PG 544: DoVinci

Dortmund Virtualized Networked Campus Infrastructure

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

Zwischenbericht

17. Juni 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
LSXII
Prof. Dr.-Ing. Olaf Spinczyk
Arbeitsgruppe Eingebettete Systemsoftware

Die Projektgruppe 544: DoVinci entwickelt ein Campus-Infrastruktur-System mit dem sich Personen auf dem Campus Anwendungen herunterladen können und diese auf ihrem (mobilen) Endgerät ausführen können. Diese Anwendungen werden in Form von Appliances (Betriebssystem + Anwendung) bereitgestellt, was dazu führt das Installation und Wartung von der Rolle des Benutzers entkoppelt werden. Stattdessen erzeugt das beim Nutzer vorhandene Betriebssystem eine virtuelle Maschine, 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 Sharing (Vermeidung von Redundanz) und Maßschneiderung (Beschränkung auf Wesentliches).

Um diese Ziele zu erreichen werden im Rahmen des Projekts 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 werden die Betreuer und Mitglieder der Projektgruppe aufgeführt und die Motivation des Projekts sowie dessen wissenschaftlicher Kontext beleuchtet. Außerdem werden die sich daraus resultierenden Ziele angeführt und die Minimalziele erklärt, welche im Projektgruppenantrag vorgegebenen sind.

1.1 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

1.2 Motivation

Software für Mobilgeräte (wie Web-Tabletts oder Mobiltelefone) leidet derzeit 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 einge-

schränkt sind. Damit gestaltet sich die Entwicklung offener und flexibler Anwendungen für Mobilgeräte schwierig, da die Portierung von Software auf eine andere Plattform ein kosten- 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 werden 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 Maschine (Virtual Appliance, VA) implementiert und an das jeweilige mobile Endgerät ausgeliefert. Diese VA beinhaltet das Betriebssystem, benötigte Bibliotheken und Middleware sowie die eigentliche Applikation und deren Daten. Dadurch hat der Nutzer den Vorteil, dass er von jeder Installations- und Konfigurationsarbeit befreit ist, indem er eine vordefinierte Umgebung bereitgestellt bekommt.

1.3 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 u.a. auch Thema des von der Arbeitsgruppe ausgerichteten IIES-Workshops ist [ES08, EN09]. Die Ressourcenbeschränktheit von mobilen eingebetteten Systemen stellt neuartige Anforderungen an Virtualisierungsinfrastrukturen, die in dieser Form bisher nicht existierten [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, zu einer deutlichen Reduktion der Größe der virtuellen Maschinen beitragen. Maßschneiderungstechniken werden von der betreuenden Arbeitsgruppe bereits erfolgreich im Bereich der Betriebssysteme [Dan09] und Datenhaltungssysteme [RSS+08] eingesetzt.

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

1.4 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 [Avi07, 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 wurden 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 [Avi07]

- 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

1.5 Aufbau dieses Dokumentes

Auf diese Einleitung folgt im Abschnitt 2 eine Bestandsaufnahme von Techniken, die im Vorfeld des Projekts von den Teilnehmern untersucht wurden. Im Abschnitt 3 wird die Aufgabenstellung in Form von Szenarien analysiert, aus denen sich unmittelbar die Anforderungen an die Einzelkomponenten ergeben. Die während der Entwicklung untersuchten Technologien werden im Abschnitt 4 aufgeführt. Der Abschnitt 5 beschreibt die Entwicklung des ersten Prototypen und schließt mit einem Fazit, in dem die zuvor gestellten Anforderungen mit den erreichten Zielen abgeglichen werden. Der letzte Abschnitt 6 ist ein Ausblick auf die weitere Projektentwicklung und den Fokus der zweiten Projekthälfte. Im Anhang (ab Seite 65) finden sich schließlich Verweise auf verwendete Literatur.

2 Seminarphase

Zu Beginn wurden die für dieses Projekt notwendigen und dem Stand der Technik entsprechenden Technologien im Rahmen eines gemeinsamen Seminars erarbeitet. Hierbei wurden insbesondere jene Technologien beleuchtet, welche für DoVinci von besonderer Relevanz sind. Zusätzlich wurden die Themen Zeit- und Projektmanagement als Grundlage einer reibungslosen Projektabwicklung vorgestellt. Diese Phase diente dazu die Projektteilnehmer auf einen gemeinsamen Wissensstand zu bringen. Kurzfassungen der vorgestellten 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 derzeit verschie-

dene 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 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. Das x86 Modell 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[Pre84].

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 [Pre84]. Dies birgt Vorteile sowohl auf Nutzer- als auch auf Publisherseite. Für DoVinci ist sowohl die Sicht ins Innere als auch das Management von VAs zentral. Hier gilt es grundlegende Probleme in beiden Bereichen zu lösen. Zu den wichtigsten zählen:

Für die VA und die ausführende Virtualisierungslösung: Maßschneiderung vs. Sharing, inkrementelle Updates, Nutzerdaten im Kontext von VAs, Parametrisierung und zeitlich begrenzte Ausführung.

