Secure Socket Layer (SSL) und die üblichen Authentifizierungsverfahren unterstützt werden, die auf den verbreiteten Web-Servern generell zur Authentifizierung verwendet werden. Alle weiteren Anforderungen und Funktionen, die nicht schon verfügbar sind, werden über die CGI-Schnittstelle erfüllt, indem entsprechende CGI-Skripte erstellt werden.

Überlegungen bezüglich des DNS-Servers Hier wird der de-facto-Standard für Linux, der DNS-Server *Bind9* (`http://www.bind9.net`) verwendet. Dieser verwaltet nur die notwendigsten Einträge ohne Erweiterungen wie DNS Dynamic Update und DNS Security Extensions zu nutzen, deren Features zwar durchaus nützlich sind, aber im Rahmen dieses ersten Prototypen unnötigen zusätzlichen Arbeitsaufwand bedeuten würden.

Fazit Dieser Entwurf soll schnell und effizient einen den Anforderungen entsprechenden Server zur Verfügung zu stellen und vor allem schnellstmöglich eine lauffähige Testumgebung für den Client zur Verfügung stellen.

5.2.3 Durchführung

Aufbauend auf unserem Entwurf wird nun die Umsetzung basierend auf der entsprechenden Hard- und Software durchgeführt. Dabei wird folgende Infrastruktur als Basis gewählt:

- Zentraler Server (stationärer Rechner)
  - Betriebssystem: Debian

Abbildung 8: Serverentwurf-Technologien-Schaubild

  - Webserver: Lighttpd mit den Modulen:
    - CGI
    - Htaccess
    - WebDAV
    - OpenSSL
  - Programmiersprache: Perl
  - Nameserver: BIND
- WLAN-Accesspoint

Folglich teilt sich die Durchführung in drei Aufgabenbereiche:

Initiale Einrichtung und Installation auf der Hardware Um die nötigen Voraussetzungen für den Betrieb des Servers zu schaffen, wird ein Desktop-Rechner als zentraler Server verwendet, welcher über eine direkte Anbindung zum eigenen WLAN-Accesspoint verfügt. Der Client und der Publisher können somit ausschließlich via WLAN auf den zentralen Server zugreifen.

Die nötige Software wird, wie oben beschrieben, installiert. Bei der Wahl der Software wird ausschließlich auf Open-Source-Software mit flexiblen Lizenzbedingungen gesetzt. Neben der Kostenersparnis ist hier unter anderem die Möglichkeit der freien Anpassung des Sourcecodes ein wichtiger Aspekt für unser Vorhaben.

Eine besondere Rolle spielen die zusätzlich installierten Module für den Lighttpd-Webserver. Folgende Module werden installiert:

- CGI: Dieses Modul dient als Schnittstelle zwischen der Programmiersprache Perl und dem Webserver.

- htaccess: Die Authentifizierung der Publisher erfolgt via htaccess.
- WebDAV: Als Erweiterung des HTTP-Protokolls um HTTP-PUT nutzen wir WebDAV für den Upload der teilweise sehr großen Image-Dateien der Appliances.
- OpenSSL: Um eine sichere Kommunikation zwischen Client/Publisher und Server gewährleisten zu können, setzen wir bei allen Anfragen und Ausgaben auf HTTPS.

Installation und Entwicklung für die Client-Server-Kommunikation Der Client soll auf einfache Art und Weise die aktuell auf dem zentralen Server verfügbaren Appliances abfragen und diese auf Wunsch herunterladen können. Dazu stellen wir eine CSV-Datei über den Lighttpd-Webserver zur Verfügung, welche ohne weitere Authentifizierung öffentlich verfügbar ist. Diese CSV-Datei enthält Datensätze zu den derzeit verfügbaren Appliances, einschließlich des Downloads-Pfads der jeweils zugehörigen Image-Datei.

Installation und Entwicklung für die Publisher-Server-Kommunikation Die Kommunikation zwischen Publisher und Server beschränkt sich auf die Verwaltung von Virtual Appliances, welche durch den Publisher über den Server zur Verfügung gestellt werden sollen. Dabei muss das Hinzufügen, Löschen und Aktualisieren von Appliances umgesetzt werden.

Voraussetzung für die Kommunikation des Publishers mit dem Server ist eine erfolgreiche Authentifizierung. Nur registrierte Publisher können ihre Appliances am Server verwalten. Die Authentifizierung ist über das htaccess-Modul des Lighttpd-Webservers realisiert und schützt alle Scripte, welche für das Hinzufügen, Aktualisieren oder Löschen von Appliances durch den Publisher verwendet werden können.

Im Folgenden gehen wir detaillierter auf die Kommunikation zwischen beiden Parteien ein. Der Ablauf für das Hinzufügen und Aktualisieren sieht wie folgt aus:

- Schritt 1: Upload der Image-Datei der Appliance via HTTP PUT (WebDAV) als Binärdatei
- Schritt 2: Aufruf des Skripts upload.cgi, welches Metainformationen von zuvor übertragenen Appliances speichert

Der erste Schritt wird dabei über das WebDAV-Modul des Lighttpd-Webservers realisiert. Dabei muss der Name der per HTTP-PUT übertragenen Image-Datei stets das generische Format „appliance⟨ID⟩.image“ einhalten, damit die Image-Datei bei Aufruf des im zweiten Schritt nötigen upload.cgi-Skripts den Metainformationen zugeordnet werden kann.

Für den zweiten Schritt wird das Perl-Skript upload.cgi umgesetzt, welches mit einer Reihe von Parametern aufgerufen werden kann. Diese werden per HTTP-POST-Request übermittelt.

Folgende Parameter müssen dabei an das Script bei Aufruf durch den Publisher übergeben werden:

- ID: eindeutige Kennung der neuen oder aber bereits existierenden Appliance
- NAME: beschreibender Name der Appliance

- CREATION_DATE: Datum der erstmaligen Erstellung der Appliance
- LAST_UPDATE_DATE: Datum der letzten Aktualisierung der Appliance
- MD5_CHECKSUM_FOR_IMAGE: MD5-Hash der zuvor übermittelten Image-Datei zur Prüfung, ob diese fehlerfrei übertragen wurde
- AUTHOR: Autor der Appliance
- DESCRIPTION: Beschreibung der Appliance

Das Script kann folgende Rückgabewerte ausgeben:

- OK: Aktion erfolgreich
- ERROR_⟨PARAMETER⟩ _MISSING: Der Parameter ⟨PARAMETER⟩ wurde nicht an das Script übergeben
- ERROR_UPLOAD_RAW_IMAGE_VIA_WEBDAV_FIRST: Es wurde zuvor keine Image-Datei übertragen oder diese ist falsch benannt
- ERROR_RAW_IMAGE_DOES_NOT_MATCH_MD5_CHECKSUM: Der Upload einer Image-Datei via HTTP PUT war fehlerhaft
- ERROR_CANNOT_ACCESS_CSV_FILE: Die Informationen der Appliance konnten am Server nicht gespeichert werden

Bei Rückgabewert OK ist die entsprechende Aktion des Publishers erfolgreich gewesen.

Ähnlich sieht die Aktualisierung einer bereits am Server existierenden Appliance aus. Wird eine Appliance mit einer bereits existierenden eindeutigen ID über den oben beschriebenen Weg hinzugefügt, so wird das upload.cgi-Skript diese Appliance aktualisieren. Dies funktioniert auch dann, wenn zuvor keine Image-Datei via HTTP-PUT gesendet wurde. Über diese Logik lassen sich Meta-Informationen zu der Appliance anpassen, ohne dass die Image-Datei erneut gesendet werden muss.

Das Löschen einer Appliance kann derzeit nur manuell durch einen Administrator am Server durch Entfernen der entsprechenden Image-Datei sowie dem Eintrag in der zentralen CSV-Datei durchgeführt werden.

Zeroconf, Linux und Bind9 Im Rahmen der Implementation des ersten Prototypen sind Probleme mit der Client-Implementation von Zeroconf und der Zusammenarbeit mit dem *Bind9*-Server aufgetaucht. Unter Linux, das für den ersten Prototyp als Referenzsystem gilt, gibt es seit einigen Jahren eine GPL-Implementation von Zeroconf mit dem Namen *Avahi*. Es gibt für *Avahi* auch Bibliotheken die, falls benötigt, die API-Funktionen von *Bonjour* auf Avahi abbilden, was die plattformunabhängige Implementierung von Zeroconf in Applikationen ermöglicht, indem unter Linux *Avahi*, unter Windows *Bonjour for Windows* und unter Mac OS X *Bonjour* zum Einsatz kommt. Allerdings arbeitet *Avahi* unter Linux nicht ohne Probleme mit einem schon vorhandenen DNS-Server zusammen, wenn dieser ebenfalls

Dienstfindung anbietet. Des Weiteren sind die API-Funktionen von *Bonjour* nicht vollständig in der Kompatibilitätsbibliothek abgebildet bzw. funktionstüchtig. Als Kompromiss wird deshalb ein Modul für den Web-Server entwickelt. Dieses Modul hat die Funktion den Dienst über Multicast-DNS und Zeroconf zu registrieren und anzubieten. Dies löst das Problem noch nicht komplett, erlaubt aber dem Prototypen zumindest die Anforderungen im Rahmen der Dienstfindung zwischen Client und Server dennoch zu erfüllen. Aufgrund dessen sind aber auch die Voraussetzungen an eine DoVinci-Netzwerk-Infrastruktur geringer ausgefallen, ein DNS-Server ist nun keine Voraussetzung mehr, was zum Beispiel eine Verwendung von DoVinci in Ad-Hoc Netzen möglich macht.

5.2.4 Evaluation

Der Server verhält sich den Anforderungen entsprechend. Der Datendurchsatz in einem Funknetzwerk entspricht den Erwartungen und den berechneten Werten, folglich existieren auch keine unerwarteten Probleme in der grundlegenden Architektur und Infrastruktur.

WLAN-Durchsatzmessung Um die Dauer der Auslieferung vorhandener Appliances in Zukunft abschätzen zu können, wurde auf zwei Wegen die Übertragungsdauer ermittelt. Zunächst wurde die Dauer anhand theoretischer Daten berechnet und im Anschluss in einem Versuch überprüft.

Grundlage der Berechnung bildet das vorhandene 54MBit/s Wireless-Netzwerk. Somit ist eine Bruttodatenrate von 54MBit/s möglich. Aufgrund der verwendeten WLAN-Protokolle folgt, dass pro Kanal zu jedem Zeitpunkt nur ein Teilnehmer senden kann. Diese Halb-Duplex-Eigenschaft beschränkt die Bruttodatenrate auf 27MBit/s[Lun07]. Die Größe der Test-Appliance beträgt 292.974.592 Byte. Die Anzahl der zu übertragenen Byte berechnet sich wie folgt:

Payload eines WLan-Pakets	1400	Byte
TCP/IP Header	40	Byte
SNAP/LLC/MAC Header	42	Byte
WPA Header		vernachlässigt

Somit ist die Anzahl der zu übertragenen Byte 309429330. Es ergibt sich eine Übertragungdauer bei 27MBit/s von 87 Sekunden. Dieser Wert lässt sich im Feldversuch über zehn Messungen in etwa bestätigen:

Messung	Zeit in Sek.
1	90,155
2	88,746
3	92,512
4	89,642
5	91,269
6	94,166
7	90,147
8	97,244
9	92,160
10	91,238
Ø	91,7279

5.3 Demo Appliance

Die Demo-Appliance für den ersten Prototyp MS1 ist eine vorlesungsbegleitende Appliance für die Veranstaltung Betriebssystembau (BSB). In dieser Veranstaltung wird von den Hörern ein Betriebssystem namens *OOStuBS* entwickelt. Die Entwicklung findet in *C++* statt, aber es sind auch Teile in *C* und *Assembler* zu schreiben. Hilfestellung wird in Form einer Webseite angeboten. Getestet und ausgeführt wird *OOStuBS* in einem Emulator und in letzter Konsequenz auf echter Hardware.

5.3.1 Anforderungen

Das Lastenheft für die erste Demo-Appliance sieht vor, dass ein Linux-Betriebssystem manuell maßgeschneidert wird, um Studenten die OOStuBS-Entwicklung zu ermöglichen. Als Virtualisierungslösung kommt KVM zum Einsatz. Das primäre Ziel soll eine möglichst kleine Appliance sein, die sich dennoch wie gewohnt verhält.
Es müssen zwingend in der Appliance enthalten sein:

- Grafische Oberfläche: Zur gewohnten Bedienung mit Maus und Tastatur
- Browser: für die Hilfestellung bei der *OOStuBS*-Entwicklung
- Editor mit Syntaxhighlighting: Zum Bearbeiten von Sourcen
- OOStuBS-Vorgaben: Zip-Dateien
- g++: Zum kompilieren der *C++*-Sourcen
- gcc: Zum kompilieren der *C*-Sourcen
- nasm: Zum kompilieren der *Assembler*-Sourcen
- bochs: x86-Emulator zum Test

5.3.2 Entwurf

Da für die Demo-Appliance des ersten Prototypen MS1 die Virtualisierungslösung KVM eingesetzt wird, fällt die Wahl des Betriebssystems auf ein aktuelles Debian-Linux mit Kernel der Version 2.6. Ab dieser Kernelversion sind paravirtualisierte Treiber enthalten, welche bei Einsatz in einer auf KVM basierenden virtuellen Maschine höhere Performanz versprechen. Die Debian-Installation installiert zunächst ein möglichst kleines lauffähiges Basissystem, welches dann per Hand weiter abgelastet und schließlich mit der erforderlichen Software ausgestattet wird. Die Wahl der grafischen Oberfläche fällt auf KDE, da diese einfach anzuwenden ist, insbesondere für Benutzer, die der Windows-Welt entstammen. Als Browser kommt Konqueror zum Einsatz, da dieser als KDE-Bestandteil bereits im System vorliegt und so kein zusätzlicher Speicherplatz für einen weiteren Browser erforderlich ist. KDE bringt einen integrierten Editor mit Namen Kate mit, welcher Syntaxhighlighting beherrscht und somit den Anforderungen gerecht wird. Die darüber hinaus geforderten *OO-StuBS*-Dateien werden in das Benutzerverzeichnis */home/dovinci* gespeichert und die Übersetzer per Paketmanagement installiert. Das so entstehende Image ist die Demo-Appliance.

5.3.3 Durchführung

In diesem Abschnitt wird nun erklärt, wie die erste Demo-Appliance erzeugt wird. Zunächst wird die virtuelle Festplatte vorbereitet, hierzu wird das Programm *kvm-img* verwendet. Eingestellt mit Parametern erzeugt es zum Beispiel eine Image-Datei im Format *qcow2* mit einer maximalen Größe von 2GB.

```
1 kvm-img create myImage.qcow2 -f qcow2 2G
```

Nun wird eine virtuelle Maschine definiert und gestartet. Hierzu muss die virtualisierte Hardware genau angegeben werden. Das erzeugte Image wird der Maschine als erste Festplatte vorgegeben. Ein Debian-Linux Installations-Image wird in der Maschine als CD-Laufwerk virtualisiert. Zur Erkennung einer Tastatur mit deutschem Tastaturlayout wird die entsprechende Option eingegeben. Die Größe des Arbeitsspeichers wird mit 512MB angegeben. Als virtuelle Grafikkarte soll ein Cirrus Logic-Adapter verwendet werden. Eine Netzwerkkarte wird ebenfalls virtualisiert und logisch mit der Hostmaschine verbunden. Die Option *virtio* erzwingt die Verwendung von Paravirtualisierung zur Leistungssteigerung bei Hardwarezugriffen.

```
1 kvm    -drive file=myImage.qcow2,if=virtio,boot=on
2        -cdrom linuxInstall.iso
3        -k de
4        -m 512
5        -vga cirrus
6        -net nic,model=virtio -net user
```

In der gestarteten Maschine wird nun eine minimale Installation der Debian-Linux Distribution installiert. Diese enthält bereits den Paketmanager *Aptitude*, welcher nun verwendet wird, um die „minimale“ Installation manuell von enthaltenem Ballast zu befreien. Das Lastenheft sieht zum Beispiel keine Kommunikation über Bluetooth vor, weswegen die Trei-

berunterstützung und Dokumentation hierzu entfernt wird. Zusätzlich werden jene im Lastenheft beschriebenen Pakete installiert, welche für die geforderte Funktionalität vonnöten sind. Schließlich wird in der virtuellen Maschine ein Kommando benötigt, welches sämtliche freien Blöcke auf der virtuellen Festplatte mit Nullen überschreibt, damit das Image außerhalb der virtuellen Maschine weiter komprimiert werden kann. Der Befehl *dd* kann mit entsprechenden Parametern solange Nullen in eine Datei schreiben bis die Festplatte voll ist. Anschließend wird diese Datei wieder gelöscht und so bewirkt, dass die unbenutzten Bereiche der Festplatte von nicht mehr benötigten Daten befreit sind.

```
1 dd if=/dev/zero of=zeroFile bs=1024 ; sync ; rm zeroFile
```

Anschließend wird die virtuelle Maschine heruntergefahren und in der Hostmaschine das Image komprimiert. Die Option *-c* gibt an, dass beim Konvertieren komprimiert werden soll, die Option *-O* ermöglicht es das Ausgabeformat anzugeben.

```
1 kvm-img convert -c myImage.qcow2 -O qcow2 myImage_compressed.qcow2
```

Das resultierende Image hat nun eine weitaus kleinere Dateigröße als die 2GB, die es maximal erreichen darf.

5.3.4 Evaluation

Image-Größen Die Übertragungszeiten von Appliances bei Transfer über Wireless-LAN hängen von der Größe der zu übertragenden Images ab. Es ist daher wichtig verschiedene Betriebssysteme, einschließlich Distributionen, auf ihre resultierende Image-Größe hin zu vergleichen, um schließlich kürzere Übertragungszeiten gewährleisten zu können.

Ein erster Versuch, in dem ein Debian-Linux manuell für die Testanwendung maßgeschneidert wurde (debian-bsb), führte zu einem Image mit *280MB* Dateigröße. In diesem Prototypen sind die grafische Oberfläche *KDE*, der Editor *Kate*, der Browser *Konqueror* und alle für OOStuBS relevanten Übersetzer und Daten integriert und einsatzfähig.

Die minimale Installation eines Debian-Linux führt zu einem komprimierten Image von *190MB* Größe. Dieses lässt sich manuell maßschneidern auf etwa *104MB*. Eine minimale Installation des auf Debian basierenden Ubuntu der Version hardy mit Kernel der Version 2.4 erzeugt nach manueller Maßschneiderung ein Image mit *115MB* Platzbedarf. Die Version karmic mit Kernel 2.6 erfordert jedoch schon *240MB*. So wird deutlich, dass die Wahl des zugrunde liegenden Betriebssystems für die virtuelle Appliance erheblichen Einfluss auf die resultierende Appliancegröße hat.

Es existieren verschiedene extrem ressourcenschonende Linuxdistributionen. Beispielsweise erfordert DamnSmallLinux (DSL) nur *61MB*, arbeitet hierzu allerdings mit dem weniger performanten Kernel 2.4, ist nicht flexibel erweiterbar und stellt eine ungewohnte grafische Oberfläche bereit. Eine Variante von DSL namens DamnSmallLinuxNot (DSLN) arbeitet mit der performanteren Kernelversion 2.6, ist jedoch ebenso unflexibel und erfordert etwa *100MB* Festplattenspeicher.

5.4 Ermittlung tatsächlich verwendeter Dateien

Um die Effektivität der manuellen Maßschneiderung zu überprüfen, wird ein Linux-Kernel so modifiziert, dass er bei jedem Öffnen einer Datei und bei jeder Programmausführung die betroffene Datei in seiner Ausgabe meldet. Lösungen auf Basis von Userspace-Programmen, die beispielsweise mittels inotify Dateizugriffe protokollieren können, verpassen viele Dateien, die beim Booten vor dem Start des Protokolls und beim Herunterfahren nach der Umstellung des Festplattenzugriffs auf „nur lesend“ durchgeführt werden. Im Gegensatz dazu erlaubt es diese Lösung, jeden Dateizugriff vom Anfang des Bootvorgangs bis zum Abschalten der virtuellen Maschine mitzuschneiden. Weiterhin können die Kernelausgaben mit dem Kernelparameter *console=ttyS0,115200* direkt auf die virtuelle serielle Schnittstelle geschickt werden, so dass die Ausgaben von KVM direkt in eine Datei des Hostsystems geschrieben werden und so zusätzliche Schreibzugriffe für das Protokoll in der Appliance ausbleiben, die das Ergebnis möglicherweise verfälschen können.

Der modifizierte Linux-Kernel wird zwecks einfacherer Handhabung statisch, d.h. ohne Verwendung von Modulen kompiliert, zur Einbindung in die Appliance reicht daher das Hineinkopieren einer einzigen Datei und eine Änderung der Bootloader-Einstellungen, damit der modifizierte Kernel zum Starten des Systems verwendet wird. Mit diesem Kernel wird ein Testlauf durchgeführt, der aus einer angenommenen „typischen“ Benutzung der Appliance besteht. Im Fall der BSB-Appliance ist dies beispielsweise der Abruf der Vorlesungswebseite, Download des zu bearbeitenden Codes, Bearbeitung und Kompilierung desselben und ein Test des Kompilats. Die bei diesem Testlauf festgestellten Dateizugriffe werden mit Hilfe eines Perl-Scripts in eine Liste aller „angefassten“ Dateien konvertiert. Diese Liste wird anschließend sowohl gegen die Liste aller in der Appliance vorhandenen Dateien als auch gegen die Datenbank der Paketverwaltung in der Appliance gehalten und die Differenzen ausgewertet.

5.4.1 Beispielevaluation der BSB-Appliance

Selbstverständlich ist es sinnlos, lediglich die „angefassten“ Dateien zu betrachten, ohne Informationen über die vorhandene Gesamtmenge zu haben. Im Fall der BSB-Appliance besteht diese Gesamtanzahl aus 29591 Dateien, welche unkomprimiert 527MB belegen. Eine Auswertung der Dateien nach ihrer Größe zeigt, dass es sich dabei vorwiegend um kleine Dateien handelt: Mehr als 95% dieser Dateien haben eine Größe von unter 45KB, ein Detail, welches beim späteren Sharing der Dateien verschiedener Appliances nicht ausser Acht gelassen werden sollte.

Ignoriert man gemeldete Dateien und Verzeichnisse, die nicht tatsächlich auf der virtuellen Festplatte liegen, also beispielsweise Einträge im dynamisch erzeugten */dev*-Verzeichnis und die virtuellen Dateisysteme */proc* und */sys*, tritt bemerkenswertes zum Vorschein. Die Menge der tatsächlich „angefassten“ Dateien auf der virtuellen Festplatte beläuft sich lediglich auf 1495 Dateien mit einer Gesamtgröße von knapp 106MB, was einem Anteil von nur etwa 20% entspricht.

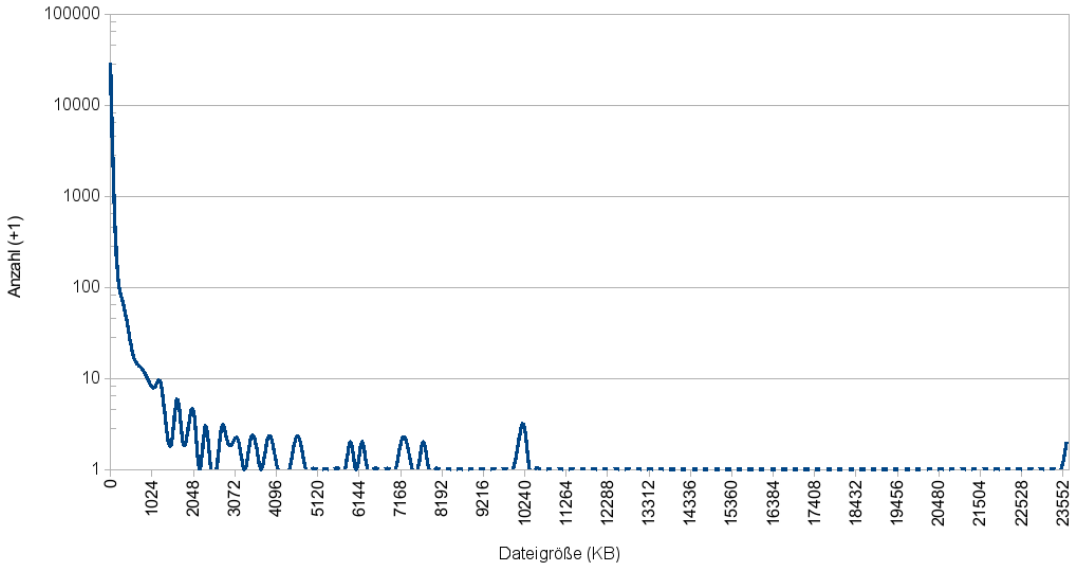

Abbildung 9: Größenverteilung der Dateien der BSB-VA

5.4.2 Referenzlauf

Natürlich kann man die Appliance nicht einfach auf die im Testlauf beobachteten Dateien reduzieren, denn es ist anzunehmen, dass im Testlauf nicht alle in der späteren Verwendung notwendigen Dateien auch verwendet werden. Ein Beispiel hierfür sind die Symbole in den Menüs von KDE-Applikationen wie beispielsweise Kate: Die Symbole liegen im System als einzelne PNG-Dateien im Dateisystem, die erst geladen werden, wenn das entsprechende Menu tatsächlich geöffnet wird. Dennoch ist dieser große Unterschied ein deutliches Indiz dafür, dass großes ungenutztes Potential für Maßschneiderung in der bisher manuell via Paketmanagement reduzierten Appliance liegt.

5.4.3 Maßschneiderungspotential von Paketmanagement

Das Paketmanagement, ein Mittel zur Maßschneiderung, kann im Rahmen einer Maßschneiderung selbst eingespart werden. Eine Betrachtung der größten im Testlauf komplett unbenutzten Verzeichnisse zeigt, dass verschiedene zum Paketmanagement gehörende Verzeichnisse mit mehr als 80MB (unkomprimiert) zur Größe der Appliance beitragen. Weitere knapp 50MB könnten sich einsparen lassen, wenn der bisher verwendete Distributionskernel durch einen zugeschnittenen ersetzt wird, der nur mit den notwendigen Treibern ausgestattet ist. Die von der Distribution mitgelieferten Kernelmodule belegen fast 50MB in der Appliance, während der für die Protokollierung erstellte statische Kernel komplett ohne Module auskommt und eine Größe von unter 2MB besitzt.

Um das Potential weiterer Maßschneiderung alleine durch Verwendung des Paketmana-

gements zu prüfen, wurde die Liste der „angefassten“ Dateien mit der vom Paketmanagement verwalteten Liste der installierten Dateien abgeglichen. Dabei ergibt sich, dass bei einer derzeitigen Gesamtzahl von 341 installierten Paketen nur Dateien aus 174 Paketen im Testlauf verwendet wurden, die übrigen 167 Pakete sind zumindest für den durchgeführten Testlauf unnötig und könnten entfernt werden. Bei diesen „nutzlosen“ Paketen handelt es sich einerseits um solche, die bei der manuellen Maßschneiderung übersehen wurden (beispielsweise *whois* und *installation-report*) und andererseits um Pakete, die lediglich durch Abhängigkeiten zwischen Paketen im System behalten werden (beispielsweise *cpio*, benötigt von *initramfs-tools*, welches von den Paketen des Distributions-Kernels benutzt wird). Die Gesamtgröße der Dateien dieser eventuell noch einzusparenden Pakete beläuft sich auf 152MB, also sehr viel weniger als die etwa 420MB Dateien, die laut Protokoll nicht angefasst werden. Dies ist ein starkes Indiz dafür, dass eine Maßschneiderung alleine auf Paketebene zum Erreichen einer geringen Größe der Appliance nicht ausreichend ist.

5.5 Vergleich virtualisierter Hardware

Untersucht wird der Einfluss der Virtualisierung auf den Energieverbrauch, HDD-Zugriffe und die CPU-Leistung. Es ist davon auszugehen, dass durch den Virtualisierungsoverhead ein Nachteil gegenüber dem Betrieb auf echter Hardware entsteht. Dieser Overhead kann bei den verschiedenen getesteten Virtualisierungslösungen unterschiedlich groß ausfallen. Aus diesem Grund wird in den Benchmarkergebnissen sowohl der Betrieb auf echter Hardware als auch auf virtueller Hardware sowie auf paravirtualisierter Hardware der verschiedenen Virtualisierungslösungen gegenübergestellt. Als Hostsystem kommt ein Rechner mit Ubuntu-Linux und Hardware-Virtualisierungsunterstützung zum Einsatz.

Verglichen werden diejenigen Virtualisierungslösungen, die unter Linux für Anwender leicht zu installieren sind. KVM, VirtualBox und VMWare werden untersucht auf Performanz, sowohl unter Verwendung von Hardware-Virtualisierungsunterstützung als auch ohne. Darüber hinaus wird der Einfluss von paravirtualisierten Treibern auf die Performanz gemessen, soweit solche Treiber verfügbar sind.

5.5.1 Vergleich der Energieverbrauchs

Zur Messung des Energieverbrauchs im laufenden Betrieb wurde alle fünf Sekunden im Host-System der Status der Batterieentladung (*/proc/acpi/battery/BAT0/state*) ausgelesen. Die Batterie liefert ihre derzeitigen Belastungen in Millivolt und Milliampere. Daraus lässt sich die Leistungsentnahme in Watt durch Multiplikation errechnen. Nimmt man nun an, dass die Werte innerhalb der vergangenen fünf Sekunden stabil waren, lässt sich durch aufaddieren (Multiplikator $\frac{5}{3600}$) ein Wert in Wattstunden angeben.

In einem exemplarischen Testlauf (Abbildung 10) ist in der Leistungsaufnahme zunächst der Bootvorgang, anschließend der CPU-Test (SuperPi) zu sehen. Daraufhin folgt der Festplattendurchsatztest (bonnie++) und das Herunterfahren der virtuellen Maschine.

Um die Vergleichbarkeit der Werte zu gewährleisten, werden die Ergebnisse zunächst mit fest eingestellten 800MHz (Abbildung 11) respektive 2000MHz (Abbildung 12) CPU-

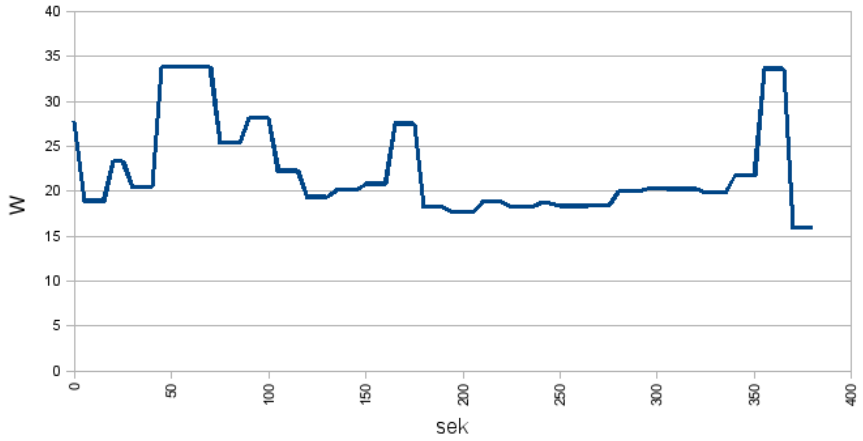

Abbildung 10: Einzelner Testlauf mit Boot → SuperPi → Bonnie++ → Shutdown

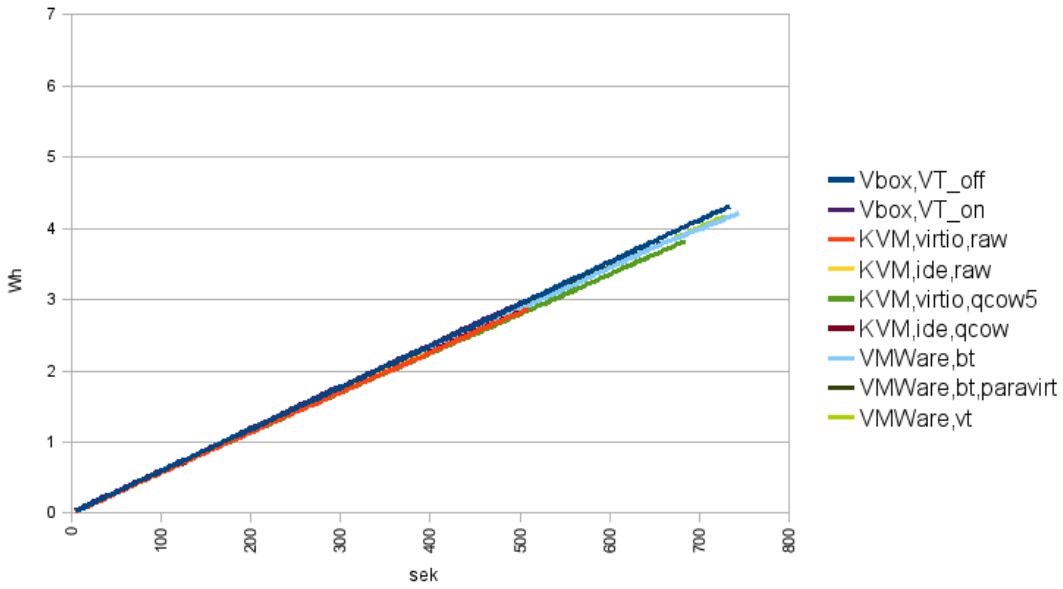

Abbildung 11: Leistungsentnahme der verschiedenen Konfigurationen bei festen 800MHz

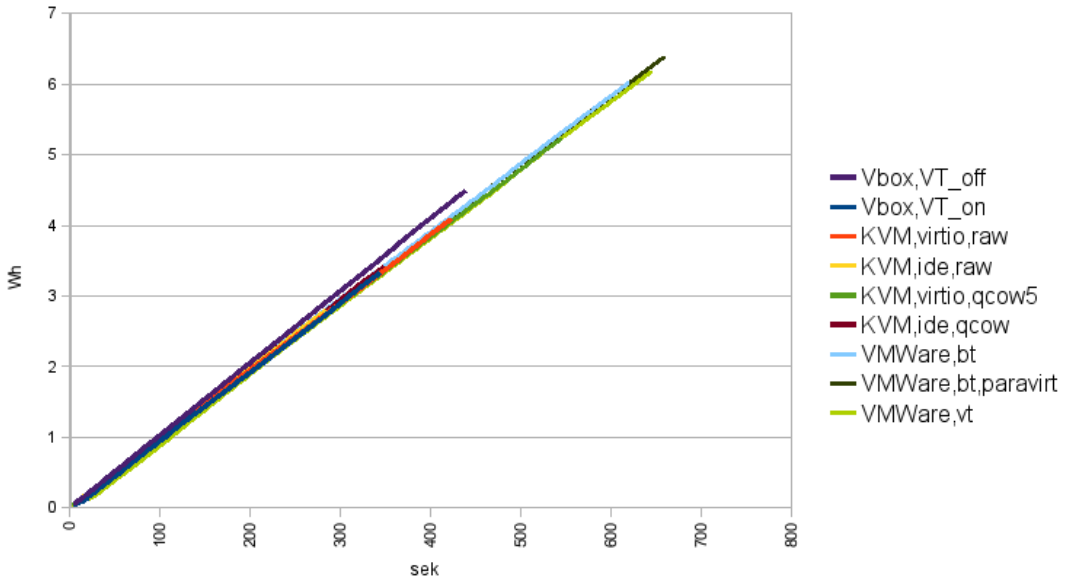

Abbildung 12: Leistungsentnahme der verschiedenen Konfigurationen bei festen 2000MHz

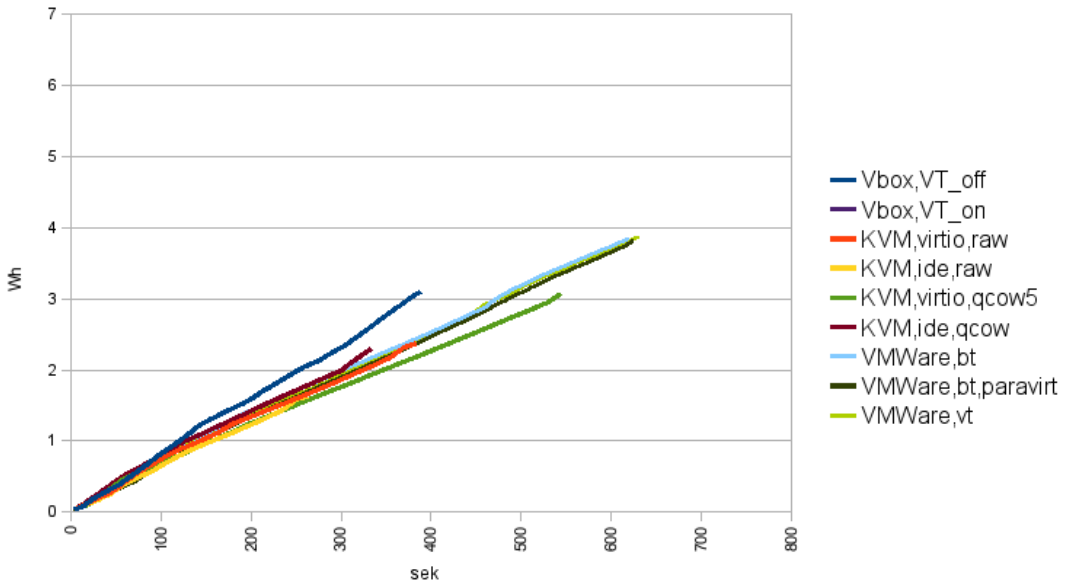

Abbildung 13: Leistungsentnahme der verschiedenen Konfigurationen bei variabler CPU-Frequenz

Frequenz erzeugt. Jedoch dominiert die CPU-Nutzung die Leistungsentnahme, so dass keine Unterschiede abzulesen sind.

Bei variabler CPU-Frequenz zwischen 800MHz und 2000MHz (Abbildung 13) ergibt sich ein differenzierteres Bild.

Das Betriebssystem der Testhardware ist die aktuelle Version 9.10 der Linux-Distribution Ubuntu. Diese ist leider nicht in der Lage das Standardbenchmark SuperPi zu starten; als Ursache ist ein Bug zu vermuten. Auf diese Weise sind keine Vergleichsmessungen auf echter Hardware gegen den Betrieb in virtualisierter Hardware möglich.

In den Energiebenchmarks hat sich somit gezeigt, dass die CPU-Frequenz der dominierende Faktor der Leistungsentnahme ist. Dies ändert sich auch bei variabler CPU-Frequenz nur geringfügig. So lässt sich für leistungskritische Benchmarks schließen, dass performantere Virtualisierungslösungen eine geringere Leistungsentnahme zeigen.

5.5.2 CPU-Performanztests

In diesem Abschnitt werden die Vergleichsergebnisse der Performanzbenchmarks aufgeführt. Im Anschluss folgt eine Deutung der Ergebnisse. Es werden mehrere Benchmarks auf mehreren Konfigurationen der Virtualisierungslösungen KVM, VirtualBox und VMware durchgeführt. Ein Shellscript ruft nacheinander die verschiedenen Virtualisierungslösungen auf und übergibt ihnen ein Benchmarkskript. Die virtuelle Maschine startet das Betriebssystem, führt das Benchmark aus und schreibt die Ergebnisse in eine Datei.

Um ein Benchmark für die Prozessorleistung zu erhalten, wurde mit Hilfe des Programms SuperPi, welches den Gauß-Legendre-Algorithmus nutzt, die Zahl Pi auf 1.048.576 Nachkommastellen genau berechnet. Die Zeiten, die dies unter den verschiedenen Virtualisierungslösungen dauert, werden schließlich verglichen.

Abbildung 14 zeigt die besten Ergebnisse einer jeden Virtualisierungslösung für das SuperPi-Benchmark. Es ist bei der Grafik zu beachten, dass die Werte nur etwa 10% auseinander liegen.

Abbildung 15 zeigt den Performanzunterschied, der sich ergibt, wenn die Nutzung paravirtualisierter Treiber abgeschaltet wird. Paravirtualisierte Treiber sind allerdings nur für KVM und VMware verfügbar. Auch in dieser Grafik sind die Unterschiede nicht größer als 15%.

Der Vergleich zwischen KVM und VMware, jeweils mit und ohne Paravirtualisierung, zeigt einen erheblichen Leistungsabfall bei VMware, wenn die Nutzung paravirtualisierter Treiber abgeschaltet wird. Die Nutzung der paravirtualisierten Treiber von VMWare führt also zu einem klaren Performanzgewinn bei der Ausführung von SuperPi. Bei KVM lässt sich ein solcher Unterschied nicht feststellen. Dies könnte daran liegen, dass KVM die hardwareunterstützte Virtualisierung (VT-x) nutzt und dieser paravirtualisierte Treiber hinzufügt. Die paravirtualisierten Treiber von VMware können hingegen nur in Kombination mit Binärübersetzung verwendet werden.

Der direkte Vergleich der besten Ergebnisse der verschiedenen Virtualisierungslösungen (Abbildung 14) zeigt einen Vorteil von etwa 10% von VMware gegenüber KVM und VirtualBox. Ein weiteres Benchmark für die CPU-Performanz ist das Kompilieren eines Linux-Kernels mittels gcc-Compiler. Hier wird für den Linuxkernel der Version 2.6.32.3 zunächst mittels *make allnoconfig* eine Minimalkonfiguration erzeugt und anschließend der Kernel

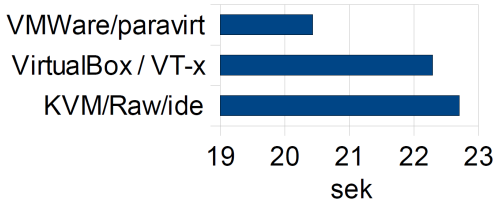

Abbildung 14: Beste Ergebnisse aller Virtualisierungslösungen

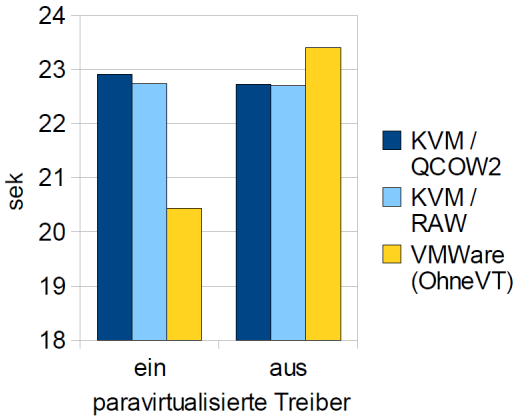

Abbildung 15: Performanzvergleich im Betrieb mit Hardware-Virtualisierungsunterstützung

kompiliert. Auch bei diesem Benchmark werden anschließend die Laufzeiten gegenübergestellt. In diesem Benchmark zeigt VirtualBox eine um etwa 50% schlechtere Leistung als KVM und VMware. In diesem Vergleich ist darüberhinaus zu sehen, dass der Betrieb auf dem System ohne Virtualisierung schneller ist als alle Virtualisierungslösungen. Dem nicht-virtualisierten Betrieb gegenübergestellt zeigt VMWare nur etwa 10% Virtualisierungsoverhead. KVM benötigt ein Drittel länger als der Betrieb auf echter Hardware. VirtualBox zeigt sich in diesem Benchmark abgeschlagen mit etwa 85% Virtualisierungsoverhead.