Für das Management und die Auslieferung der VAs: Dienstfindung via WLAN, effiziente Auslieferung, Authentifizierung für Nutzer und Publisher, zeitlich begrenzte Auslieferung, Updates verbreiten und Einhaltung von Datensicherheit/Datenschutz.

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. JeOS (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 enthalten.

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 [Pre84]. 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 Konflikte 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 [Pre84]. 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 [Pre84]. Der Erfolg von Zeitmanagement ist stark abhängig von der Einstellungen und vom konsequenten Verhalten. 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 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 [Pre84].

2.11 Zeroconf

Zeroconf[1] 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. Besonders interessant für DoVinci ist das zur Dienstlokalisierung verwendete Domain Name System Service Discovery Protokoll[2]. 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

[1] `http://www.zeroconf.org`
[2] `http://www.dns-sd.org`

schon vorhanden, welche eine Implementation in neuen Projekten noch weiter vereinfachen.

3 Analyse der Aufgabenstellung

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

3.1 Anwendungsfallanalyse

Die erste Analyse des zu entwickelnden Systems wird in Form einer Szenarienbeschreibung realisiert. Es werden hierzu relevante Situationen durchdacht und in den Kontext des zu entwickelnden Systems gesetzt. Diese Situationen werden als Szenarien festgehalten und die darin enthaltenen Anwendungsfälle in Diagramme überführt.

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 „Virtual 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, wodurch Prüfungsanmeldungen oder Bücherausleihe erheblich vereinfacht werden. Damit die Integrität des Host Systems gewahrt wird, ist die virtuelle Hardware, auf der die Appliance betrieben wird, vom Host-Systems isoliert.

Student Die Rolle Student agiert in diesem Szenario als Client und somit Nutzer der Appliance. Er kann die Appliance in ihrer maßgeschneiderten Form (angepasst auf sein spezielles System) 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 kann Zugriff auf die UniCard(-Daten) des Studenten haben und diese werden in die Appliance weitergeleitet. Somit 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 Erstl-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. Der Publisher hat 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.

Anwendungsfalldiagramm: siehe Abbildung 1

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. Allerdings handelt es sich weniger um einen Zugriffsschutz, vielmehr um ein Verfahren um Kunden nicht versehentlich mit veralteten Daten zu belästigen.

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-Hauses dient er als Navigationssystem und hilft die Waren zu finden, außerhalb fungiert er als Katalog, Einrichtungsplaner und Einkaufszettel.

Anwendungsfalldiagramm: siehe 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 virtuali-