5.5.3 HDD-Performanztests

In diesem Abschnitt findet ein Vergleich der Festplatten-Datendurchsätze der Virtualisierungslösungen KVM, VirtualBox und VMware statt. Der Datendurchsatz beim Schreiben und Lesen wird mit dem Benchmark Bonnie++ erhoben. Auch Bonnie++ wird als Benchmarkskript an die virtuellen Maschinen übergeben und die Ergebnisse werden in eine Datei geschrieben. Vergleicht man die Durchsatzergebnisse der Virtualisierungslösungen miteinander, fällt auf, dass KVM und VirtualBox im Schreibdurchsatz erheblich bessere Werte erreichen als das native System (Abbildung 17). Dieses Verhalten lässt sich eventuell durch Cachingeffekte erklären, hervorgerufen durch die kleine virtuelle Festplatte. Im Vergleich der Virtualisierungslösungen zeigt die Grafik, dass KVM und VirtualBox weitaus bessere Werte im Schreibdurchsatz erreichen als VMWare. Nur im Lesezugriff zeigt sich der erwartete Virtualisierungsoverhead, wodurch die Werte sämtlicher Versuche auf virtualisierter Hardware schlechter ausfallen als die direkt auf der Hardware. KVM und VirtualBox liefern jedoch auch hier geringfügig bessere Werte als VMWare.

KVM kann verschiedene Imageformate als virtuelle Festplatten einbinden. Um herauszufinden ob sich zum Beispiel aus der Komprimierung des Formats QCOW2 Performanzunter-

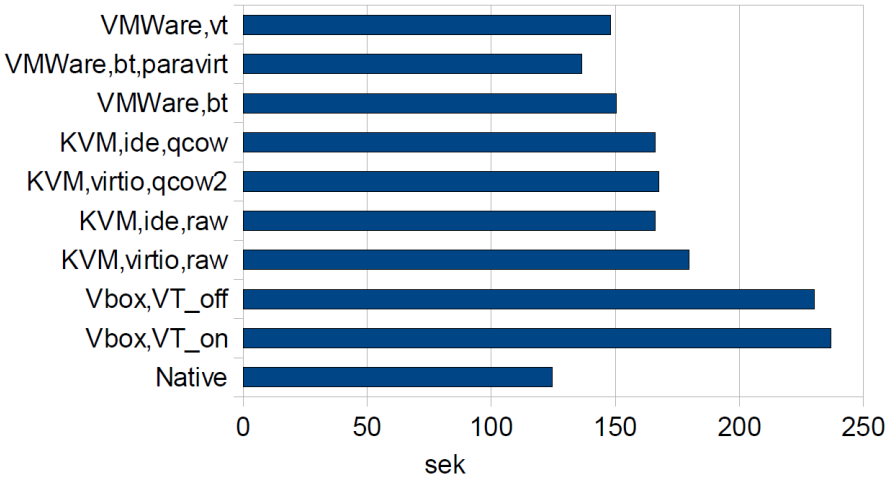

Abbildung 16: Kompilieren eines Kernels, Vergleich aller getesteten Konfigurationen

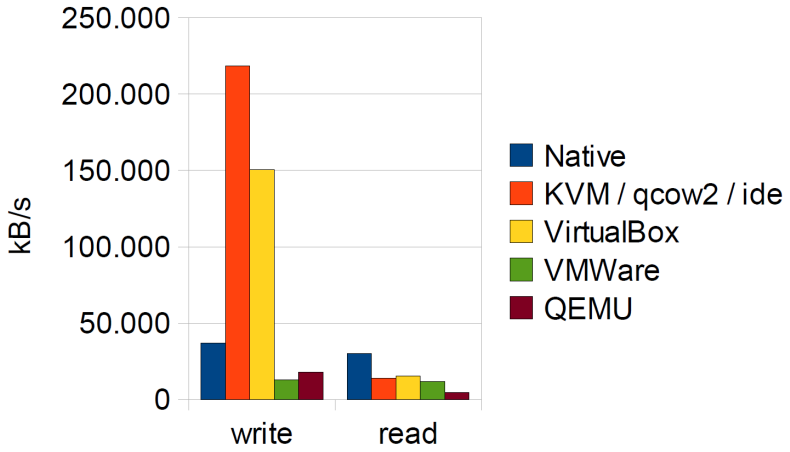

Abbildung 17: Bonnie++ Ergebnisvergleich aller Virtualisierungslösungen mit dem nativen System

Abbildung 18: Bonnie++ Vergleich der unterschiedlichen Imageformate von KVM

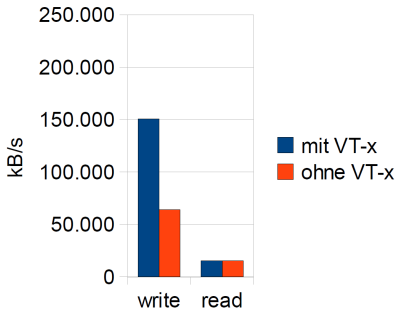

Abbildung 19: Bonnie++ in VirtualBox mit und ohne VT-x

schiede ergeben, werden die Durchsätze unter den verschiedenen Imageformaten gemessen und in Abbildung 18 dargestellt. Der Unterschied zwischen ein- und ausgeschalteter Paravirtualisierung ist bei KVM im Fall des Festplattendurchsatzes besonders deutlich zu sehen. Interessanterweise ist der Datendurchsatz ohne Paravirtualisierung weitaus größer. Der Unterschied zwischen den einzelnen Imageformaten ist dagegen nicht besonders groß. Das moderne und komprimierbare Format QCOW2 zeigt bei der Benutzung paravirtualisierter Treiber etwas schlechtere Ergebnisse als das Rohdatenformat raw. Die enormen Vorteile, die QCOW2 bietet, werden durch diesen Nachteil jedoch nicht aufgewogen.

Steht auf der realen Hardware VT-x nicht zur Verfügung (Abbildung 19), so ergeben sich für den Schreibdurchsatz unter VirtualBox starke Leistungseinbußen.

5.5.4 Dateisystem-Performanztests

Um die Performanz von Dateisystemzugriffen zu testen, werden 64MB Kernelsourcen entpackt und auf die Festplatte geschrieben. Bei den vielen, sehr kleinen Dateien in einem Kernelpaket dominieren die Festplattenzugriffszeiten das Benchmark. So wird mit diesem Benchmark nicht die CPU-Performanz gemessen, wie das bei starken Komprimierungen auf wenigen großen Dateien der Fall wäre. Das Ergebnis (Abbildung 20) zeigt den erwarteten Virtualisierungsoverhead bei allen Virtualisierungslösungen. Das beste Ergebnis erreicht also erwartungsgemäß der Betrieb direkt auf der Hardware, ohne Virtualisierungsschicht. Die besten Ergebnisse der Virtualisierungslösungen erzielt in diesem Versuch VirtualBox, sowohl mit, also auch ohne Hardware-Virtualisierungsunterstützung. KVM erreicht bei Verzicht auf den paravirtualisierten HDD-Treiber den zweiten Platz, gefolgt von VMWare. Am längsten hat das Entpacken mit KVM mit paravirtualisierten Treibern gedauert.

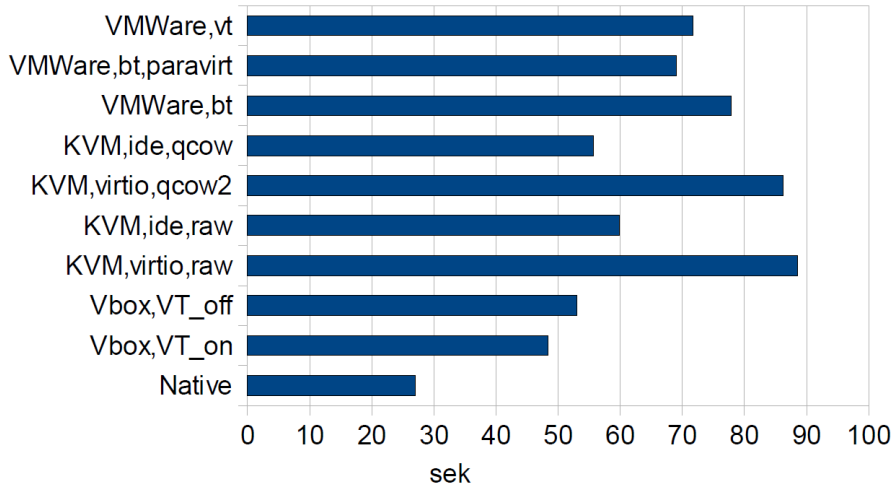

Abbildung 20: Entpacken von 64MB Kernelsourcen, Vergleich aller getesteten Konfigurationen

5.6 Checkliste Anforderungen

5.6.1 Anforderungen an den Server

MW1 Dienstangebot Priorität: 1 √
MW2 Erreichbarkeit Priorität: 1 √
MW3 Maßschneiderung Priorität: 1 ×
MW4 Sharing Priorität: 1 ×
MW5 VA Verwaltung Priorität: 2 ×
MW6 Datenintegrität Priorität: 2 √
MW7 Datensicherheit Priorität: 2 ×
MW8 Nutzerdaten Verwaltung Priorität: 3 ×

5.6.2 Anforderungen an den Client

C01 Dienstfindung Priorität: 1 ×
C02 Integration der Virtualisierungslösung Priorität: 1 ×
C03 VA Verwaltung Priorität: 1 √
C04 Zeitbeschränkte Laufzeit von VAs Priorität: 1 ×
C05 VA Update Priorität: 1 ×
C06 Sharing Priorität: 1 ×
C07 Authentifizierung von Benutzern Priorität: 1 ×
C08 Nutzerdaten lokal speichern Priorität: 1 ×
C09 Nutzerdaten zentral speichern Priorität: 3 ×
C10 Benutzerinterface Priorität: 3 √
C11 Plattformunabhängigkeit Priorität: 3 ×

C12 USB-Boot Priorität: 3 $\times$

5.6.3 Anforderungen an das Publisher-Tool

P01 VA Verwaltung Priorität: 1 $\times$
P02 VA Konfiguration Priorität: 1 $\times$
P03 Performanz Priorität: 1 $\times$
P04 Kosten Priorität: 1 $\surd$
P05 Usability Priorität: 1 $\times$
P06 VA update Priorität: 1 $\times$
P07 VA Zusammenstellung Priorität: 2 $\times$
P08 Plattformunabhängigkeit Tool Priorität: 2 $\surd$
P09 Plattformunabhängigkeit VA Priorität: 2 $\times$
P10 Zeitliche Begrenzung Priorität: 2 $\times$
P11 Kapselung Priorität: 2 $\surd$
P12 Automatisches Update Priorität: 2 $\times$
P13 Zielgruppe Priorität: 3 $\times$
P14 VA Statistiken Priorität: 3 $\times$

5.6.4 Anforderungen an die VA

A01 Universeller Einsatz Priorität: 1 $\surd$
A02 Peripheriezugriff Priorität: 1 $\surd$
A03 Performanz, Stabilität und Ressourcennutzung Priorität: 1 $\surd$
A04 Isolation Priorität: 1 $\surd$
A05 Systemverträglichkeit Priorität: 1 $\surd$
A06 Zugriffsschutz Priorität: 2 $\times$
A07 safety & security Priorität: 2 $\times$
A08 Fehlerbenachrichtigung Priorität: 3 $\times$

5.7 Fazit

In diesem Abschnitt wird zusammengefasst, welche Ergebnisse im Bereich des Client-Tools, des Servers und der Demo-VA durch den MS1 in Bezug auf das Gesamtprojekt erreicht wurden.

Client Das Client-Tool verfügt über eine grafische Bedienoberfläche und die Funktionen, die zum Herunterladen, Starten und Hochladen von Appliances nötig sind. Auch Dienstfindung funktioniert bereits, wenn sich der Client und der Server im gleichen Netzwerk befinden. Die Bedienung des Tools ist durch die grafische Oberfläche intuitiv und schnell zu verstehen. Sharing und Maßschneiderung werden bislang nicht berücksichtigt.

Server Der Serverteil des MS1 stellt bereits fast alle Funktionalitäten bereit, die vom Gesamtprojekt an den Server gefordert waren. Lediglich Maßschneiderung und Sharing werden auch hier noch nicht unterstützt. Der Server verursacht durch die Verwendung von WebDAV-PUT und CSV-Dateien viel geringeren Overhead als ein Upload mit dem normalen HTTP POST. Es existiert allerdings noch keine grafische Oberfläche zur Konfiguration des Servers. Des Weiteren ist noch keine dynamische Dienstfindung möglich, da der Server eine feste IP-Adresse hat und die Client-Dienstfindung auf diese Adresse eingestellt ist.

Demo-VA Zu diesem Zeitpunkt des Projektes existiert als Demo-VA eine Appliance für das Fach Betriebssystembau. Sie enthält einen Editor und alle Compiler, die zum Programmieren des Übungs-Betriebssystems in C++ nötig sind. Als Basisbetriebsystem wird eine Version der Debian-Distribution verwendet. Die VA ist von Hand maßgeschneidert und es wurden alle Module des Basisbetriebssystems entfernt, die nicht für die Bearbeitung des Projekts benötigt werden. Eine VA, die noch kleiner ist und nur für einen einzigen Zweck, beispielsweise das Starten von zwei auswählbaren Programmen, geschaffen wurde, ist noch nicht implementiert.

Gesamtprojekt Alles in allem stellt der MS1 im Groben die Funktionalität bereit, die DoVinci liefern soll. Grob bedeutet hier, dass der MS1 ein statisches System ist. Der Server ist nur über eine feste IP zu erreichen und es existiert nur eine Appliance, die heruntergeladen werden kann. Einzig das Client-Tool ist bereits in der Lage eine beliebige, vom Server angebotene Appliance zu verwenden.

Die Funktionen, Appliances auf das anwendungsspezifisch erforderliche Maß zu verkleinern (Maßschneiderung) sowie den Download und die Datenhaltung bereits vorhandener Dateien zu vermeiden (Sharing), sind im ersten Milestone von DoVinci noch nicht verwirklicht.

5.8 Ausblick nach MS1

Der Aufbau der zentralen Komponenten (Server mit Anbindung eines WLAN-Accesspoints, Client-/Publishertool) mit elementarem Funktionsumfang ist mit Abschluss des ersten Prototypen bzw. Milestone 1 erreicht und damit die Basis für weitere Vertiefungen in einzelne wichtige Aspekte gelegt.

Es wurden in den drei Bereichen des MS1 ca. 28% der Anforderungen an das Gesamtprojekt erreicht. Davon waren ca. 67% Anforderungen mit Priorität eins. Die Funktionalität, die DoVinci bieten soll, wurde damit bis auf eine wichtige Ausnahme erreicht. Das Thema Maßschneiderung und Sharing wurde in keinem der drei Bereiche des MS1 berücksichtigt.

Das Ziel der zweiten Projekthälfte ist es den ersten Prototypen mit starkem Fokus auf die Bereiche der Optimierung der Größe der einzelnen Appliance-Images (Maßschneiderung, maßgeblich auf Serverseite) sowie der Nutzung von Gemeinsamkeiten verschiedener Appliances (Sharing, maßgeblich auf Clientseite) weiterzuentwickeln. Hierzu werden verschiedene Ansätze und Tools in den Bereichen Maßschneiderung und Sharing untersucht

und ihre Eignung für das Projekt überprüft, um diese gegebenenfalls schließlich zu implementieren.

Alle weiteren Entwicklungen werden ab diesem Punkt im Rahmen des Milestone 2 stattfinden.

6 Maßschneiderung

Unter Maßschneiderung versteht man den Zuschnitt einer Lösung auf einen gegebene Eigenschaft hin. Solche Lösungseigenschaften können beispielsweise geringer Speicher- oder Hintergrundspeicherbedarf, hohe Stabilität oder sparsamer Energieverbrauch sein. Eine Lösung kann unter Berücksichtigung von Maßschneiderungsaspekten entwickelt werden oder einfach gegeben sein um anschließend maßgeschneidert zu werden. Eine Nebenbedingung der Maßschneiderung ist, die Lösung möglichst stark zu optimieren, ohne den erforderlichen Funktionsumgang zu reduzieren.

Die Forderung Maßschneiderung zu implementieren geht aus einigen Anforderungen der bisherigen Entwicklung hervor. Das sind die Performanzanforderung an die Publisherfunktion P03, die Maßschneiderungsanforderung an den Server MW3 und die Ressourcennutzungsanforderung an die Appliances A03.

6.1 Anforderungen

Die Maßschneiderung dient allgemein dazu, die Größe beliebiger Appliance-Images zu minimieren, ohne dabei deren die Funktionalität zu berühren, und ist damit wichtige Grundlage für das Sharing. Nicht benötigte Dateien sollen aus beliebigen Appliances möglichst vollständig entfernt werden.

M01 **Wirksamkeit der Maßschneiderung** **Priorität: 2**

Das Maßschneiderungspotential dateibasierter Maßschneiderung kann anhand eines Logs der in einem Referenzlauf verwendeten Dateien erfasst werden. Die Summe der Größen der nicht verwendeten Dateien stellen das Maßschneiderungspotential dar, da es ineffektiv wäre, angefasste Dateien zu löschen. Diese müssten später wieder hinzugefügt werden. Nach der automatisierten Maßschneiderung sollen wenigstens 30% des Maßschneiderungspotentials ausgeschöpft worden sein.

M02 **Automatisierung und Nutzerunterstützung** **Priorität: 1**

Die Maßschneiderung sollte vornehmlich automatisiert ablaufen. Alle Prozesse die ohne Nutzereingriffe möglich sind, sollen auf diese verzichten. Wenn Nutzerwissen benötigt wird, soll dieses geführt abgefragt werden. So sollen beispielsweise Komprimierungsverfahren automatisch angewendet werden; die Frage ob ein spezifisches Programm aus einer Appliance entfernt werden kann, soll hingegen dem Benutzer gestellt werden.

M03 **Dauer der Maßschneiderung** **Priorität: 1**

Der Prozess der Maßschneiderung soll in vertretbarer Zeit zu bewältigen sein. Nutzereingaben ausgenommen, soll der Prozess in weniger als einer Stunde abgeschlossen sein. Die Benutzereingaben sollen mit vernünftigen Default-Werten eingestellt sein, sodass die Nutzereingaben auch übersprungen werden können.

M04 **Unveränderte Funktionalität** **Priorität: 1**

Der Maßschneiderungsprozess soll die Funktionalität einer Appliance nicht funktional beeinflussen. Ausgenommen sind Anpassungen, welche lediglich den Komfort einschränken (z.B. durch Wegfall unnötiger Sprachübersetzungen eines Programms).

6.2 Entwurf

Das Ziel der Maßschneiderung in DoVinci ist die Reduktion des Footprints der virtuellen Appliances auf dem Hintergrundspeicher. In DoVinci ist die Maßschneiderung in drei Tools aufgeteilt, die jeweils grundverschiedene Ansätze zur Verschlankung von Appliances verfolgen.

In der ersten Phase sollen definitiv verzichtbare Dateien von der virtuellen Festplatte gelöscht werden. Log-Dateien und Caches können zum Beispiel im Allgemeinen gelöscht werden, ohne die Funktion der virtuellen Appliance zu gefährden. Dies wird im folgenden generische Maßschneiderung genannt.

In der zweiten Phase soll die Appliance von innen heraus untersuchen und herausgefunden werden welche Programme installiert sind. Es soll dann den Benutzer zu jedem installierten Programm fragen ob es tatsächlich in der Appliance benötigt wird. Falls nicht, sollen die nicht benötigten Programme entfernt werden. Dies wird im folgenden anwendergeführte Maßschneiderung genannt.

In der dritten Phase werden gelöschte Dateien durch Nullen ersetzen und anschließend die virtuelle Festplatte defragmentiert. Dies soll das Ergebnis der anschließenden Komprimierung verbessern.

Der Entwurf der einzelnen Tools wird im Folgenden vorgestellt.

6.2.1 Generische Maßschneiderung

Ziel der generischen Maßschneiderung ist es, eine beliebige Appliance soweit wie möglich zu minimieren, jedoch ohne dabei die zentrale Funktionalität der Appliance einzuschränken. Die generische Maßschneiderung unterscheidet sich zu der anwendergeführten Maßschneiderung in der Hinsicht, dass keinerlei Nutzer-Input nötig ist. Konkret sollen folgende Optimierungen vorgenommen werden:

1. Entfernung von ungenutzten Paketen über den vorhandene Paket-Manager

2. Entfernung von temporären Dateien (z.B. Browser-Cache, Downloads)

3. Entfernung von nicht benötigten Übersetzungsdateien (z.B. Entfernung von Sprachunterstützungen für Chinesisch, Japanisch, usw.)

6.2.2 Anwendergeführte Maßschneiderung

Das Grundprinzip der Maßschneiderung besteht darin, ein Programm an einen bestimmten Kontext anzupassen. In diesem konkreten Fall bedeutet Kontext, welche Programme der Benutzer verwenden will und welche nicht. Natürlich kann dies nicht automatisch ermittelt werden, sondern erfordert Informationen, die der Nutzer liefern muss. Die Idee des Tools der Anwendergeführten Maßschneiderung ist es, diese Fragen zu stellen. Der Nutzer wird so unter anderem gefragt, ob er bestimmte Programme benötigt. Eine Beispielfrage könnte lauten: „Benötigen Sie OpenOffice?“. Nachdem der Nutzer alle Fragen beantwortet hat, entfernt das Tool schließlich die nicht gewünschten Programme aus dem Image des Benutzers.

6.2.3 Dateisystembereinigung und Komprimierung

Viele Dateisysteme löschen Dateien, indem sie schlicht ihren Speicherort „vergessen“, die Datei jedoch auf der Festplatte belassen. Diese gelöschten, noch existenten, aber für das Betriebssystem unerreichbaren Dateien behindern den Erfolg einer Komprimierung der Festplatte in ein gepacktes Image. Das Maßschneiderungstool ersetzt die gelöschten Dateien durch Nullen und komprimiert anschließend das Image. Für diesen Zweck wird das zu komprimierende Festplattenimage an den virtuellen IDE-Controller der Optimierungsappliance angedockt. Nach dem Bootvorgang wird das zu komprimierende Image gemountet und das Tool *Zerofree* ausgeführt. Dieses Tool überschreibt eben die unreferenzierten Dateien mit Nullen. Ein weiteres Tool, *Zeroswap*, leert noch die optionale Swap-Partition. Danach wird die Appliance heruntergefahren, und das Image wird durch VirtualBox mit dem Befehl *compact* komprimiert, wobei alle 0-Bereiche aus dem dynamisch wachsenden Image herausgelöst und damit freigegeben werden.

6.3 Implementierung

Der Entwurf der Maßschneiderung sieht die Aufteilung in drei Tools vor. Um bessere Maßschneiderungsergebnisse zu erzielen, ist es wichtig, Kenntnis über das zu maßschneidernde Betriebssystem zu haben. DoVinci setzt voraus, dass die zu maßschneidernde Appliance ein Debian-Linux oder ein Derivat davon enthält.

Die erste Maßschneiderungsphase verwendet das Programm *BleachBit*. Dieses löscht Caches, Cookies, Protokoll-Dateien, temporäre Dateien und vieles mehr.

Die zweite Maßschneiderungsphase verwendet das Programm *Debfoster*. Dieses fragt den Benutzer, ob er die installierten Programme, zum Beispiel OpenOffice, tatsächlich in der Appliance benötigt. Falls nicht, entfernt es die nicht erwünschten Programme.

Die dritte Maßschneiderungsphase verwendet die Programme *Zeroswap* und *Zerofree* und eine Komprimierungslösung. Die ersten beiden schreiben Nullen auf die virtuelle Festplatte in Bereichen von Swap-Dateien und gelöschten Dateien. Anschließend wird die *genullte* virtuelle Festplatte komprimiert.

Die Implementierung dieser drei Tools wird im Folgenden beschrieben.

6.3.1 Generische Maßschneiderung

Bei der Implentierung des Tools, das die generische Maßschneiderung ausführt, zeigten sich einige Probleme. In ersten Versuch wurde ein Perl-Skript geschrieben, welches das Programm *BleachBit* ausführt. Dieses Skript sollte dann eine virtuelle Appliance als virtuelle CD (*.iso-Datei) übergeben bekommen und dann in der virtuellen Maschine die Minimierung vornehmen. Dieser Ansatz wurde allerdings verworfen, da die Maßschneiderung durch eine VA ausgeführt werden sollte. Diese OptimierungsVA bindet das zu maßschneidernde Image als zweite Festplatte ein und minimiert es dann. Aber auch dieser Ansatz funktionierte nicht für die generische Maßschneiderung, da das Programm *Bleachbit* nur die erste Festplatte in einem System modifizieren kann. Der dritte versuchte Ansatz ist, das Programm *BleachBit* im zu maßschneidernden Image vorinstalliert zu haben und innerhalb der OptimierungsVA dann den Befehl *chroot* auf die zweite Festplatte auszuführen. So kann *Bleachbit* erfolgreich ausgeführt werden. Da allerdings kein Basisimage zur Verfügung gestellt werden soll, auf dem das Programm schon vorinstalliert ist, wurde schließlich ein Installationsskript erzeugt, welches das benötigte Programm installiert und ausführt. Dabei wird ein Shellskript verwendet, welches *Bleachbit* aus einer lokalen Datei (die neben dem Skript vorliegt) in die VA kopiert. Dann wird ein Perl Skript gestartet, welches die Ausführung der generischen Maßschneiderung anstößt.

6.3.2 Anwendergeführte Maßschneiderung

Bei der Implementierung für das Tool 2 verhielt es sich ähnlich wie für Tool 1, welches für die generische Maßschneiderung zuständig ist. Es wurde erst versucht, mit Hilfe eines Perl-Skriptes das Programm *Debfoster* auf ein Image auszuführen, welches dem Skript übergeben wird. Später wurde *Debfoster* dann in die OptimierungsVA vorinstalliert, um von dort aus ein als zweite Festplatte angehängtes Image zu bearbeiten. Allerdings kann auch *Debfoster* nur auf der root-Festplatte arbeiten, so dass das Tool 2 ebenfalls (wie im Abschnitt Generische Maßschneiderung beschrieben) durch das dem Publisher zugeschickte Skript per *apt-get* vorinstalliert und ausgeführt wird.

6.3.3 Dateisystemkomprimierung

Durch die sehr speziellen Voraussetzungen an die Verwendung dieses Tools ist die Implementation anfangs davon geprägt gewesen Voraussetzungen zu prüfen. Schließlich stellte sich heraus, dass es sich bei der Lösung nur um eine Aneinanderreihung von Aufrufen vorhandener Programme handelt. Die aufgerufenen Programme sind *Zerofree* in der *Optimie-*

rungsVA und danach ein Aufruf von *VBoxManage* mit der Option *"–"compact* von außerhalb auf der virtuellen Festplatte.

Zerofree Dieses kleine Programm wird unter Linux und anderen UNIX-Derivaten dazu verwendet, die nicht genutzten Blöcke in *ext2*- und *ext3*-Dateisystemen mit Nullen zu überschreiben. Dies ist nötig da beim Löschen von Dateien in diesen Dateisystemen die beteiligten Blöcke nicht überschrieben, sondern nur als freier Speicher markiert werden. Dies ermöglicht unter anderem, gelöschte Dateien unter Umständen (wenn keine neuen Daten die alten Daten überschrieben haben) wiederhergestellt werden können. Diese selten genutzte Möglichkeit der Datenwiederherstellung ist jedoch nicht relevant für das Maßschneidern virtueller Appliances. Für dieses zählt einzig die Nutzung von Zerofree und ähnlicher Programme zur Verringerung der Größe von Appliances.

Zeroswap Weiterhin wurde im Laufe der Entwicklung ein Tool namens *Zeroswap* implementiert, welches äquivalent zu *Zerofree* funktioniert, jedoch auf Swap Partitionen wirkt. Dadurch können, in Kombination mit *VBoxManage "–"compact*, drei Arten von Partitionen(*ext2*, *ext3*, *swap*) erheblich auf eine nachfolgende Komprimierung optimiert werden.

VBoxMange "–"compact Die Funktion *compact* von *VBoxManage* wird genutzt um die Größe von VirtualBox Festplatten-Image-Dateien zu reduzieren. Dazu werden alle Blöcke, die nur aus Nullen bestehen, entfernt und speziell in der Datei vermerkt. Dadurch verbrauchen diese Null-Bereiche keinen Speicherplatz mehr. Der Erfolg dieser Komprimierung hängt jedoch maßgeblich von der Anzahl und Größe dieser Null-Bereiche ab. Um diese zu vergrößern bedarf es der vorherigen Ausführung von *Zerofree* und *Zeroswap*, die im Dateisystem der virtuellen Appliance die freien Blöcke mit Nullen überschreiben.

6.4 Evaluation

Die Maßschneiderung von virtuellen Appliances zielt in erster Linie darauf ab, den Umfang der Appliance zu verkleinern. Als Evaluationsgröße kommt daher die Speicherplatzeinsparung in Betracht. Da die zu erzielende Einsparung stark von der zu optimierenden Appliance abhängt, werden im folgenden Abschnitt eine Debian- Minimalinstallation, eine Debian Standardinstallation sowie die BSB-Appliance gegenübergestellt. Unterschieden werden auch die verschiedenen Einsparpotenziale. Die Einsparungen der verschiedenen Maßschneiderungstools werden also einzeln und innerhalb der Appliance betrachtet. Der Einfluss der Maßschneiderung auf den Erfolg der Komprimierung wird gesondert dargestellt. Hierzu wird die Größe des resultierenden Festplattenimages herangezogen.

6.4.1 Einsparungen der datei- und paketbasierten Tools

Als erstes wird das Tool BleachBit aufgerufen. Hierbei handelt es sich um ein generisches Maßschneiderungstool, welches keine Benutzereingaben erfordert. Mit diesem Tool gelingt es, die Debian-Minmalinstallation um 14% ihrer ursprünglichen Größe zu verkleinern. Diese

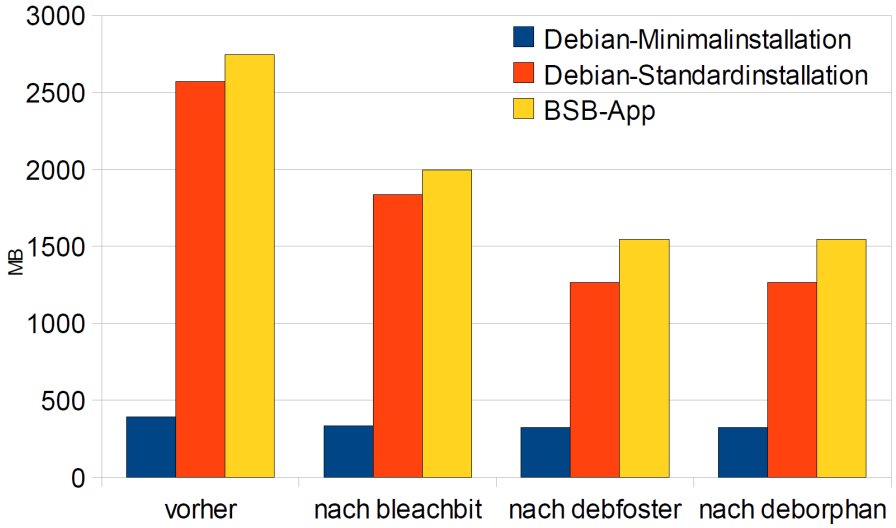

Abbildung 21: Einsparungen der datei- und paketbasierten Tools

geringe Einsparung lässt sich dadurch erklären, dass eine Debian-Minimalinstallation kaum noch Pakete enthält, die entfernt werden können. Bei einer Debian-Standardinstallation beträgt die Reduktion bereits 29%. Im Fall der BSB-Appliance, die zusätzliche Nutzerdaten enthält, die nicht eingespart werden können, ist die prozentuale Einsparung etwas geringer, während die absolute Einsparung nahezu identisch ist.

Beim zweiten Tool handelt es sich um Debfoster. Dieses Tool bietet dem Nutzer die Möglichkeit, nicht benötigte Pakete zu deinstallieren. Im Falle der Debian-Minimalinstallation ist es leicht möglich etwa 4% freien Speicherplatz zu gewinnen. Die Reduktion ist stark abhängig von den simulierten Benutzereingaben. Die Debian-Standardinstallation und die BSB-Appliance lassen sich um viele umfangreiche Pakete verringern, wie beispielsweise OpenOffice. Dies führt zu einer Verringerung der Größe um 31% im Falle der Debian-Standardinstallation und nur um 22% im Falle der BSB-Appliance, aus oben genanntem Grund.

Das dritte Tool ist Deborphan. Dieses entfernt Pakete, die von keinem Abhängigkeitspfad des Debian Paketmanagements mehr erreicht werden. Der Erfolg dieses Maßschneiderungstools fällt allerdings sehr gering aus, da Debfoster die Aufgabe von Deborphan bereits weitestgehend erfüllt. Die Einsparungspotentiale aller drei VAs liegen im Bereich von unter einem Prozent.

6.4.2 Einsparung durch Komprimierung

In diesem Abschnitt wird verstärkt untersucht, wie Komprimierung den Maßschneiderungserfolg beeinflusst. Konkret wird betrachtet, wie die Differenz der Größe einer VA vor und nach dem Maßschneiderungsprozess sich im Vergleich zur der Differenz der Größe einer VA verhält, die vor und nach dem Maßschneiderungsprozess komprimiert wird.

Die VA mit der Debian-Minimalinstallation lässt sich durch die Maßschneiderung von Do-

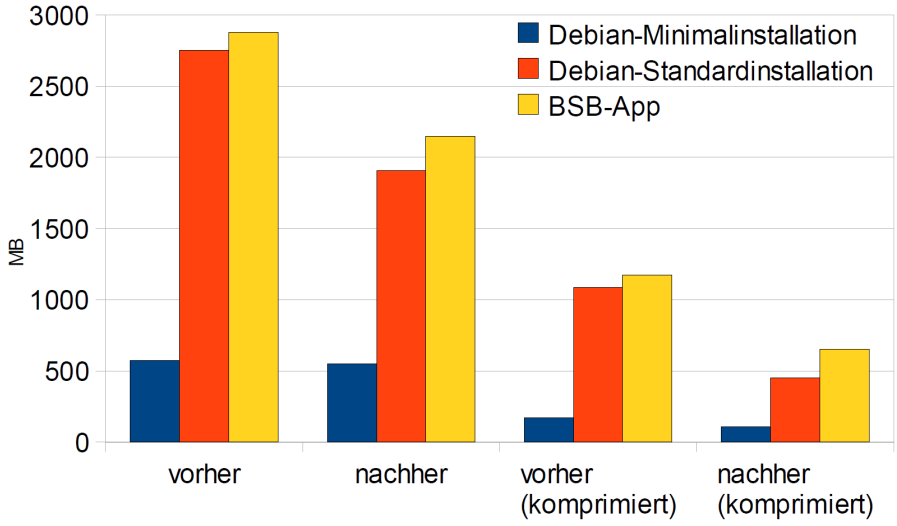

Abbildung 22: Einsparung durch Komprimierung

Vinci um nur 4% verkleinern. Es ist jedoch so, dass die Maßschneiderung auch auf bessere Komprimierungserfolge hin ausgelegt wurde. Deshalb werden nun sowohl die ursprüngliche VA, als auch die maßgeschneiderte VA komprimiert. Der Platzgewinn beträgt nun statt 4% schon 38%. Die Größe einer VA mit einer Debian-Standardinstallation verringert sich durch die Maßschneiderung um 31%. Die komprimierten Größen zeigen hingegen sogar eine Ersparnis von 58%. Im Falle der BSB-Appliance kann die Größe der VA ohne Komprimierung um 25% und mit Komprimierung um 45% reduziert werden.

Offensichtlich sind die Maßnahmen, die in der Maßschneiderung vorgenommen wurden, um die Komprimierungserfolge zu vergrößern, erfolgreich. Auffällig ist jedoch, dass die Größe einer VA nicht im gleichen Maße schrumpft, wie innerhalb der VA Speicherplatz durch Maßschneiderung gewonnen wurde. Eine Möglichkeit besteht darin, dass durch eine ungünstige Wahl der Blockgröße durch VirtualBox die eigentliche Dateigröße einer VA nicht den wirklich belegten Speicherplatz innerhalb der VA repräsentiert. Weiterhin ist es offenbar so, dass diese ungünstige Begleiterscheinung durch Komprimierung kompensiert werden kann. Ungeachtet einer möglichen Erklärung macht Komprimierung nach einem Maßschneiderungsvorgang in jedem Fall Sinn, da die resultierende VA dadurch erheblich verkleinert werden kann.

7 Sharing

Allgemein bezeichnet Sharing die gemeinsame Nutzung von Ressourcen. In DoVinci sind die Ressourcen, welche gemeinsam genutzt werden jene Dateien, die in mehreren Appliances identisch enthalten sind. Hierzu wird die Menge zu übertragender Dateien für eine neue, herunterzuladende Appliance abgeglichen mit der Menge bereits vorhandener Dateien. Idealerweise wird keine Datei identischen Inhalts doppelt heruntergeladen. Im Optimalfall muss für eine neue Appliance keine Datei heruntergeladen werden, die nicht direkt mit der Anwendung in Verbindung steht, die in der neuen Appliance enthalten ist.

7.1 Anforderungen

Ziel des Sharings ist die Nutzung des Wissens über gemeinsam genutzte Ressourcen verschiedener Appliances. So sollen die Schnittmengen möglichst effizient genutzt werden, um Vorteile auf Client-Seite zu erzielen und die Übertragungsmengen gering zu halten.

S01 **Differenz** **Priorität: 1**

Wird eine Appliance in exakt gleicher Version ein zweites Mal über das Client-Tool vom Server geholt, sollen keine weiteren Dateien mehr heruntergeladen werden müssen, mit Ausnahme der Meta-Daten. Generell sollen Schnittmengen zwischen verschiedenen Appliances auf Dateiebene erkannt werden, so dass diese Dateien weder doppelt heruntergeladen, noch doppelt gespeichert werden müssen.

S02 **Performanzeinbußen** **Priorität: 2**

Die Performanz von Festplatten Lese- und Schreibzugriffen soll aufgrund des Sharings um nicht mehr als 25% reduziert werden.

S03 **Platzoverhead** **Priorität: 2**

Der durch Sharingunterstützung zusätzlich benötigte Festplattenspeicher einer Appliance soll nicht mehr als 15% der Ursprungsgröße betragen.

S04 **Transparenz** **Priorität: 1**

Das Sharing soll für den Nutzer transparent sein.

S05 **Nutzerdaten** **Priorität: 1**

Vom Nutzer eingebrachte Dateien und Änderungen sollen nicht über das Sharing in andere Appliances oder gar zum zentralen Server übertragen werden können.

S06 **Paralleler Dateizugriff** **Priorität: 1**

Auch wenn eine Datei von mehreren Appliances genutzt wird, sollen diese Appliances zeitgleich gestartet werden können.

7.2 Entwurf

In DoVinci soll Sharing zwischen virtuellen Appliances betrieben werden, genauer gesagt soll Sharing auf Dateiebene angeboten werden, um den von mehreren virtuellen Appliances verbrauchten Speicherplatz auf der Festplatte des Nutzers zu minimieren.

Dazu sollen die Dateien der Appliances in einem gemeinsamen Objectstore gehalten werden, auf den mehrere virtuelle Appliances zugreifen. Weiterhin soll das System in der Lage sein, die durch das Dateisystem verwalteten Dateien an verschiedenen Orten aufzubewahren. Darüber hinaus soll es möglich sein, Dateien die zum Betrieb einer Appliance benötigt werden, zur Laufzeit von einem entfernten Objectstore nachzuladen. So soll es möglich sein, dass Dateien, die höchstwahrscheinlich nicht benötigt werden, nur dem Anschein nach lokal vorhanden sind. Ihr Speicherplatzbedarf soll so vollständig eingespart werden, bis die Datei tatsächlich einmal angefordert wird. Dann muss sie jedoch mit erheblichen Performanzeinbußen von einem entfernten Objectstore nachgeladen werden. Um dieses Verhalten innerhalb der Appliances zu ermöglichen ist die Entwicklung eines eigenen Dateisystems, dem *DoVinciFS*, notwendig.

7.2.1 Appliance-Vorbereitung

Die Idee der Appliance-Vorbereitung ist, die Appliance soweit wie möglich auf das Sharing vorzubereiten. Das Sharing-Tool erweitert eine gegebene Appliance um Sharing in DoVinci und trägt die enthaltenen Dateien in den lokalen Objectstore ein. Um die Datei innerhalb verschiedener Appliances zu identifizieren wird jeder enthaltenen Datei eine eindeutige Identifikationsnummer ID zugewiesen. DoVinciFS muss deshalb in alle VAs eingebunden werden, die von einem Publisher erstellt und hochgeladen werden. Außerdem werden die Dateiinhalte aus einzelnen Dateien entfernt und in dem Objectstore gespeichert. Wie IDs berechnet werden und Dateiinhalte in den Objectstore eingebracht werden, ist in Abschnitt 7.3.3 erklärt. Die Vorbereitung der Appliances bedeutet nur den Schritt der Einbindung von DoVinciFS in ein VA-Image.