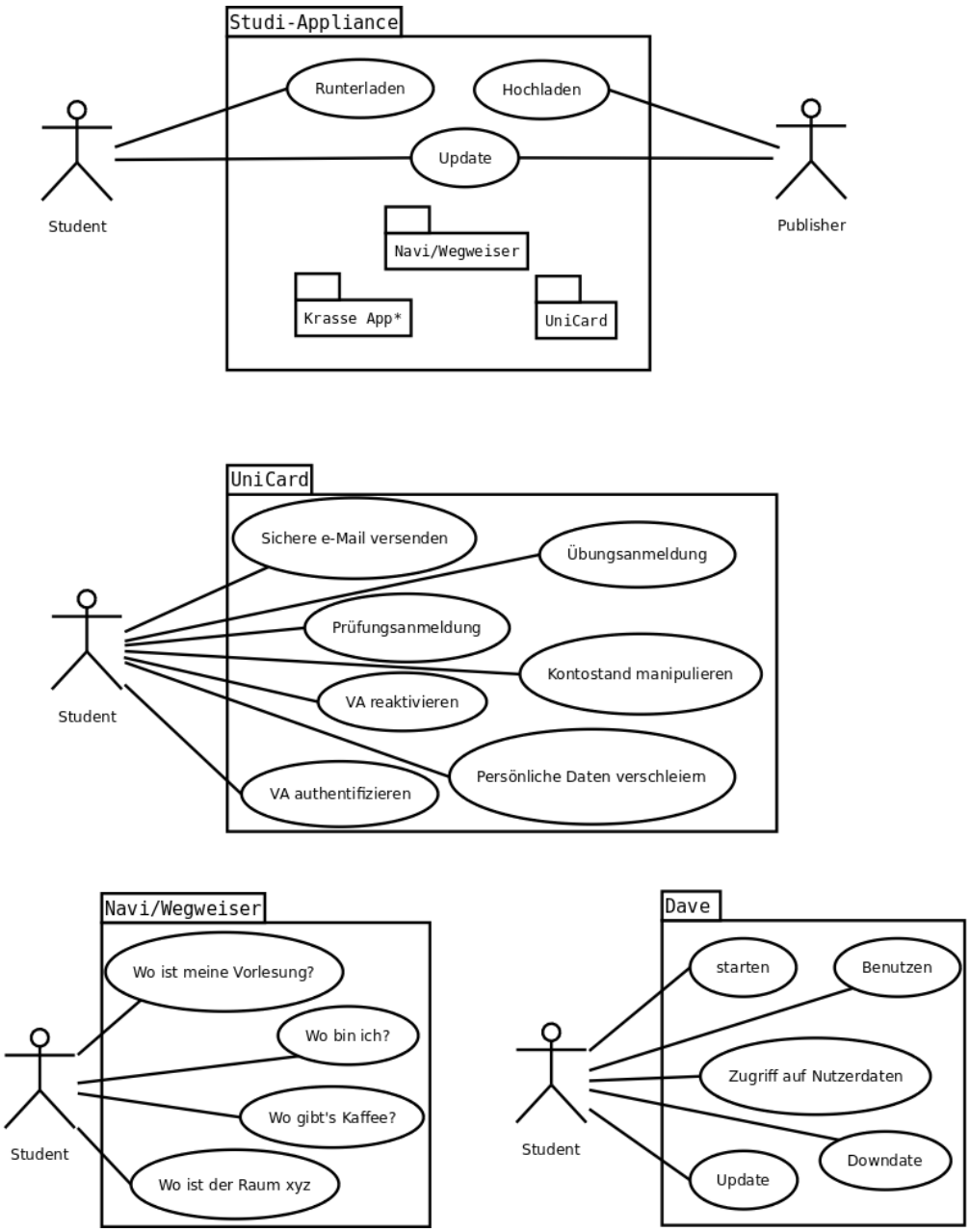

Abbildung 1: Anwendungsfalldiagramm der Studi-Appliance 3.1.1

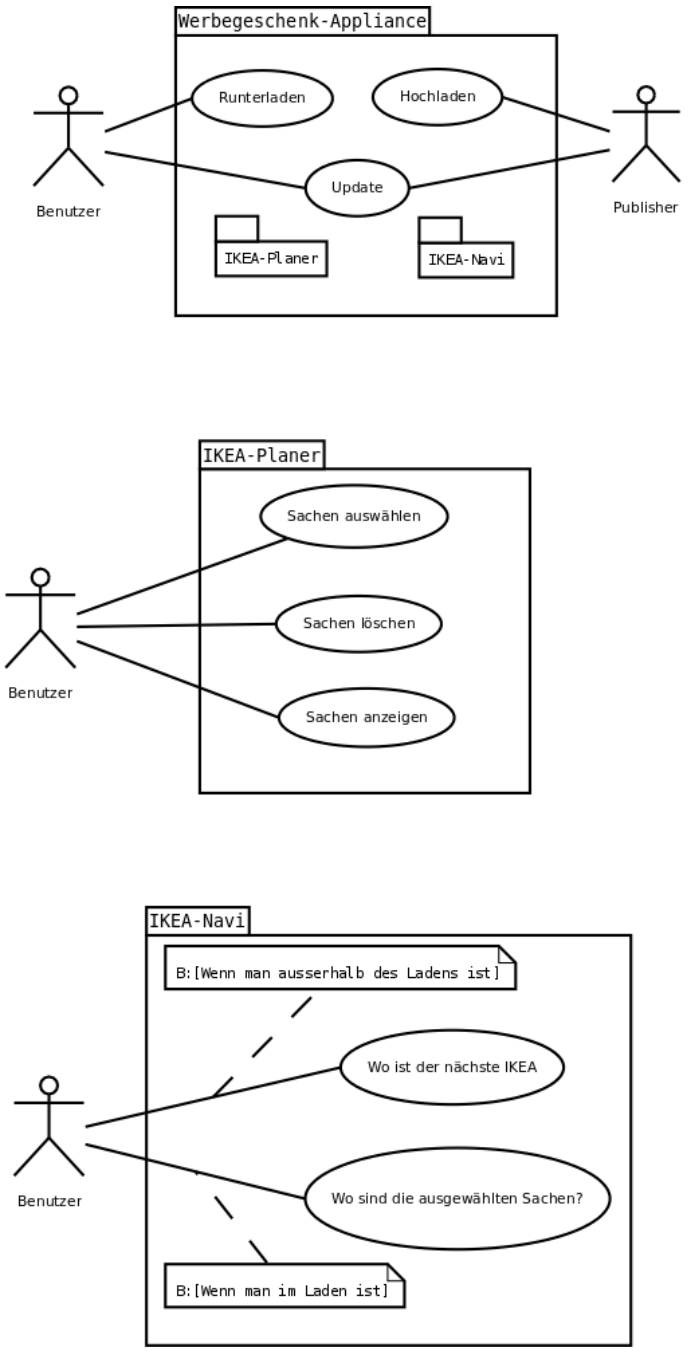

Abbildung 2: Anwendungsfalldiagramm der Werbe-Appliance 3.1.2

sierte 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. Auch die Gründung eines Campus-Web2.0-Netzwerks wird so erleichtert.

Anwendungsfalldiagramm: siehe Abbildung 3

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. 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 und wo die VA verfügbar ist, einstellbar.