7.2.2 Dateisystem

Das Dateisystem *DoVinciFS* soll es ermöglichen, von mehreren VAs aus auf einen gemeinsamen Datenbestand zuzugreifen. Dateien die in mehreren VAs identisch vorkommen, sollen so möglichst nur einmal auf dem Hostrechner vorgehalten werden. Um dieses Ziel zu erreichen, muss ein Dateisystem entworfen werden, welches den normalen

Dateisystem-Baum der VA aufteilt. Diese Aufteilung ermöglicht die Trennung verschiedener VA-spezifischer Daten, dazu gehören Zugriffszeiten, Dateirechte, der Dateiname und das Verzeichnis der Datei auf der einen Seite und die gemeinsam nutzbaren Daten, also den Dateiinhalt selbst auf der anderen Seite. Zur Herstellung dieser Trennung wird in der VA eine Modifikation des ursprünglichen Dateibaums vorgenommen, bei der lediglich der Inhalt der Dateien entfernt wird. Dadurch sind alle betroffenen Dateien nur noch 0 Bytes lang. Als zusätzliche Information wird hinterlegt, unter welcher ID der Dateiinhalt abgespeichert wurde. DoVinciFS muss dabei nicht wissen, wie die Vergabe einer ID erfolgt, es muss lediglich dazu in der Lage sein, aus einer ID den tatsächlichen Speicherort des Dateiinhalts zu ermitteln.

Zur Speicherung der ID sind verschiedene Möglichkeiten denkbar – in einer getrennten Datenbank, an Stelle des ursprünglichen Dateiinhalts, oder als erweitertes Attribut der gekürzten Datei. Letztere Methode ist die für dieses Problem wohl einfachste Lösung, da die relevanten Informationen ohne Mehraufwand auch dann noch erhalten bleiben, wenn die Datei beispielsweise umbenannt oder verschoben wird. Eine Speicherung an Stelle des ursprünglichen Dateiinhalts kommt dagegen nicht in Frage, da die ungekürzten Dateien beliebige Inhalte besitzen können. Daher könnte sonst der Dateiinhalt mit einer ID verwechselbar sein.

Die Möglichkeit, im „gekürzten“ Dateisystem doch wieder Dateien unterzubringen, die ihren Inhalt selbst enthalten und keine externen Daten referenzieren, erlaubt es, Schreibzugriffe einfach durch Einkopieren des Orginalinhalts und Entfernen der ID umzusetzen. Ein direkter Schreibzugriff auf Dateien im Objectstore darf natürlich nicht erfolgen, da alle dort abgelegten Dateien potentiell von einer anderen VA benötigt werden, die in dieser Datei den unveränderten Inhalt erwartet. Neuberechnung einer ID und erneute Ablage im Objectstore würde dagegen dazu führen, dass bei stückweisem Schreiben einer Datei potentiell sehr viele „Dateileichen“ im Objectstore landen, die nur einem temporären Zustand bei unvollständig geschriebenen Daten entsprechen. Eine Rückübertragung von veränderten Dateien in den Objectstore würde darüber hinaus die auf Clientseite notwendige Verwaltung der Zuordnung von VA zu Objectstore-Dateien erschweren. Diese ist notwendig, um beim Löschen einer VA alle nicht mehr benötigten Dateien aus dem Objectstore zu löschen.

Damit die Einbindung von DoVinciFS mit vertretbarem Aufwand in eine möglichst große Breite von Distributionsvariationen möglich ist, werden lediglich die Verzeichnisse /usr, /home, /opt und /var in den Objectstore ausgelagert. In UNIX-Systemen beinhalten diese vier Verzeichnisse üblicherweise den Großteil der Datenmengen des Systems und werden bei großen Installationen traditionell auf getrennte Partitionen oder Festplatten verteilt. So ist es leicht und unabhängig von der Distribution möglich, sie während des Systemstarts mittels DoVinciFS einzubinden.

7.2.3 Objectstore

Der Objectstore realisiert zentrale Speicherung für Dateiinhalte von Appliances. Die Dateiinhalte werden über Hashwerte referenziert und können bei Bedarf aus der Appliance heraus von einem Objectstore nachgeladen werden. Stellt sich beim Zugriff auf eine Datei heraus, dass die Daten bisher nicht im Dateisystem der Appliance vorhanden sind, wird zu-

nächst der lokale Objectstore auf dem Hostsystem des Benutzers abgefragt. Befindet sich die gewünschte Datei auch nicht in diesem, so erfolgt eine weitere Abfrage an einen global verfügbaren Objectstore über ein Netzwerk. Von dort wird der entsprechende Dateiinhalt heruntergeladen und ab diesem Zeitpunkt im lokalen Objectstore vorgehalten.

Die Verwendung eines Hashes (die derzeitige Implementierung verwendet SHA-1) verhindert die Notwendigkeit zentraler Kontrolle über eine eindeutige Referenz von Dateiinhalten und erleichtert den Umgang mit harten Dateisystemlinks und mehrfach existierenden, äquivalenten Dateien. Somit ist es möglich, ein lokales, oder im Internet dezentrales, Netz von Objectstores aufzubauen. Dies ermöglicht sowohl Lastverteilung, als auch Ausfallsicherheit dieses Prinzips. Die Konsistenz von Objectstores ist zumindest prinzipiell durch die Hash-Referenzierung sichergestellt. So lässt sich auch in der VA der Hash einer heruntergeladenen Datei überprüfen.

Eine spätere Erweiterung um Sharing auf Block-Ebene innerhalb eines oder mehrerer Objectstores könnte transparent implementiert und eingeführt werden, um den benötigten Speicherplatz noch weiter zu reduzieren. Schon ein zusätzliches Leerzeichen in einer Datei bedingt einen unterschiedlichen Hashwert bei ansonsten gleichem Inhalt.

Die Realisierung des Objectstores lässt sich über die Kombination verschiedener existierender Lösungen erreichen. Der Entwurf berücksichtigt lediglich den Weg von Dateiinhalten aus dem Objectstore in eine Appliance. Es ist nicht vorgesehen nachträglich geänderte Dateien wieder in einem Objectstore zur Verfügung zu stellen. Als simple Lösung bietet sich ein SQL-Server an, welcher die Hashes auf Dateiinhalte abbildet. Vermutlich genügt auch die Möglichkeit von HTTP-Downloads und der Einbindung des lokalen Objectstores über die Shared-Folder-Funktion von VirtualBox.

7.3 Implementierung

Die während der Projektgruppe erstellte Implementation von Sharing benutzt das Betriebssystem Linux und die sogenannte FUSE[14]-Bibliothek, die es vereinfacht ein neues Dateisystem zu implementieren. Durch die starke Vereinfachung kann bei der Implementierung primär auf die Sharing-Aspekte des entstehenden Dateisystems geachtet werden.

Der Objectstore wird in Form einer Verzeichnisstruktur angelegt, in der Dateien liegen, deren Dateiname der Hash-Code ihres Inhalts ist. Der Zugriff auf diesen Objectstore wird mit Hilfe der Shared-Folders von VirtualBox realisiert. Diese ermöglichen eine Kommunikation zwischen Host- und Gastsystem für den Austausch von Dateien.

Das Dateisystem ist generisch gestaltet, so dass es unter vielen verschiedenen Konfigurationen betrieben werden kann. Die Konfiguration wird während der Laufzeit durch das DoVinci-Client-/Publishertool vorgenommen und ermöglicht einen hohen Grad an Flexibilität. So kann die Position und der Zugriffsweg auf den entfernten Objectstore vom Client-/Publishertool vorgegeben werden.

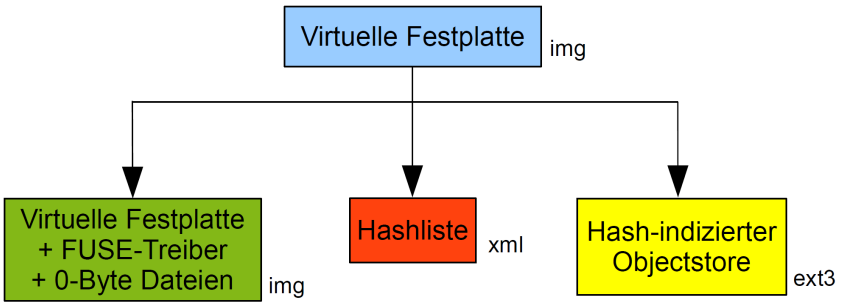

Abbildung 23: Vorbereitung eines Festplattenimages auf Sharing in DoVinci

7.3.1 Appliance-Vorbereitung

Ein gegebenes Image wird bei der Vorbereitung für Sharing in DoVinci in mehrere Komponenten aufgeteilt. Dies erfolgt, bevor das Image als Festplatte in eine Virtualisierungslösung eingebunden wird. Zur Anpassung des Dateisystems an das DoVinciFS wird ein Skript ausgeführt. Die Abbildung 23 zeigt die resultierenden Komponenten des Images nach der Aufteilung. Es entstehen:

- Ein deutlich verkleinertes Image der virtuellen Festplatte: Dieses besteht aus wenigen zum Booten notwendigen Dateien, 0-Byte-Dateien mit ihren Hash-IDs und dem Treiber für DoVinciFS.
- Eine Hashliste der enthaltenen Dateien: In dieser Liste werden die Hashes sämtlicher 0-Byte-Dateien der virtuellen Appliance gesammelt. Dadurch kann leicht nachgeprüft werden, welche Dateien übertragen werden müssen und welche nicht, falls sie am Ziel schon vorhanden sind.
- Ein Hash-indizierter Objectstore: Der Objectstore enthält die tatsächlichen Dateiinhalte, indiziert durch ihren Hashwert. Auf diese Weise kann das DoVinciFS aus dem Image heraus, anhand des Hashwertes, den Dateiinhalt geliefert bekommen.

Damit das Skript fehlerfrei arbeiten kann, müssen die Dateien dovincifs, mount.dovinci und VBoxLinuxAdditionsx86.run gemeinsam in einem Verzeichnis liegen. Leider wurden derzeit nur Debian-Distributionen getestet. Deshalb wird in einem Vorbereitungsschritt überprüft, ob ein Debian vorliegt, bevor die eigentliche Installation von DoVinciFS begonnen wird. Erst wenn beide Bedingungen erfüllt sind, können die von DoVinciFS benötigten Bibliotheken installiert und in das Verzeichnis */lib* kopiert werden. Schließlich müssen noch die DoVinciFS-Dateien in das Verzeichnis */bin* kopiert werden, damit die Vorbereitung abgeschlossen ist. Danach wird die Virtualisierungslösung gestartet. Das Ergebnis ist eine virtuelle Appliance, die auf einen Objectstore zugreift und somit Sharing unterstützt.

[14] Filesystem in Userspace: `http://fuse.sourceforge.net`

7.3.2 Dateisystem

Das Dateisystem *DoVinciFS* ist die Schnittstelle zwischen den auf 0 Bytes reduzierten Dateien der Appliance und ihrem tatsächlichen Inhalt im Objectstore. Um zum einen die Entwicklung zu vereinfachen und zum anderen eine Einbindung in möglichst viele Appliances zu ermöglichen, basiert DoVinciFS auf FUSE. Dieses System ermöglicht es unter Linux, Dateisysteme zu schreiben, deren Hauptteil nicht klassisch im Kernel, sondern im Userspace abläuft. Damit entfällt zum einen die fehlerträchtige Entwicklung von Kernelcode und zum anderen muss das Dateisystem nicht jedes Mal speziell für den in gerade dieser Appliance verwendeten Kernel kompiliert werden.

Die Hauptinformationen, welchen Inhalt DoVinciFS unter seinem Mountpoint präsentieren soll, stammen wiederum aus dem Dateisystem der Appliance selbst. Da auch bei identischem Dateiinhalt in zwei VAs weitere Attribute der Datei, wie beispielsweise die Zugriffsrechte oder die letzte Modifikationszeit, unterschiedlich sein können, gibt es einen vollständigen Dateibaum aller von DoVinciFS behandelten Dateien, bei denen lediglich der Dateiinhalt entfernt wurde. Die Dateilänge ist also auf 0 Byte reduziert. Dies ermöglicht es, fast alle Zugriffe in DoVinciFS direkt auf den Referenzbaum abzubilden – lediglich bei der Abfrage der Dateigröße und bei Zugriff auf den Dateiinhalt sind Eingriffe notwendig.

DoVinciFS muss natürlich eine Möglichkeit haben, die wahre Dateigröße zu ermitteln und Datenzugriffe auf den wirklichen Inhalt der Datei abzubilden. Um dies mit möglichst geringem Aufwand zu ermöglichen, werden die Dateigröße und eine Datei-ID in den erweiterten Attributen der Leerdatei gespeichert. Den tatsächlichen Inhalt einer Datei kennt DoVinciFS nicht. Es weiß nur, dass sie eine bestimmte Länge hat, und wie es aus einer ID einen Pfad bestimmt, unter dem der tatsächliche Dateiinhalt erreichbar ist. Die derzeitige Implementierung der Objectstore-erzeugenden Tools verwendet die SHA1-Checksumme des Dateiinhalts als ID. Fehlen diese Attribute beim Zugriff auf den Referenzbaum, so nimmt DoVinciFS an, dass die Datei im Referenzbaum nicht in den Objectstore ausgelagert wurde und greift direkt auf sie zu. Dies ermöglicht auch die Umsetzung von Schreibzugriffen. Bei einem Schreibzugriff wird der Dateiinhalt vom Objectstore in die Datei im Referenzbaum umkopiert und die erweiterten Attribute entfernt. Würde dies nicht gemacht, würde ein Schreibzugriff die Datei im Objectstore ändern, was sich potentiell auf andere Appliances auswirken kann, die die gleiche Datei referenzieren.

Weiterhin ist in DoVinciFS das dynamische Nachladen von Dateien implementiert, die im Objectstore fehlen. Wenn bei einem Objectstore-Zugriff die gewünschte Datei nicht vorhanden ist, aber DoVinciFS eine URL zum Nachladen mitgeteilt wurde, so wird automatisch versucht die fehlende Datei von dort nachzuladen und so der Inhalt des Objectstores ergänzt. Wenn der Download fehlschlägt, wird dem System in der VA ein Ein-/Ausgabefehler (EIO) beim Zugriff signalisiert.

Des Weiteren existiert eine Kommunikationsschnittstelle zwischen Appliances und dem Client-Tool. Sollte das Nachladen von Dateien fehlschlagen, wird mit Hilfe dieser Schnittstelle überprüft, ob eine Netzwerkverbindung zum Server aufgebaut werden kann. Sollte dies nicht der Fall sein, wird anstelle des Ein-/Ausgabefehlers (EIO) die Appliance angehalten. Den Benutzer erreicht dann eine Mitteilung, dass er sich mit dem Internet, oder einem anderen Netzwerk verbinden soll, von welchem aus der DoVinci-Server erreichbar ist.

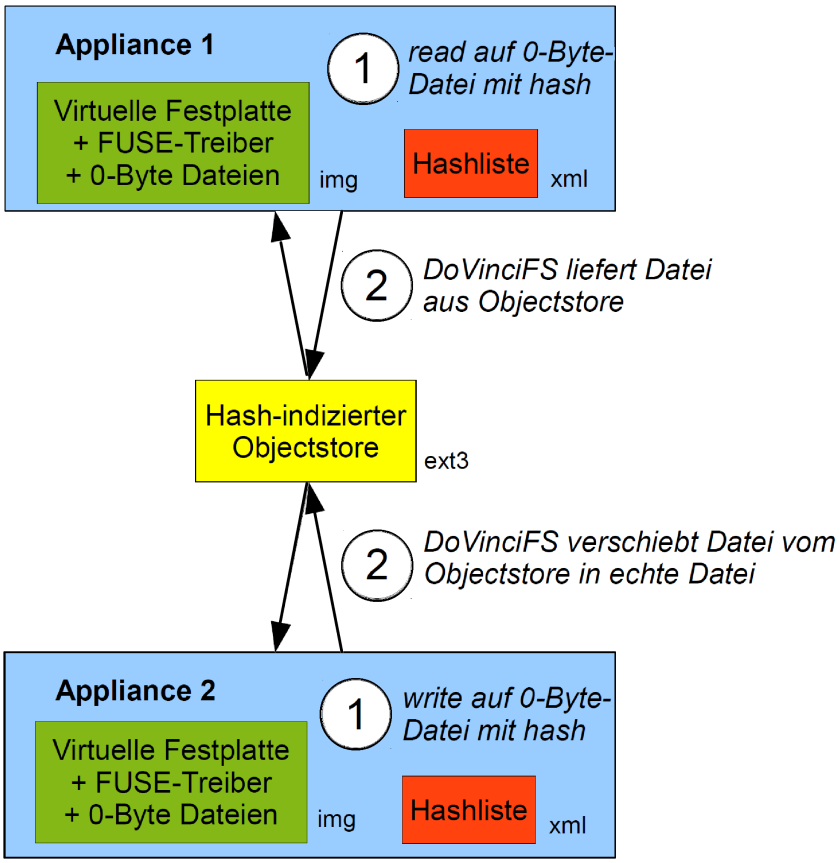

Abbildung 24: Festplattenzugriffe unter Verwendung des Objectstores

Für die Ziele im Bereich Maßschneiderung (siehe 6) und für Debugging-Aufgaben ist es auch nötig, verfolgen zu können, auf welche Dateien im DoVinciFS zugegriffen wurde. Dazu wird eine Möglichkeit implementiert eine Liste von angefassten Dateien in eine beliebige Datei zu schreiben. Um den Overhead dieser Listenerzeugung zu reduzieren, Speicherplatz zu sparen und das Parsen sowohl automatisch als auch manuell zu vereinfachen, wird ein Cache verwendet. So wird ein Dateizugriff nur dann protokolliert, wenn die zugegriffene Datei nicht zu den letzten 1000 angefassten Dateien zählt.

7.3.3 Objectstore

Die Realisierung des Objectstores gliedert sich in zwei Bereiche, den lokalen Objectstore, also das Speicherverfahren von gemeinsam genutzten Dateien auf dem Clientcomputer, und dem entfernten Objectstore auf einem (weltweit) erreichbaren Server.

Der entfernte Objectstore lässt sich gut in die schon vorhandene Infrastruktur eingliedern. Wie bereits im Abschnitt zum Dateisystem erwähnt, werden Dateien nachgeladen, welche sich nicht im lokalen Objectstore befinden. Das Hochladen von Dateien in den Objectstore findet nur während des Publishingprozesses statt und ist dort beschrieben, nachträglich ist

diese Funktionalität nicht notwendig und demzufolge auch nicht geplant. Es genügt also serverseitig, den Download von Dateien anzubieten. Die Dateien werden über die vorhandenen Hashes referenziert und können so heruntergeladen werden. Die Integritätsprüfung von geteilten, heruntergeladenen Dateien kann innerhalb der VA beim Zugriff stattfinden. Der Hash über den Dateiinhalt liefert eine Aussage zum fehlerfreien Abschluss des Downloads. Derzeit findet keine on-the-fly-Überprüfung dieses Hashes statt, um die Zugriffgeschwindigkeit nicht weiter zu senken.

Informationen über die Verwaltung des Objectstores auf dem Server finden sich in dem entsprechenden Abschnitt zum Server.

Der Objectstore auf dem Clientcomputer unterscheidet sich prinzipiell nicht von der im Entwurf diskutierten Lösung. Die gewählte Virtualisierungslösung VirtualBox ermöglicht über sogenannte „Shared Folder“ von Haus aus den Austausch von Dateien zwischen virtueller Appliance und Hostsystem. Diese geteilten Dateiordner werden genutzt, um innerhalb der Appliances auf den Objectstore zuzugreifen. Bei der DoVinci-Installation wird ein Ordner konfiguriert, welcher als Objectstore genutzt wird. Dieser wird beim Herunterladen einer VA durch den DoVinci-AppManager mit den von der VA benötigten Dateien gefüllt. Diese Dateien eignen sich für Sharing, werden von vorhandenen VAs benötigt und sind nicht bereits in dem Objectstore vorhanden. Beim Erstellen der VA durch den AppManager wird dieser Ordner als „Shared-Folder“ (SF) für diese VA zugewiesen. Innerhalb der VA lässt sich der SF wie eine Festplatte einbinden und wird von dem DoVinci-Dateisystem genutzt.

Beim Löschen von VAs auf dem Clientcomputer kann es vorkommen, dass Dateien im Objectstore liegen, welche nicht mehr benötigt werden. Um diese zu entfernen werden die erstellten Hash-Listen des Publishing-Prozesses genutzt. Der AppManager prüft auf Anforderung, ob alle im Objectstore vorhandenen Dateien von mindestens einer VA genutzt werden und löscht diese andernfalls.

7.4 Evaluation

In diesem Abschnitt wird untersucht, inwieweit das Sharing erfolgreich ist, und welche Nachteile es mit sich bringt. Die wichtigste Größe in diesem Abschnitt ist der eingesparte Speicherplatz beim Betrieb von mehr als einer Appliance. Anschließend werden die Einflüsse der Sharing-Implementierung auf die Performanz von Lesezugriffen gemessen. Hierzu wird die Kompilationszeit eines Linux-Kernels als Benchmark herangezogen, wie das schon bei der Evaluation des ersten Milestones der Fall war (siehe 5.5.2). Eine für den späteren Endnutzer besonders wichtige Messgröße ist die Bootdauer, auch diese wird im folgenden Abschnitt mit dem Betrieb ohne Sharing verglichen.

7.4.1 Eingesparter Speicherplatz

Es ergeben sich durch das Sharing Einsparungseffekte zwischen mehreren Appliances. Bei paarweiser Betrachtung von je zwei VAs mit unterschiedlicher Zielsetzung (aber jeweils auf Basis eines Debian-Systems) können etwa 20% bis 25% Platz im Objectstore durch gemeinsame Nutzung von Dateien gespart werden. Wird eine dritte Appliance hinzugezogen,

so steigt der Einsparfaktor auf knapp 35%, das System erzielt also wie erwartet eine bessere Effizienz, wenn mehr Appliances untereinander ihre Daten gemeinsam nutzen.

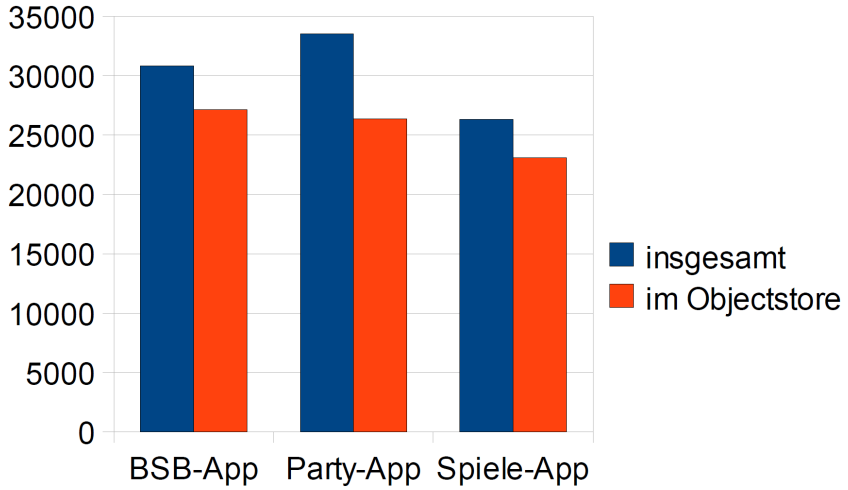

Abbildung 25: Vergleich der Gesamtanzahl Dateien zu jenen im Objectstore

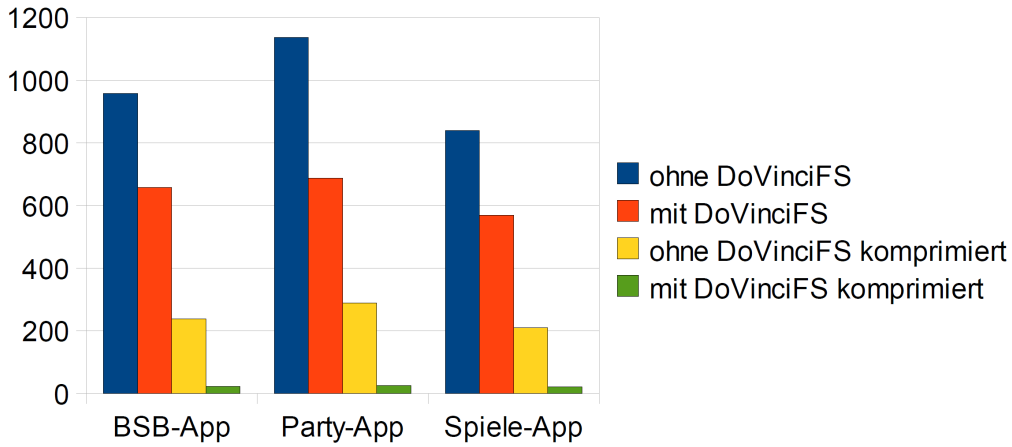

Abbildung 26: Größe der Basisimage-Datei

7.4.2 Lesezugriffsperformanz

Unter Extrembedingungen wie beispielsweise dem Kompilieren eines Linux-Kernels, was viele Zugriffe auf eine große Zahl kleiner Dateien in kurzem, zeitlichem Abstand produziert, wird der Geschwindigkeitsnachteil durch das Sharing mit DoVinciFS besonders deutlich. Die Kompilationsdauer schnellt von 6 Minuten in einer Appliance ohne DoVinciFS auf ca. 24 Minuten hoch, wenn die Appliance mit DoVinciFS versehen wurde und so sowohl Compiler, als auch Quellcode aus dem Objectstore nachgeladen werden müssen. Eine Möglichkeit,

diese Geschwindigkeitseinbuße zu reduzieren, wäre es, die vom Objectstore angeforderten Dateien zuvor innerhalb der VA in einer Ram-Disk zu cachen und so den zeitraubenden Umweg über das „Shared-Folder“ von VirtualBox zu vermeiden. Diese Idee wurde jedoch aus Zeitgründen nicht implementiert.

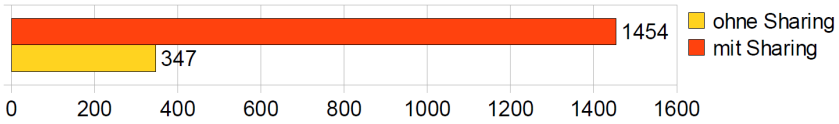

Abbildung 27: Dauer eines Kompiliervorgangs in Sekunden

7.4.3 Boot-Dauer

Die Boot-Dauer wird durch das Sharing natürlich negativ beeinflusst – die Shared Folder von VirtualBox sind vermutlich nicht auf maximale Performanz optimiert. Darüber hinaus ist auch die DoVinciFS-Implementierung zunächst an Stabilität, als an Geschwindigkeit, orientiert. Es ist zum jetzigen Zeitpunkt unvermeidlich, bei aktivem Sharing Einbußen zu verzeichnen. Diese sind jedoch noch im akzeptablen Rahmen – die Bootdauer einer VA mit und ohne Sharing unter gleichen Rahmenbedingungen ist mit Sharing etwa doppelt so lang wie ohne (69 statt 34 Sekunden).

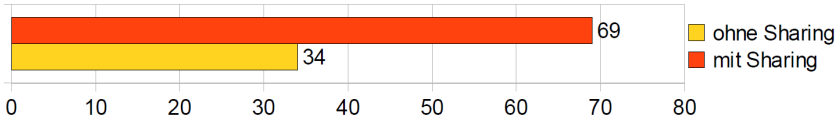

Abbildung 28: Dauer des Bootvorgangs in Sekunden

8 Client-/Publishertool

Die GUI des ersten Milestones implementiert den Betrieb von DoVinci noch allein unter Linux. Dabei wird die Virtualisierungslösung KVM verwendet. Die Migration zu VirtualBox, die Integration von Maßschneiderung und Sharing, sowie eine gefälligere optische Aufmachung sind die Hauptgesichtspunkte des nun folgenden Kapitels.

Die GUI des ersten Milestones sieht noch keine Maßschneiderung und kein Sharing vor. In Kapitel 6 ist beschrieben, wie die Maßschneiderung einer virtuellen Appliance in DoVinci durchgeführt wird. Die Konfiguration der einzelnen Tools erfordert jedoch teilweise Benutzereingaben. Auch die Frage, ob Unterstützung für Sharing eingebracht werden soll, muss in die Konfiguration der virtuellen Appliance einfließen.

In diesem Kapitel wird daher beschrieben, wie die Integration der Tools in das Client-/Publishertool erfolgt.

8.1 Anforderungen

I01 **Einfachheit** **Priorität: 2**

Die Konfiguration muss für Benutzer ohne Vorkenntnisse möglich sein.

I02 **Funktionsumfang** **Priorität: 2**

Die Konfiguration muss für Benutzer mit Vorkenntnissen alle relevanten Einstellungsmöglichkeiten bieten.

I03 **Benutzerunterscheidung** **Priorität: 2**

Die Konfiguration der virtuellen Appliances, bevor diese auf den Server hochgeladen werden, soll nur in der Rolle des Publishers möglich sein. Dabei geht es um die Ausführung der Maßschneiderungstools.

I04 **Flexibilität** **Priorität: 3**

Durch die Interaktivität mit dem Benutzer sollen mehrere unterschiedliche Konfigurationsmöglichkeiten zugelassen und unterstützt werden.

I05 **Integrität** **Priorität: 1**

Die Arbeitsweise des Clienttools soll der vorgenommenen Konfiguration entsprechen.

I06 **Integrität im Bezug auf MS1** **Priorität: 2**

Nach der Konfiguration soll das Clienttool die bereits erfüllten Anforderungen (siehe 5.6.2) weiterhin erfüllen.

8.2 Entwurf

Der Client musste im Bezug auf den MS1 hauptsächlich um Integrationsanteile erweitert werden. Im Client-/Publishertool des MS1 war noch kein Sharing und keine Maßschneiderung implementiert. Der Client musste dahingehend erweitert werden, dass die nun durch das DoVinciFS und die Maßschneiderungstools bereit gestellten Funktionen durch den Client angestoßen werden. Um die Benutzerfreundlichkeit der Client-GUI zu erhöhen, wird angestrebt, die GUI möglichst intuitiv zu gestalten. Dazu wird die Menubar durch eine Toolbar mit bunten Widgets ersetzt, neue Features, wie z.B. Appliances Starten oder Stoppen, werden durch zusätzliche Push-Buttons angeboten und die GUI durch Fortschrittsbalken erweitert. Wegen der Umstellung von KVM auf VirtualBox wird eine Schnittstellenklasse geschrieben, um das Bedienen von VirtualBox über den Client zu ermöglichen. Damit der Nutzer eine Übersicht über die vorhandenen und verfügbaren Appliances erhält, wird eine SQLite-Datenbank eingerichtet. Des Weiteren wird die bisherige .csv-Datei mit der Liste der zum Download bereitgestellten Appliance durch eine XML-Datei ersetzt.

8.3 Implementierung

Der DoVinci-Client des ersten Milestones wird erweitert, um den neuen Anforderungen gerecht zu werden und zusätzliche Funktionalitäten zu bieten. Die Migration von KVM nach VirtualBox hat an sich keine Auswirkungen auf die Struktur der DoVinci-GUI, jedoch muss eine Schnittstellenklasse geschrieben werden, um das Bedienen von VirtualBox über den Client zu ermöglichen. Neue Features wie z.B. Starten/Pausieren/Stoppen von Appliances, das Anstoßen von Maßschneiderungsprozessen über die GUI sowie der Wunsch nach einer intuitiv bedienbaren Benutzerschnittstelle mit einem gefälligeren Erscheinungsbild bringen zusätzliche Änderungen mit sich.

Eine SQLite Datenbank wird aufgesetzt, um den Zustand der Appliances und des lokalen Objectstores festzuhalten. In Kapitel 9 wird beschrieben, wie der Down- bzw. Uploadvorgang der Appliances angepasst wird, um Sharing und Maßschneiderung gerecht zu werden. Die entsprechenden Schnittstellenklassen werden clientseitig angepasst, um diesem Protokoll zu entsprechen. Des Weiteren werden die zum Download zur Verfügung gestellten Appliances in einer XML-Liste verwaltet, weshalb Schnittstellen zum Parsen von XML clientseitig implementiert werden.

8.3.1 Verbesserungen an der Benutzeroberfläche

Der Client aus MS1, welcher lediglich die rudimentäre Funktionalität und eine erste Benutzeroberfläche bereitgestellt hatte, wird im Wesentlichen optisch erheblich verändert und hat

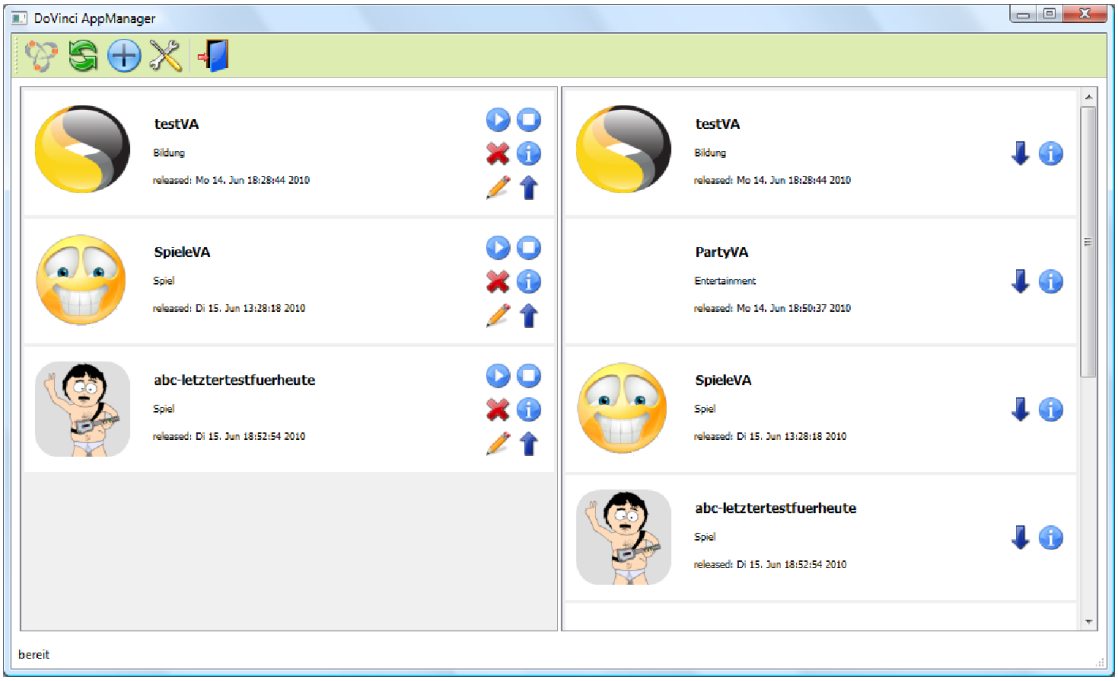

Abbildung 29: Screenshot der Client-/Publisher GUI

den Namen „DoVinci AppManager“ erhalten. Es wird Wert darauf gelegt, ihn möglichst einfach und benutzerfreundlich zu gestalten. Die farbliche Erscheinung wird mit Stylesheets gestaltet, welche von Qt unterstützt werden. In Anlehnung an den iPod von Apple gibt es keine tabellarische Ansicht der lokal vorhandenen und auf dem Server verfügbaren Appliances mit ihren zugehörigen Metadaten mehr. Stattdessen werden die VAs nun durch „AppWidgets“ dargestellt, wobei der Name, die Kategorie und das Erstellungsdatum neben dem Icon zu lesen sind. Bereits heruntergeladene VAs werden in der Serverliste nicht mehr angezeigt, wodurch eine verbesserte Übersicht für den Nutzer gegeben ist. Die drei Buttons, welche dazu dienten eine Appliance zu löschen, runter oder hoch zu laden, sind nun in die AppWidgets verlagert. Diese und weitere Funktionen, wie das Starten, Stoppen und Bearbeiten einer Appliance, sind durch Pushbuttons jeweils direkt neben den VA-Icons visualisiert. Durch Klicken des Info-Pushbuttons erscheint ein Dialogfenster mit allen, für den Benutzer relevanten Metadaten zu der VA. Für die intuitive Bedienung wird zudem die Menubar entfernt. An dieser Stelle ist nur noch die Toolbar zu sehen. In dieser Toolbar sind die Widgets für die Dienstfindung, das Aktualisieren der Serverliste, das Erstellen einer neuen VA, für Optionen, sowie für das Schließen des Client-Programms anzuklicken.

Um eine neue VA zu erzeugen wird dem Benutzer der Push-Button „Neue VA erzeugen“ angeboten. Wenn dieser gedrückt wird, wird der Benutzer zur Eingabe erforderlicher Informationen aufgefordert. Die Pfade zu der .vdi-Datei und zu einem Icon der VA werden abgefragt. Anschließend stößt der Publisher den Prozess der VA-Optimierung/Maßschneiderung

an. Danach ist die VA für den Upload fertig.

Das Hoch- oder Runterladen einer Appliance wird durch einen Fortschrittsbalken visualisiert, so dass der Nutzer den aktuellen Status des Ladevorgangs mitverfolgen kann.

8.3.2 Schnittstellenklasse für VBoxManage

Für die Virtualisierung wird Oracle VirtualBox eingesetzt. Dieses Programm bietet ein Kommandozeilentool namens VBoxManage. Der DoVinci-Client verwendet dieses Tool, um die Virtualisierungslösung zu steuern. Zu diesem Zweck existiert eine Schnittstellenklasse, welche dafür verantwortlich ist, eine Appliance bei VirtualBox anzumelden, sie zu starten, stoppen und sie abzumelden. Für VirtualBox sprechen die einfache Handhabbarkeit mittels Konsolenbefehlen, das Vorhandensein einer Open Source-Version und die Plattformunabhängigkeit. Anderen Virtualisierungslösungen vereinen diese Features nicht so konsequent.

Für den Anmeldevorgang wird eine neue virtuelle Maschine erstellt und die verschiedenen Einstellungen festgelegt, wie beispielsweise die Größe des Arbeitsspeichers. Weiterhin muss das Grundgerüst der Appliance als Festplatte eingebunden werden. Für jeden dieser Schritte wird ein eigener Prozess ausgeführt. Aufgrund der Eigenart von VBoxManage den Prozess nach dem Start der Appliance zu beenden, muss Polling verwendet werden, um herauszufinden, wann eine Appliance gestoppt wird. Nach dem Stoppen einer Appliance wird diese direkt bei VirtualBox abgemeldet, um dem Benutzer keine Reste von Appliances in seiner VirtualBox-Umgebung zu hinterlassen.

Der Client bekommt somit Methoden für alle notwendigen Schritte im Benutzungszyklus einer Appliance geliefert. Weiterhin werden nach relevanten Ereignissen, wie dem Stoppen einer Appliance, entsprechende Qt-Signale emittiert.

8.3.3 Anstoßen des Minimierungsprozesses

Für den Minimierungsprozess innerhalb des Clients werden die Methoden der Schnittstellenklasse für VBoxManage genutzt. Der Minimierungsprozess beinhaltet das Anmelden einer Appliance an VirtualBox. Das Minimierungsimage wird als erste Festplatte, das zu minimierende Image als zweite Festplatte angefügt. Nach dem Starten der Appliance wird durch Polling ermittelt, wann der Minimierungsprozess innerhalb der Appliance beendet ist. Nachträglich wird der Befehl zur Minimierung von außen ausgeführt, wobei alle 0-Sektoren aus dem Image entfernt werden.

8.3.4 SQLite-Datenbank

Zum Verwalten der Daten wurde eine SQLite-Datenbank implementiert. SQLite wurde gewählt, weil der Zugriff auf die Datenbank ohne den Server funktioniert und die Datenbank in eine lokale Datei abgespeichert wird. SQLite ist ein schnelles, wie eine Datei funktionierendes Datenbanksystem, welches nur geringe Anforderungen an den Server stellt und wenig Administrationsaufwand generiert. In der Datenbank werden zwei Tabellen verwaltet, eine für die lokalen VAs, und eine für die VAs, die nur auf dem Server verfügbar sind. Sie werden aus der Datenbank gelesen und in der GUI dargestellt. Ursprünglich wurde die

SQLite-Datenbank für die Verwaltung der Dateien des lokalen Objectstores entwickelt und sollte darüber hinaus zusätzlich die Metadaten der Appliances verwalten. Da die Verwaltung der Dateien des lokalen Objectstores jedoch anders implementiert wird, bleibt es lediglich bei der Verwaltung der oben genannten Tabellen.

8.3.5 Parsen der XML-Dateien

In der Implementierung des Client-Tools im ersten Milestone wurde vom Server eine Liste der aktuell zum Download bereitgestellten Appliances angefordert. Diese Liste der verfügbaren virtuellen Appliances wurde als .csv-Datei übertragen. Diese VA-Liste wird nun durch eine XML-Datei implementiert. Zudem ist eine Schnittstellenklasse implementiert worden, um die heruntergeladene XML-Datei in einen DOM-Baum umzuwandeln, so dass die Metadaten extrahiert und in die Datenbank abgespeichert werden können.

8.3.6 Struktur der Datenhaltung

Als Speicherort für benötigte Dateien wird je nach Betriebssystem der typische Ort für Anwendungsdaten verwendet. Unter Windows ist dies z.B. im Verzeichnis des Nutzers unter AppData/DoVinci AppManager, unter Linux typischerweise im Home-Verzeichnis des Nutzers untergebracht.

Konkret werden unterhalb des DoVinci AppManager-Ordners folgende weitere Ordner angelegt:

/objectstore
Hier befinden sich alle Dateien, aus welchen aus einem Appliance-0-Byte-Image gelinkt wird. Konkret ist das 0-Byte-Image einer Appliance nur ein leerer Baum, welcher nur auf Hashwerte für die konkreten Dateien linkt. Die zu diesen Hashwerten zugehörigen Dateien sind hier zu finden. Dabei wird jede Datei mit dem Hashwert benannt, eine Dateiendung gibt es nicht.

/db
Hier befindet sich die SQLite-Datenbank-Datei, welche den Zustand der installierten Appliances und weitere globale Konfigurationen speichert.

/minva
In minva ist ein VirtualBox-Image der OptimierungsVA zu finden. Diese Appliance kann mit einer konkreten Appliance im Anhang ausgeführt werden und stößt dann nach dem Boot-Vorgang den Optimierungsprozess an. Sobald dieser beendet ist, wird die OptimierungsVA automatisch wieder geschlossen.

/serverlists
In diesem Ordner wird pro in Reichweite befindlichem Server eine XML-Datei abgelegt, welche die jeweils angebotenen Appliances und zugehörige Meta-Daten beinhaltet.

8.4 Evaluation

Nach den vielen Änderungen am Client-/Publisher-Tool, die seit dem ersten Milestone vorgenommen wurden, wird nun eine neue Evaluation vorgenommen. Dabei wird die Benutzerfreundlichkeit untersucht, welche seit dem MS1 erheblich optimiert wurde. Darüber hinaus kann nun benannt werden, welchen Umfang das Installationspaket für den Endnutzer hat.