Anwendungsfalldiagramm: siehe Abbildung 4

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

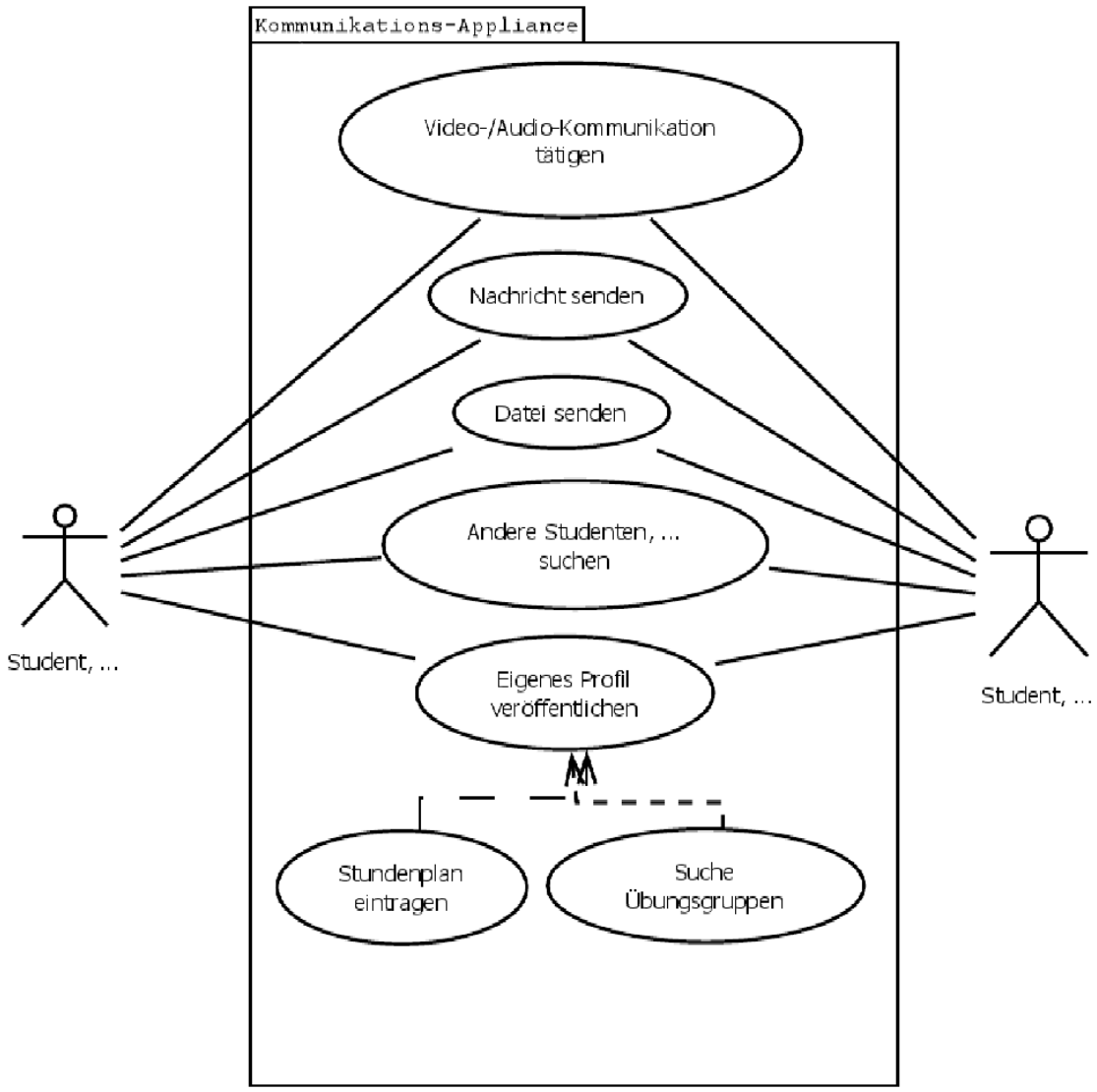

Abbildung 3: Anwendungsfalldiagramm der Kommunikations-Appliance 3.1.3

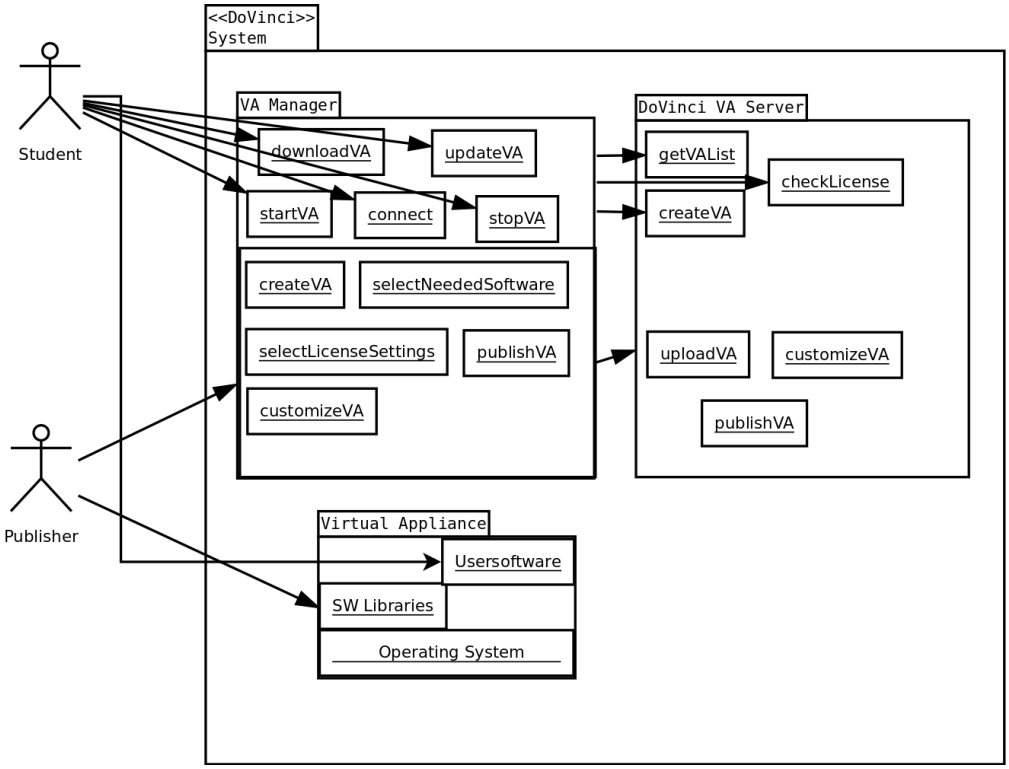

Abbildung 4: Anwendungsfalldiagramm der Vorlesungs-Appliance 3.1.4

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.

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.