8.4.1 Usability

Die grafische Benutzeroberfläche des Clients ist sehr einfach gehalten. Solange keine VA heruntergeladen wurde, gibt es nur vier verschiedene Buttons, die der Benutzer anklicken kann. Einer davon, der Button „Neue VA erstellen“, ist nur dann verfügbar, wenn der Benutzer als Publisher eingeloggt ist. Dadurch kommt es zu keiner Verwirrung des Benutzers aufgrund zu vieler Optionen. Wenn nun die Schaltfläche „Serverliste aktualisieren“ angeklickt wird, sucht das Clienttool automatisch nach Netzwerken, in denen VAs angeboten werden. Diese erscheinen dann deutlich sichtbar im rechten Teil der GUI. Auch dort erscheinen dann lediglich zwei neue Buttons. Einer davon gibt dem Benutzer Informationen über die VA, der andere initiiert den Downloadvorgang. Ein Prozessbalken zeigt währenddessen den Fortschritt des Downloads.

Wenn der Benutzer als Publisher eingeloggt ist, kann er durch Drücken des Buttons „Neue VA erstellen“ eine neue VA erzeugen. Im nun erscheinenden Fenster kann der Benutzer in Textfeldern und PopUp-Menüs die Metadaten seiner VA eintragen. Zusätzlich muss er den Dateipfad zu dem Image angeben, welches er verwenden möchte. Danach wird die erzeugte VA in die Liste der heruntergeladenen VAs auf der linken Seite der GUI aufgenommen. Von dort aus kann die VA wieder per Button auf den Server hochgeladen werden. Versucht der Benutzer eine VA hochzuladen, die auf dem Server bereits verfügbar ist, dann wird die VA nicht hochgeladen, und der Benutzer bekommt eine Fehlermeldung.

Insgesamt ist das Clienttool sehr intuitiv zu bedienen und sollte auch für wenig versierte Benutzer zugänglich sein.

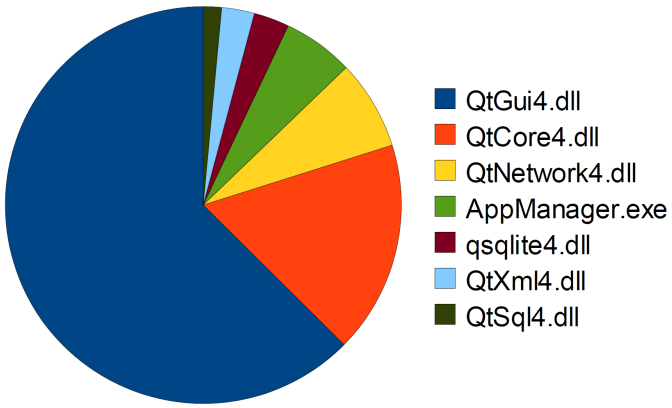

Abbildung 30: Verhältnis der Dateigrößen der ausgelieferten Dateien

8.4.2 Platzbedarf

Das Client-/Publisher Tool wird in Form einer ausführbaren Datei mit etwas weniger als einem Megabyte Platzbedarf ausgeliefert. Die plattformunabhängige Implementierung macht es jedoch erforderlich, die plattformspezifischen Qt-Bibliotheken mit auszuliefern. Das Installationspaket für Microsoft Windows erfordert so 11,4MB Speicherplatz. Komprimiert ergeben sich 5,2MB. Angenommen, der Client wird über WLAN heruntergeladen, dauert dies 1,5 Sekunden. Wahrscheinlicher ist hingegen der Download über das Internet. Bei einer DSL-Leitung mit 10MBit/s benötigt der Download 4,4 Sekunden.

9 Server

9.1 Anforderungen

In dieser Aufzählung abgehakte Anforderungen sind schon während des MS1 erfüllt worden. Ansonsten werden die Anforderungen nur noch kurz beschrieben, längere Beschreibungen sind an anderer Stelle in diesem Dokument zu finden (siehe 3.3.3).

MW1 **Dienstangebot** **Priorität: 1**

Der Server muss Dienste (VAs) bereitstellen, katalogisieren, sortieren sowie speichern und archivieren können.

MW2 **Erreichbarkeit** **Priorität: 1**

Der Server solle dauerhaft erreichbar sein und seinen Dienst anbieten können.

MW3 **Maßschneiderung** **Priorität: 1**

Der Server soll die Optimierung der Größe der zu übertragenden VA-Images und deren Updates durch die Nutzung von Maßschneiderungsverfahren unterstützen.

MW4 **Sharing** **Priorität: 1**

Der Server soll, soweit möglich, auf dem Client schon existierende Daten nicht erneut übertragen müssen. Zu diesem Zweck muss er bereits vorhandene Komponenten ermitteln, die Differenzen berechnen und ausschließlich diese übermitteln.

MW5 **VA Verwaltung** **Priorität: 2**

Die vorhandenen und angebotenen VAs sollen verwaltet werden können. Hierzu gehören insbesondere eine mögliche Lizenzierung einzelner Komponenten, Zugangsbeschränkungen sowie die nachträgliche Änderung (Hinzufügen oder Entfernen von installierter Software) einer Appliance. Des Weiteren sollen die Maßschneiderung und das Sharing konfiguriert und komplette VAs aus dem System entfernt werden können.

MW6 **Datenintegrität** **Priorität: 2**

Die Herkunft einer Appliance soll offensichtlich dargestellt werden und möglicherweise mit Hilfe von Zertifikaten signiert werden können.

MW7 **Datensicherheit** **Priorität: 2**

Die Daten innerhalb einer VA sollten entsprechend der Möglichkeiten vor Datendiebstählen geschützt werden. Hierzu gehören zum einen die Benutzerdaten des Client, zum anderen Lizenzierungsdaten von Software, welche vor dem Auslesen zu schützen sind.

MW8 **Nutzerdaten Verwaltung** **Priorität: 3**

Anfallende Benutzerdaten bei der Arbeit mit einer VA sollen an zentraler Stelle gespeichert werden können, um eine Verwendung in einer weiteren VA oder über die Lizenzdauer einer VA hinaus zu erlauben.

9.2 Entwurf

Aufbauend auf den Ergebnissen des ersten Milestones (siehe 5.2) wurden verschiedene Punkte ausgewählt um den vorhandenen Entwurf entsprechend der Anforderungen weiter zu verbessern.

Die Dateigrössen von Appliances sowohl beim Up- als auch beim Download sollen verringert werden. Dazu werden alle zu übertragenden Dateien server- wie auch clientseitig mit einem existierenden Komprimierungsverfahren vor einer Übertragung verkleinert.

Zur Beschleunigung der Ausführung einer neu heruntergeladenen Appliance durch das Client-Tool wird die Funktionalität des Prefetching eingeführt.

9.3 Implementierung

Die Komprimierung am Server beinhaltet verschiedene Aufgaben. Aufgrund der Benutzung von Sharing (siehe 7) werden nicht mehr nur virtuelle Festplatten-Images übertragen sondern auch die dazu gehörenden Objectstores.

Das 0-Byte-Grundgerüst wird nun sowohl server- als auch clientseitig nur noch im zip-Format übertragen um Speicherplatz und Netzwerkbandbreite zu sparen. Zu diesem Zweck werden diese Dateien vom Client schon gepackt an den Server übertragen und ungeändert durch den Server weiterverarbeitet.

Während im MS1 noch die gesamte Menge an Dateien übertragen wurde, muss nun nur noch die Menge der Dateien übertragen werden, welche dem Client-Tool noch nicht vorliegen. Mit Initiierung eines Download-Vorgangs durch das Client-Tool, wird im Anhang eine Liste der bereits lokal im Objectstore vorhandenen Dateien zum Server übermittelt. In Form einer Liste von Hashes teilt der Server dem Client-Tool daraufhin mit, welche Dateien das Client-Tool für die konkrete Appliance noch benötigt. Diese stellt der Server dem Client-Tool dann als eine zip-Datei mit den zugehörigen Dateien zur Verfügung. Das Client-Tool kann daraufhin diese Menge an Dateien gebündelt herunterladen.

Der Server verfügt nun über einen zentralen Objectstore, welcher, für die an diesem Server befindlichen Appliances, alle Dateien zu den zugehörigen Hashwerten bereit hält. Tech-

nisch gesehen verwaltet der Server den Objectstore genau so, wie es das Client-Tool tut. Konkret werden die Dateien zu den Hashwerten über eine zweistufige Ordnerstruktur verteilt. Dabei wird ein konkreter Hashwert auf erster Ebene in den Ordner einsortiert, welcher den Namen des ersten Buchstaben des Hashwertes hat, analog erfolgt die Einsortierung in die zweite Ebene (Auswahl durch zweiten Buchstaben des Hashwertes). Ein Hashwert abcdefg liegt also nach der Einsortierung in folgender Ordnerstruktur (ausgehend von der Wurzel des Objectstores): */a/b/abcdefg*

Prefetching Beim Prefetching lädt das Client-Tool direkt mit Download des 0-Byte-Grundgerüsts ein Paket von Dateien herunter, welches die absolut notwendigen Dateien für diese Appliance beinhaltet. Daraus resultiert, dass diese Dateien beim erstmaligen Ausführen nicht nachgeladen werden müssen, sondern im lokalen Objectstore bereits vorhanden sind.

Die Erstellung dieses Prefetching-Pakets ist im Rahmen des Upload-Vorgangs durch den Publisher eingebettet. Dabei führt der Publisher mit Hilfe des Client-Tools einen sogenannten Referenzlauf durch, welcher alle Dateien (Hashwerte) aufzeichnet, die während dieses Referenzlaufes berührt werden. Anschließend wird die Liste dieser Hashwerte im Rahmen des normalen Upload-Vorgangs an den Server übermittelt. Der Server kann diese Informationen dann für das Erstellen des jeweiligen Prefetch-Pakets im Rahmen des Download-Vorgangs durch das Client-Tool nutzen.

9.4 Evaluation

Es gelten weiterhin die Evaluationsergebnisse des ersten Milestones (siehe dazu Abschnitt 5.2.4). Allerdings hat sich die zu erwartende Datenmenge stark reduziert. Somit ist bei gleicher Übertragungsrate eine kürzere Übertragungsdauer zu erwarten.

10 Demo-VAs

Im ersten Milestone wurde zu Demonstrationszwecken eine virtuelle Appliance erstellt (siehe 5.3), die als Szenario den Übungsbetrieb der Veranstaltung Betriebssystembau unterstützen sollte. Die Änderungen, die seit dem ersten Milestone vorgenommen wurden, erfordern jedoch die Neuerstellung dieser DemoVA. Die automatisch vorgenommene Maßschneiderung, die Unterstützung von Sharing, und der Wechsel auf die Dateiformate von VirtualBox machen die primären Unterschiede aus.

Darüber hinaus werden zwei weitere Appliances vorgestellt, die mehr dem Charakter einer virtuellen Appliance entsprechen, anstelle einer isolierten Betriebssystemumgebung mit vorinstallierten Programmen.

10.1 BSB-App

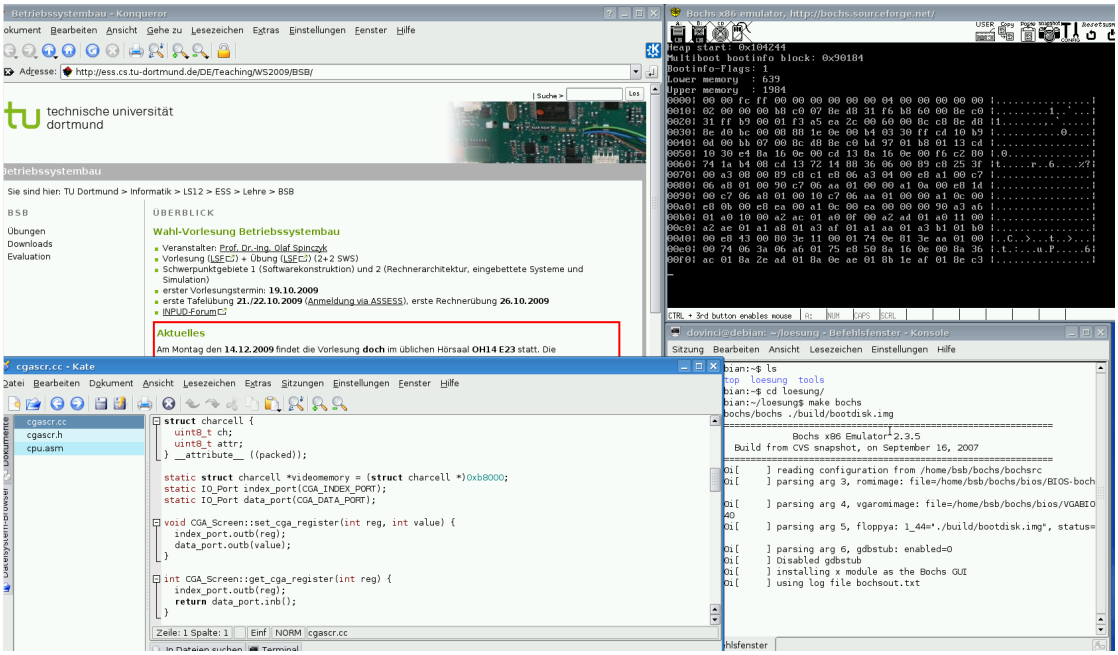

Abbildung 31: Screenshot der BSB-App

Gerade für Vorlesungen im Fachbereich Informatik werden häufig spezielle Entwicklungsumgebungen benötigt. Oft sind diese Programme nur für bestimmte Betriebssysteme vorhanden. Die Festlegung auf ein Betriebsystem erleichtert auch meist den Umgang mit den von den Studierenden erzeugten Daten. Ein solcher Fall existiert auch in der Veranstaltung *Betriebssystembau*. Der Übungsbetrieb dieser Veranstaltung verwendet Programme, die für Linux angepasst wurden. Die BSB-Appliance bietet nun alle diese Tools in einer virtualisierten Umgebung an. Somit muss ein Student sich nicht erst ein Linux installieren, sondern

kann sich stattdessen einfach die Appliance herunterladen und in dieser Umgebung arbeiten. Neben dem Vorteil des Wegfallens eines Installationsprozesses kommt diese Methode auch den Veranstaltern zu Gute, denn sie können sich bei ihren Anleitungen voll und ganz auf die von ihnen definierte Umgebung beziehen und müssen nicht auf die unterschiedlichen Eigenarten der Systeme der Studenten eingehen. Konkret besitzt die BSB-App die notwendigen Übersetzer, einen Texteditor und die Emulatorsoftware Bochs sowie die Übungsaufgaben mit Quellcode. Details siehe 5.3.1.

10.2 Party-App

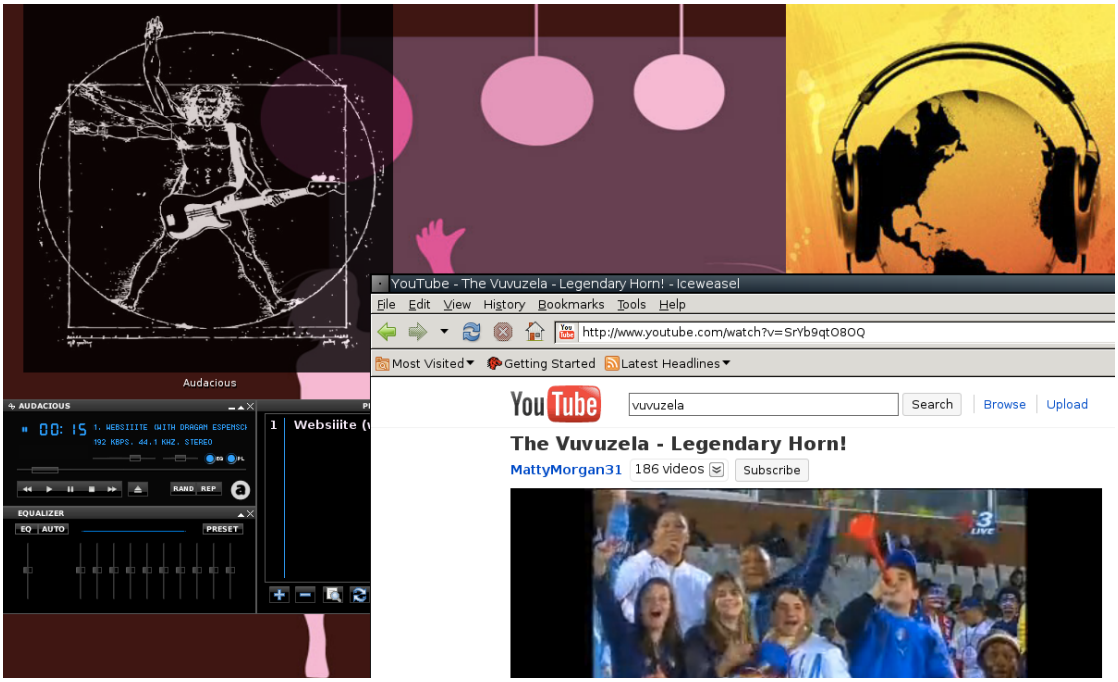

Abbildung 32: Screenshot der Party-App

Die Party-App soll einen Anwendungsfall demonstrieren, der den Vorteil der virtuellen Umgebung ausnutzt. Als Szenario wurde eine Party angenommen, auf der die Gäste selbstständig die Playlist des MP3-Abspielprogramms ändern und außerdem Webzugriff über einen Browser haben möchten. Das Internet wird verwendet, um Videos abzuspielen oder relevante Informationen wie Busfahrpläne anzuzeigen. Die Appliance besteht aus einem Debian-Linux mit dem leichtgewichtigen Window-Manager Fluxbox. Der Desktop besteht aus nur zwei Icons, einem für den Browser und einem für den MP3-Player. Der Nutzen der Appliance liegt darin, dass die Gäste nicht in den Dateien des Gastgebers herumstöbern können. Außerdem kann der Gastgeber aussuchen, welche Musikdateien zur Auswahl freigegeben werden.

Um das Ausbrechen aus der Appliance zu erschweren ist ein Herunterfahren der virtuellen Maschine nur durch Eingabe des Rootpassworts möglich. Um das Herunterfahren durch die VirtualBox-Funktionen zu verhindern, startet die Appliance im Vollbildmodus, welcher nur durch spezielle Tastenkombinationen verlassen werden kann, die vom Gastgeber zuvor festgelegt werden können.

10.3 Spiele-App

Abbildung 33: Screenshot der Spiele-App

Weitere Vorteile der Appliancelösung sind die Einfachheit der Installation sowie die Plattformunabhängigkeit. Diese Vorteile werden durch die Spiele-Appliance demonstriert. Hier startet direkt nach dem Boot-Vorgang das Spiel FrozenBubble. Die Anwendung startet direkt im Vollbildmodus. Beendet der Benutzer das Spiel, wird die Appliance wieder heruntergefahren. Somit hat der Benutzer weder etwas mit der Installation des Programms zu tun, noch muss er sich Sorgen machen, dass sein Betriebssystem inkompatibel ist. Weiterhin hinterlässt eine Appliance keine Spuren im Hostsystem. Eltern können ihre Kinder an ihren Rechnern spielen lassen, ohne hinterher Datenmüll löschen zu müssen.

11 Fazit

Folgend wird ein Fazit zu den einzelnen Teilbereichen gezogen sowie auf den Abschluss des Gesamtprojekts geblickt.

11.1 Client

Das Client-/Publishertool hat seit dem ersten Milestone (siehe 5.7) einige große Schritte vorwärts gemacht. Die Implementierung einer gefälligeren Benutzeroberfläche, die Einführung von Plattformunabhängigkeit, die Integration von Maßschneiderung und Sharing in das Bedienkonzept, sowie die Schnittstelle zum ebenfalls weiterentwickelten DoVinci-Server sind die hauptsächlichen Änderungen.

Die grafische Benutzeroberfläche ist nun mit Hilfe von CSS verwirklicht, was eine zuverlässigere Positionierung der GUI-Bestandteile auf verschiedenen Plattformen gewährleistet und die Wartbarkeit verbessert. Die virtuellen Appliances sind in der Oberfläche nun zusätzlich repräsentiert durch ein hochauflösendes Icon. Durch die konsequente Verwendung der *Qt*-Bibliotheken ist der Client nun vollständig plattformunabhängig. Der Vorgang der Maßschneiderung ist zweischrittig implementiert. Im ersten Schritt wird dem Publisher ein Softwarepaket angeboten, welches er in seiner Appliance installieren und ausführen kann. Im zweiten Schritt wird die Appliance dann in einer virtuellen Maschine noch weiter geschrumpft. Zur Sharingunterstützung wird das Dateisystem DoVinciFS installiert. Vom Client wird dann ein lokaler Objectstore verwaltet und über die *Shared Folder* von *VirtualBox* werden den Appliances die Dateien bereitgestellt. Das Nachladen fehlender Dateien vom Server, sowie der zip-komprimierte Datenverkehr mit dem Server sind nun ebenfalls Bestandteil des Client-/Publishertools.

11.2 Server

Der DoVinci-Server funktioniert einwandfrei und wie erwartet. Bereits im Fazit zum ersten Milestone (siehe 5.7) wurde erklärt, dass der Server durch die Verwendung von WebDAV-PUT sowie csv-Dateien einen geringeren Overhead erzeugt als mit dem gängigen HTTP-POST. Durch die Verwendung ressourcensparender Softwarepakete ist das Serverpaket schlank und effizient. Die konfigurationsfreie Dienstfindung wird Einbindung der plattformunabhängigen Lösung Zeroconf realisiert. Anders als zunächst angedacht findet die Maßschneiderung nicht im Server statt, sondern im Client-/Publishertool. Zusätzlich zum ersten Milestone wurde jedoch die Verwaltung von virtuellen Appliances mit Sharingunterstützung implementiert. So profitiert auch der Server von den Synergieeffekten, die durch die gemeinsame Dateinutzung entstehen. Zur Reduktion des Übertragungsvolumens werden die Dateien und die Basisimages virtueller Appliances nun komprimiert übertragen.

11.3 Demo-Apps

Die Demo-Appliance aus dem ersten Milestone (siehe 5.7) ist nun nicht mehr per Hand, sondern automatisiert maßgeschneidert. Darüber hinaus unterstützt sie Sharing und kann so

gegebenenfalls in erheblich kürzerer Zeit übertragen werden. Zusätzlich sind zwei weitere Apps implementiert worden. Die PartyApp demonstriert einen Anwendungsfall, in dem das eigentliche Host-Betriebssystem zugunsten der virtuellen Maschine verdeckt wird. Es ist in dieser App nicht möglich, in das Host-Betriebssystem auszubrechen, so wird der Rechner des Gastgebers vor Zugriffen durch Partygäste geschützt. Die FrozenBubble-App demonstriert ganz speziell den Charakter einer virtuellen Appliance. Sie bootet und startet direkt das Spiel FrozenBubble im Vollbildmodus. Wird das Spiel verlassen, fährt die Maschine direkt herunter und wird geschlossen. Die dennoch vorhandene Desktopumgebung wird völlig maskiert.

11.4 Maßschneiderung

Die Maßschneiderung in DoVinci ist in drei Tools aufgeteilt. Da eine vollständige Automatisierung sich als schwer umsetzbar gezeigt hat, ist die Maßschneiderungslösung von DoVinci nur semi-automatisch. Die Tools zur generischen und benutzergeführten Maßschneiderung werden dem Benutzer als Installationspaket bereitgestellt. Er muss sie selbst installieren und ausführen. Die Bereinigung des Dateisystems und die anschließende Komprimierung werden von außen durch eine OptimierungsVA durchgeführt. Der Erfolg einer automatischen Maßscheiderung hängt stark von der zu maßschneidernden Appliance ab. Es konnten jedoch generell rund 25% Speicherplatzbedarf eingespart werden, wobei der Aufwand für den Benutzer sehr gering ist. Leider ist das Tool zur benutzergeführten Maßschneiderung nur für Debian-Linux und dessen Derivate geeignet.

11.5 Sharing

Die Sharingunterstützung in DoVinci ist sehr erfolgreich in der Verkleinerung eines zu übertragenden Basisimages. In Versuchen konnte ein unkomprimiertes Image von 665MB um etwa 97% auf komprimierte 24MB reduziert werden. Dabei wurden viele der enthaltenen Dateien in den Objectstore ausgelagert, damit diese von anderen Appliances gemeinsam genutzt werden können. Diese gemeinsame Nutzung beträgt in Versuchen etwa 25% der Dateien bei zwei Appliances, etwa 35% bei drei gemeinsam nutzenden Appliances. Da bei der Umsetzung zunächst die Funktion und weniger die Performanz des Verfahrens im Vordergrund stand, ist Sharing in DoVinci mit Einbußen bei der Dateizugriffsgeschwindigkeit verbunden. Bei Verwendung von Sharing verdoppelt sich in etwa die Bootdauer. Aufgaben mit sehr vielen Dateizugriffen dauerten in Versuchen siebenmal länger. Als Ursache für diesen Einbruch wird die Verwendung der „Shared Folder“ von VirtualBox vermutet. Diese sind nicht auf Performanz hin implementiert und wurden nur der Einfachheit halber in der Sharinglösung von DoVinci verwendet.

11.6 Appstore für den Campus

Die zentrale Idee, die im Laufe der Projektgruppe entstand, trug den Titel „Appstore für den Campus“. Mobile Endgeräte sollten mit Apps versorgt werden, die konfigurationsfrei auf beliebigen Geräten laufen, sofern die Hardwareanforderungen erfüllt sind. Diese Idee

bleibt jedoch eine Idee, da die wichtigste Voraussetzung, eine Virtualisierungslösung für eingebettete Systeme, noch nicht existiert. DoVinci konnte allerdings zeigen, dass skalierendes Provisioning von virtuellen Appliances über WLAN auf dem Campus möglich ist. Und zwar durch Integration von Lösungen zur konfigurationsfreien Dienstfindung, plattformunabhängigen Bibliotheken, Maßschneiderung und Sharing. Der Test der Idee „Appstore für den Campus“ wurde zwar nur auf x86-Hardware durchgeführt, war aber in hohem Maße erfolgreich. Es konnte gezeigt werden, dass die Idee umsetzbar ist, wenn in Zukunft eine Virtualisierungslösung für eingebettete Systeme bereitsteht.

11.7 Gesamtprojekt

Alles in allem kann zum Abschluss des Projekts auf eine ganzheitliche und umfassende Lösung geblickt werden. Die in den vorangegangenen Abschnitten beschriebenen, wichtigen Komponenten sind mindestens auf Prototyp-Level implementiert und funktionieren.

Im Abschnitt 1.5 werden die Ziele genannt die im Projektgruppenantrag definiert wurden und die im Verlauf der Projektgruppe erreicht werden sollten. Es ist festzustellen, dass alle Ziele erreicht wurden. Zunächst wurden vorhandene Virtualisierungslösungen evaluiert und getestet im Hinblick auf ihre Fähigkeiten für das Provisioning von VAs. VirtualBox wurde ausgewählt und an den DoVinci-Client angebunden. Durch die konfigurationsfreie Infrastruktur, die auch den Betrieb von mehr als einem DoVinci-Server erlaubt, ist auch das zweite Ziel erreicht. Das dritte Ziel betrifft die Entwicklung von Maßschneiderungslösungen für virtuelle Maschinen. Auch diese wurden erfolgreich entwickelt und somit das Ziel erreicht.

Die ebenfalls im Projektgruppenantrag definierten, sogenannten Minimalziele waren zum größten Teil schon nach der ersten Hälfte der Projektgruppe erreicht. Zum Abschluss sind alle Minimalziele erreicht. MZ1 betrifft die konfigurationsfreie Dienstlokalisierung, welche in DoVinci durch die Nutzung von Zeroconf gegeben ist. Als Virtualisierungslösung wurde für das Minimalziel 2 VirtualBox gewählt. MZ2a ist durch die Anbindung an den DoVinci-Client gelöst, MZ2b ist gelöst durch die Wahl von VirtualBox welches die gemeinsame Nutzung von Hardwareressourcen ermöglicht. Das dritte Minimalziel, Provisioning, ist als gelöst anzusehen, durch die Entwicklung des DoVinci-Servers und dessen Zusammenspiel mit dem DoVinci-Client durch die konfigurationsfreie Dienstfindung. Auch die geforderten Beispielapplikationen sind erbracht worden. Das vierte Minimalziel sieht die Dokumentation und Umsetzung von Maßschneiderungsideen vor, auch diese wurden erbracht. Von den Minimalzielen nicht gefordert wurde darüber hinaus Sharing implementiert.

Werden die im Projektgruppenantrag gestellten Minimalziele sowie die selbstgestellten Anforderungen an die Teilkomponenten als Maßstab für den Erfolg der Projektgruppe herangezogen, so ist ein hervorragendes Fazit über das Gesamtprojekt zu ziehen.

Neben der rein technischen Umsetzung kann darüber hinaus die Teilnahme am Campusfest 2010 als voller Erfolg bezeichnet werden. Hier wurden eine Reihe von Kontakten geknüpft sowie erste Interessenten seitens der Professoren für DoVinci gewonnen.

11.8 Checkliste Anforderungen

Es wurden 65% der zu Projektbeginn an die Teilkomponenten gestellten Anforderungen erfüllt. Von den Anforderungen die höchste Priorität erhalten haben sind es jedoch bereits 86%. Es folgt die Einzelaufstellung der Anforderungen, aufgeschlüsselt nach den Bereichen für die sie gestellt wurden.

11.8.1 Anforderungen an den Server

MW1 Dienstangebot Priorität: 1 $\surd$
MW2 Erreichbarkeit Priorität: 1 $\surd$
MW3 auf Publisher-Tool übergegangen
MW4 Sharing Priorität: 1 $\surd$
MW5 VA-Verwaltung Priorität: 2 $\times$
MW6 Datenintegrität Priorität: 2 $\surd$
MW7 Datensicherheit Priorität: 2 $\times$
MW8 Nutzerdaten-Verwaltung Priorität: 3 $\times$

11.8.2 Anforderungen an den Client

C01 Dienstfindung Priorität: 1 $\surd$
C02 Integration der Virtualisierungslösung Priorität: 1 $\surd$
C03 VA-Verwaltung Priorität: 1 $\surd$
C04 Zeitbeschränkte Laufzeit von VAs Priorität: 1 $\times$
C05 VA-Update Priorität: 1 $\times$
C06 Sharing Priorität: 1 $\surd$
C07 Authentifizierung von Benutzern Priorität: 1 $\surd$
C08 Nutzerdaten lokal speichern Priorität: 1 $\surd$
C09 Nutzerdaten zentral speichern Priorität: 3 $\times$
C10 Benutzerinterface Priorität: 3 $\surd$
C11 Plattformunabhängigkeit Priorität: 3 $\surd$
C12 USB-Boot Priorität: 3 $\times$

11.8.3 Anforderungen an das Publisher-Tool

P01 VA-Verwaltung Priorität: 1 $\surd$
P02 VA-Konfiguration Priorität: 1 $\surd$
P03 Performanz Priorität: 1 $\surd$
P04 Kosten Priorität: 1 $\surd$
P05 Usability Priorität: 1 $\surd$
P06 VA-update Priorität: 1 $\times$
P07 VA-Zusammenstellung Priorität: 2 $\times$
P08 Plattformunabhängigkeit Tool Priorität: 2 $\surd$
P09 Plattformunabhängigkeit VA Priorität: 2 $\times$

P10 Zeitliche Begrenzung Priorität: 2 $\times$
P11 Kapselung Priorität: 2 $\surd$
P12 Automatisches Update Priorität: 2 $\times$
P13 Zielgruppe Priorität: 3 $\times$
P14 VA Statistiken Priorität: 3 $\times$

11.8.4 Anforderungen an die VA

A01 Universeller Einsatz Priorität: 1 $\surd$
A02 Peripheriezugriff Priorität: 1 $\surd$
A03 Performanz, Stabilität und Ressourcennutzung Priorität: 1 $\surd$
A04 Isolation Priorität: 1 $\surd$
A05 Systemverträglichkeit Priorität: 1 $\surd$
A06 Zugriffsschutz Priorität: 2 $\times$
A07 safety & security Priorität: 2 $\times$
A08 Fehlerbenachrichtigung Priorität: 3 $\times$

11.8.5 Anforderungen an die Maßschneiderung

M01 Wirksamkeit der Maßschneiderung, Priorität: 2 $\surd$
M02 Automatisierung und Nutzerunterstützung, Priorität: 1 $\surd$
M03 Dauer der Maßschneiderung, Priorität: 1 $\surd$
M04 Unveränderte Funktionalität, Priorität: 1 $\surd$

11.8.6 Anforderungen an das Sharing

S01 Differenz 1 $\surd$
S02 Performanzeinbußen 2 $\times$
S03 Platzoverhead 2 $\surd$
S04 Transparenz 1 $\surd$
S05 Nutzerdaten 1 $\surd$
S06 Paralleler Dateizugriff 1 $\surd$

12 Ausblick

In diesem Gesamtausblick nach dem Projektgruppenende werden Themen angesprochen, in denen nachweislich nicht die optimale Lösung gefunden oder implementiert wurde. DoVinci hatte das Ziel, ein verwendungsfähiges Produkt zu liefern, daher wurden einige Ansätze, für die bisher keine gut funktionierende Teilimplementierung existiert, außer Acht gelassen. Diese Ansätze werden hier im Folgenden dargestellt.

12.1 Virtualisierung

Es existiert noch immer keine Virtualisierungslösung für eingebettete Systeme, die es erlaubt, in dem vorhandenen Betriebssystem virtuelle Maschinen zu starten. Xen ist für ARM-Prozessoren verfügbar, muss jedoch vom Gerätehersteller installiert werden. Die Firma VMware entwickelt derzeit eine Virtualisierungslösung für ARM-Prozessoren, diese ist allerdings noch nicht öffentlich verfügbar. Dies macht es zum gegenwärtigen Zeitpunkt unmöglich virtuelle Appliances für Mobilgeräte bereitzustellen. Es gibt bereits Mobiltelefone mit Xen als Virtualisierungslösung, um das Betriebssystem von Systemfunktionen zu trennen, jedoch wurde darauf geachtet, dass die Virtualisierung für den Nutzer nicht zugreifbar ist.

Erst wenn Virtualisierung für Mobilgeräte auch für Endnutzer verfügbar ist, ist die Voraussetzung für virtuelle Appliances auf mobilen Geräten geschaffen. Dann können auch Appliances für Mobilgeräte bereitgestellt werden, was nach den Erfahrungen bezüglich DoVinci ein sinnvolles Konzept darstellt. Dahin gehende Pläne sind jedoch bisher nicht bekannt.

12.2 Apps für den Campus

Seit der Einführung des Begriffs „App“ durch Apple ist der Gedanke an konfigurationsfrei zu betreibende on-demand verfügbare Anwendungen in aller Munde. Leider sind die Apple-Apps nicht kompatibel mit den verbreiteten Betriebssystemen für x86-Maschinen. Die Apps werden vornehmlich für den Betrieb auf dem iPhone entwickelt, insofern handelt es sich bei diesen um eine Insellösung. Um auf dem Campus die große Masse der Endnutzer erreichen zu können, wäre eine „package once, deploy everywhere“-Lösung wünschenswert. Es müssten verschiedene Smartphone-Betriebssysteme, Linux, Windows sowie MacOS unterstützt werden. Dafür sind virtuelle Appliances ein gangbarer Weg, der jedoch im Moment an der Verbreitung von Virtualisierung auf mobilen Endgeräten scheitert.

12.3 Maßschneiderung

DoVinci betreibt Maßschneiderung auf Paket- und Dateiebene. Weitere Maßschneiderungserfolge versprechen die Block-, Funktions- und Code-Ebene, welche jedoch nicht betrachtet wurden, da zum einen die automatisierte Maßschneiderung auf Funktions- und Codeebene derzeit noch Forschungsgegenstand ist und bisweilen nicht produktreif. Zum anderen liefe eine solche Mikrooptimierung konträr zu dem Ziel des Sharing. Da beispielsweise die C-Library nach Reduktion auf die wirklich verwendeten Funktionen in zwei unterschiedlichen

VAs mit hoher Wahrscheinlichkeit nicht mehr in beiden VAs identisch wäre und so der Sharingfaktor zwischen diesen zwei VAs reduziert würde. Eine Optimierung die aufgefallen ist, aber nicht implementiert wurde, ist das automatische Entfernen von Kommentaren, Leerzeichen und Tabs aus XML-Dateien, ohne die Semantik zu verändern.

12.4 Sharing

DoVinci bietet Sharing auf Dateiebene an. Alternativ wäre auch Sharing auf Blockebene möglich gewesen. Blockbasiertes Sharing ist feingranularer, jedoch ist der Verwaltungsoverhead potentiell höher, insbesondere wenn man sehr kleine Blockgrößen wählt um einen möglichst hohen Sharingfaktor zu erzielen. Blockbasiertes Sharing hat zudem starke Abhängigkeiten vom verwendeten Dateisystem der VA, beispielsweise durch *tail packing*, welches die Enden mehrerer Dateien in den gleichen Block schreibt oder alleine schon durch die Reihenfolge in der Dateien in das Dateisystem geschrieben werden, welches ihnen unterschiedliche Blocknummern zuweisen kann und somit zu unterschiedlichen Inhalten in den Dateisystem-Verwaltungsblöcken führt. Andererseits hätte blockbasiertes Sharing zusätzliches Potential. Beispielsweise könnten zwei Dateien die sich lediglich durch ein zusätzlich eingefügtes Leerzeichen unterscheiden den gemeinsamen Teil vor der Änderung sharen.

Literatur

[BDB+08] BRAKENSIEK, Jörg ; DRÖGE, Axel ; BOTTECK, Martin ; HÄRTIG, Hermann ; LACKORZYNSKI, Adam: Virtualization as an enabler for security in mobile devices. In: *IIES '08: Proceedings of the 1st workshop on isolation and integration in embedded systems*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–126–2, S. 17–22

[BDF+03] BARHAM, Paul ; DRAGOVIC, Boris ; FRASER, Keir ; HAND, Steven ; HARRIS, Tim ; HO, Alex ; NEUGEBAUER, Rolf ; PRATT, Ian ; WARFIELD, Andrew: Xen and the art of virtualization. In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA : ACM, 2003. – ISBN 1–58113–757–5, S. 164–177

[EGT03] EVI, Nemeth ; GARTH, Snyder ; TRENT, Hein: *Handbuch zur Linux Sytemverwaltung*. Munich : München, Markt-+-Technikverlag, 2003. – ISBN 3–82726442–1

[EN09] ENGEL, Michael ; NOLTE, Jörg: IIES '09: Proceedings of the 2nd Workshop on isolation and integration in Embedded Systems. New York, NY, USA : ACM, 2009. – ISBN 978–1–60558–464–5

[ES08] ENGEL, Michael ; SPINCZYK, Olaf: IIES '08: Proceedings of the 1st workshop on isolation and integration in embedded systems. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–126–2

[Hei08] HEISER, Gernot: The role of virtualization in embedded systems. In: *IIES '08: Proceedings of the 1st workshop on isolation and integration in embedded systems*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–126–2, S. 11–16

[HSH+08] HWANG, Joo-Young ; SUH, Sang-Bum ; HEO, Sung-Kwan ; PARK, Chan-Ju ; RYU, Jae-Min ; PARK, Seong-Yeol ; KIM, Chul-Ryun: Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones. In: *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, 2008. – ISSN 0197–2618, S. 257–261

[Jer07] JEREMY FÜR KERNELTRAP.ORG: Interview: Avi Kivity. `http://kerneltrap.org/node/8088`, 23. Apr. 2007

[KKL+07] KIVITY, Avi ; KAMAY, Yaniv ; LAOR, Dor ; LUBLIN, Uri ; LIGUORI, Anthony: KVM: The Linux Virtual Machine Monitor. In: *Proceedings of the Linux Symposium*, 2007, S. 225–230

[LHSP+09] LOHMANN, Daniel ; HOFER, Wanja ; SCHRÖDER-PREIKSCHAT, Wolfgang ; STREICHER, Jochen ; SPINCZYK, Olaf: CiAO: An Aspect-Oriented Operating-System Family for Resource-Constrained Embedded Systems. In: *Proceedings of the 2009 USENIX Annual Technical Conference (USENIX '09)*. San Diego, CA, USA, June 2009, S. 215–228

[Lun07] LUNTOVSKYY, Andriy: Protocol stack and capacity modeling for WLAN / TU Dresden. 2007. – Forschungsbericht

[Mar07] MARWEDEL, Peter: *Eingebettete Systeme*. Berlin [u.a.] : Springer, 2007 (eXamen.press). – ISBN 978–3–540–34048–5 – 3–540–34048–3. – Embedded system design <dt.>

[PG74] POPEK, Gerald J. ; GOLDBERG, Robert P.: Formal requirements for virtualizable third generation architectures. In: *Commun. ACM* 17 (1974), Nr. 7, S. 412–421. http://dx.doi.org/http://doi.acm.org/10.1145/361011.361073. – DOI http://doi.acm.org/10.1145/361011.361073. – ISSN 0001–0782

[RSS+08] ROSENMÜLLER, Marko ; SIEGMUND, Norbert ; SCHIRMEIER, Horst ; SINCERO, Julio ; APEL, Sven ; LEICH, Thomas ; SPINCZYK, Olaf ; SAAKE, Gunter: FAME-DBMS: tailor-made data management solutions for embedded systems. In: *SETMDM '08: Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management*. New York, NY, USA : ACM, 2008. – ISBN 978–1–59593–964–7, S. 1–6

[Sei06a] SEIFERT, Josef W.: *Projektmanagement für kleinere Projekte*. Offenbach : Gabal, 2006. – ISBN 3–8974–9655–0

[Sei06b] SEIWERT, Lothar: *Das neue 1 x 1 des Zeitmanagement*. Munich : Gräfe und Unzer, 2006. – ISBN 3–7742–5670–5

[SLB07] STOESS, Jan ; LANG, Christian ; BELLOSA, Frank: Energy management for hypervisor-based virtual machines. In: *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*. Berkeley, CA, USA : USENIX Association, 2007. – ISBN 999–8888–77–6, S. 1–14

[Su08] SU, Disheng: *Mini VM - Extending KVM towards Embedded Systems*. http://www.linux-kvm.org/wiki/images/3/38/KvmForum2008$kdf2008_20.pdf. Version: 2008. – speaker at virtualization mini summit, Ottawa, 2008

[WR07] WARNKE, Robert ; RITZAU, Thomas: *QEMU virtuelle Computer für viele Betriebssysteme; QEMU Version 0.9.0. (German) [QEMU virtual computer for many operating systems]*. Norderstedt : Books on Demand GmbH, 2007. – 292 S. http://d-nb.info/986260371/04. – ISBN 3–8370–0876–2