Anwendungsfalldiagramm: siehe Abbildung 5

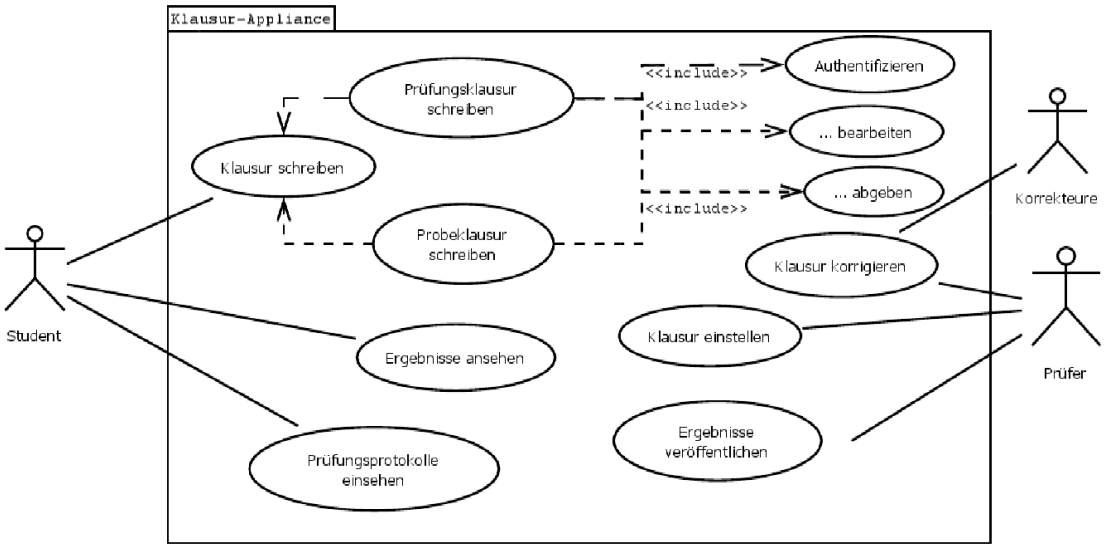

Abbildung 5: Anwendungsfalldiagramm der Klausur-Appliance 3.1.5

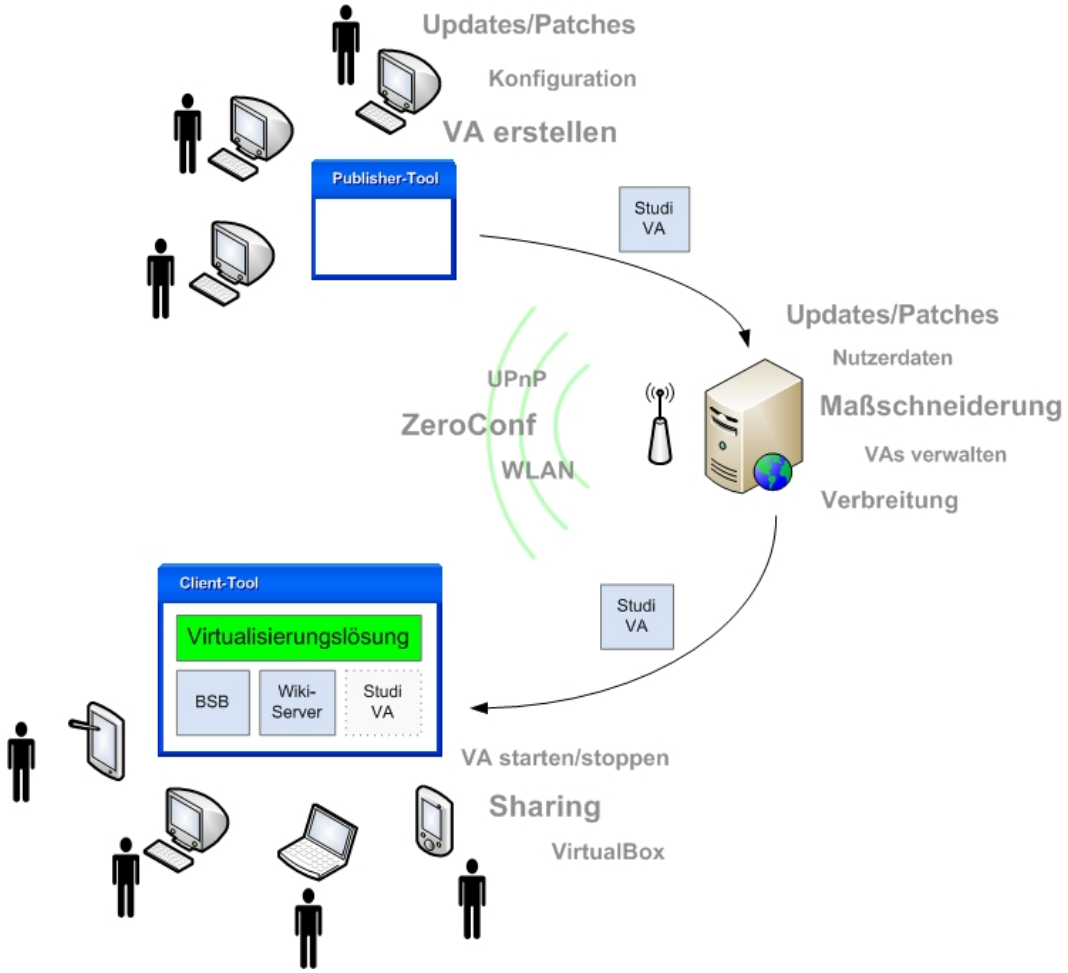

Abbildung 6: Grobentwurf des zu entwickelnden Systems

3.2 Anforderungsanalyse

In diesem Abschnitt werden die Anforderungen an die einzelnen Systemkomponenten sowie an die Appliances mit Bezug auf die Anwendungsszenarien beschrieben.

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).

3.2.1 Anforderungen an den Client

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

Die Anwendungsszenarien setzen voraus, dass der Client automatisch die aktuell verfügbaren DoVinci-Server im Netzwerk findet und die dort verfügbaren Appliances auflistet. (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. (3.1.4)

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

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

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

Die Anwendungsszenarien erfordern, dass der Client Appliances mit zeitlich beschränkter Laufzeit unterstützt. Dies kann z.B. im Falle lizenzierter Komponenten innerhalb einer VA auftreten. Im Falle einer Zeitbeschränkung sollte der Benutzer rechtzeitig darauf hingewiesen werden. (3.1.2, 3.1.4, 3.1.5)

C05 **VA Updates** **Priorität: 1**

Alle Anwendungsszenarien setzen voraus, dass der Client das Updaten von VAs unterstützt. 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. (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. (3.1.1,3.1.2, 3.1.3, 3.1.4, 3.1.5)

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

Die Anwendungsszenarien setzen voraus, dass der Client die Authentifizierung von Benutzern am Server unterstützt, gegebenenfalls in verschiedenen Varianten (z.B. per Passwort, Benutzername + Passwort oder Lizenzschlüssel). (3.1.1, 3.1.4)

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

Für die Anwendungsszenarien ist es 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. (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. (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.

3.2.2 Anforderungen an das Publisher-Tool

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

Das Publisher-Tool soll in der Lage sein mehrere VAs verwalten zu können. (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. (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. (3.1.1, 3.1.4)

P04 **Kosten** **Priorität: 1**

Es sollen dem Publisher durch die Nutzung der DoVinci Infrastruktur keine Kosten entstehen. (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. (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. (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. (3.1.3, 3.1.4, 3.1.5)

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

Das Publisher-Tool soll für mehrere Plattformen verfügbar sein. (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. (3.1.4)

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

Es soll möglich sein, die Laufzeit einer VA bei den Benutzern zeitlich einzuschränken. (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. (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). (3.1.3, 3.1.4)

3.2.3 Anforderungen an den Server

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

Der Server muss Dienste (VAs) bereitstellen, katalogisieren, sortieren sowie speichern und archivieren können. (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. (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. (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. (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. (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. (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. (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. (3.1.1, 3.1.3, 3.1.4, 3.1.5)

3.2.4 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.

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. (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.

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.

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)

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

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

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

Die VA soll Fehler protokollieren und diese an den VA-Server übertragen können.

4 Grundlegende 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 [Pre84]. 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 Virtualisierung (full virtualization) 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 [Pre84]. Mittlerweile gibt es eine lauffähige Lösung des Hypervisors Xen auf ARM, welche auf der ARM 9 Architektur getestet ist [3]. 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.

[3] `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 [Pre84]. Für das Projekt 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 [4]. 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 ein em 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.

[4] `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 [Pre84]. 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 das Projekt 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 das Projekt 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, das für den Transport per WLAN noch abgelastet 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.

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 *Win-*

dows Vista mit ca. 6GB gegenüber, allerdings kommen noch Auslagerungs- und Ruhezustandsdateien hinzu, welche etwa 1,5GB zusätzlichen Speicherplatz verbrauchen [5][6]. Es gibt zu *Windows 7* noch kein Tool wie *nlite* und *vlite*.

Windows XP Das älteste unter den modernen Windows, aber auch das kleinste (Grundinstallation 1,5GB + Auslagerungsdatei), unter welchem 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 [7] 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 Appliancebetriebssystem 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. Bei einem Versuch mit der Oberfläche Gnome ist diese jedoch unangenehm aufgefallen. In der Distribution Ubuntu wird mit Gnome ein Paket zwingend mitinstalliert, welches für den Download und die Darstellung von Online-Wetterdaten verantwortlich ist. Diese Abhängigkeit bringt 20MB unnötige Festplattenbelegung mit sich und ist unumgehbar, es sei denn die Paketabhängigkeiten werden verändert.

DamnSmallLinux (DSL) *DSL* ist eine weitere 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 liefern [8]. Die Distribution enthält Webbrowser, E-Mail Client, Tabellenkalkulation, MP3-

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

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 im Projekt 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 [9]. 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.

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 an-

[9] `http://news.netcraft.com/archives/web_server_survey.html`

dere 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 [10]. 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 das Projekt 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 (wovon eine beachtliche Menge bei DoVinci zu erwarten ist) ungemein und soll es ermöglichen die Anforderungen an den DoVinci-Server mit Standardhardware zu erfüllen.

4.4.3 Andere Webserver-Software

Es gibt eine große Zahl an anderen Webservern die aber wegen ihrer geringen Verbreitung und praktischer Unsichtbarkeit im Netz nicht weiter berücksichtigt werden.

4.4.4 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, Phyton, OpenGL und SQL. Es hinterlässt zudem nur einen kleinen Speicherabdruck auf eingebetteten Systemen. Des Weiteren bietet die Firma Nokia eine sehr gute

[10] `http://www.lighttpd.net/story`

Online Reference Documentation [11]. Somit ist Qt geeignet im weiteren Verlauf des Projekts DoVinci eingesetzt zu werden.

[11] `http://doc.trolltech.com/`

5 Entwicklung eines ersten Prototypen MS1

Der erste Milestone 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 3.2.1 und die Anforderungen an das Publisher-Tool 3.2.2 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 runtergeladen 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. Qt ist eine C++-Klassenbibliothek für die plattformübergreifende Programmierung grafischer Benutzeroberflächen, die unter anderem für Linux, Windows und MacOS X verfügbar ist und bietet neben den Klassen für die grafische Benutzeroberfläche auch Möglichkeiten zur systemunabhängigen Programmierung von Netzwerkschnittstellen und anderen für den Client wichtigen Funktionen.

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 sowie das Löschen lokal vorhandener VAs. Zwei weitere Buttons unterhalb der Liste der

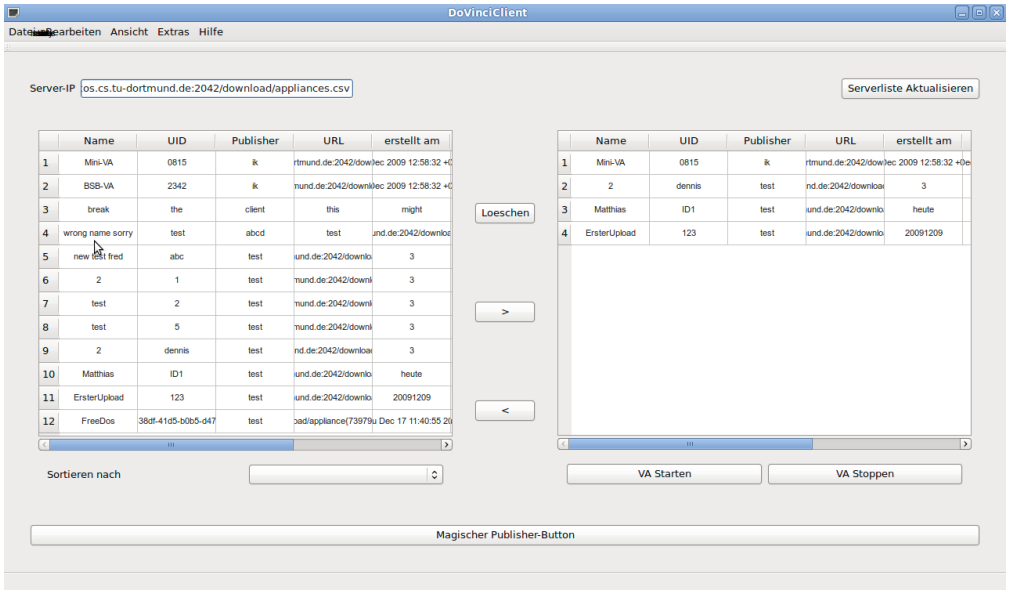

Abbildung 7: GUI des Client-Tools

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.

5.1.4 Evaluation

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.

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 vorgesehene 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 mitgesendet 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.

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.

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. Die Implementierung des Clients und die verwendeten Bibliotheken bieten hoffentlich ein gutes Grundgerüst für die Erweiterbarkeit im Hinblick auf den MS2. 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.

5.2.1 Anforderungen

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 wurden:

Abbildung 8: Schaubild des Server Entwurfs

Dienstangebot Die Liste der Virtual Appliances wird dem Client in Form eines CSV-Dokuments (Character Separated Values) via HTTPS über einen Lighttpd Webserver bereitgestellt. Die vorhergehende Dienstfindung kann optional per Zeroconf-Protokoll über den Nameserver BIND erfolgen, diese Möglichkeit wird jedoch im MS 1 nicht vom Client benutzt.

Weitergehend wird die Möglichkeit des Uploads neuer und der Aktualisierung vorhandener Appliances durch einen Publisher zur Verfügung gestellt. Dabei teilt sich der Upload-Prozess in zwei Schritte:

1. WebDAV-Upload der Image-Datei einer Appliance
2. Übertragung der Informationen der zuvor gesendeten Image-Datei an ein CGI/Perl-Script (siehe VA Verwaltung für eine genaue Parameterbeschreibung)

Erreichbarkeit Der zentrale Server ist ein stationärer Rechner mit angebundenem WLAN-Accesspoint. Beide befinden sich dauerhaft in Betrieb.

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

VA Verwaltung Die Informationen zu den Virtual Appliances werden intern im CSV-Format gespeichert. Folgende Informationen werden pro Virtual Appliance abgelegt: Eine eindeutige Kennung (ID), der Name der Appliance, das Erstellungsdatum, der Zeitpunkt des letzten Updates, der Name des Autors, die Beschreibung der Appliance sowie ein Verweis auf die Image-Datei. Diese Daten werden abgesichert durch eine MD5-Prüfsumme zur Kontrolle der korrekten Übertragung.

Die Appliances selbst werden im Binärformat als Image-Dateien abgelegt. In der CSV-Datei wird dann jeweils auf diese Image-Dateien verlinkt.

Datenintegrität Der Upload neuer Appliances wird ausschließlich autorisierten Publishern angeboten. Die Zugangsdaten werden vom Serverbetreiber vergeben. Technisch wird dies via htaccess-Schutz direkt im Webserver im Rahmen des Upload-Prozesses umgesetzt.

Datensicherheit Die Datenübertragung zwischen Client und Server wird über eine SSL-verschlüsselte HTTP-Verbindung (HTTPS) abgewickelt.

In diesem Prototyp gelten außer den Zugangsdaten der Publisher alle Daten als öffentlich. Durch die SSL-Verschlüsselung ist die sichere Übertragung der Zugangsdaten während des Logins gewährleistet. Auf Serverseite sind sie in einer htpasswd-Datei abgelegt, welche die Passwörter einwegverschlüsselt speichert.

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 dieses ersten Prototypen ist, dass der Prototyp schnellstmöglich mit Funktionen zur Verfügung stehen soll, 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*) genutzt zur Realisierung der Dienstfindungs-Anforderungen. Die Verwendung von Zeroconf ist weniger kompliziert als die Alternative des Simple Service Discovery Protokolls aus UPnP, was sich mit den Ergebnissen der Vorarbeiten deckt [Pre84]. 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 Web-Server Software für den Server, muss eine Reihe von Anforderungen erfüllen und bestimmte Eigenschaften aufweisen. Es soll die Möglichkeit geben Module zu benutzen, eine gute Anzahl an schon vorhandenen und getesteten Modulen sollen verfügbar sein, sowie eine Implementierung optimierter Daten-

zugriffsverfahren für den Up- und Download. 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 von zusätzlichen zugeschnittenen CGI-Skripten erfüllt.

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
  - 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 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 Scripts 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 Scripts den Metainformationen zugeordnet werden kann.

Für den zweiten Schritt wird das Perl-Script 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 Script 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 [Pre84]. 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 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[L$^+$07]. 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

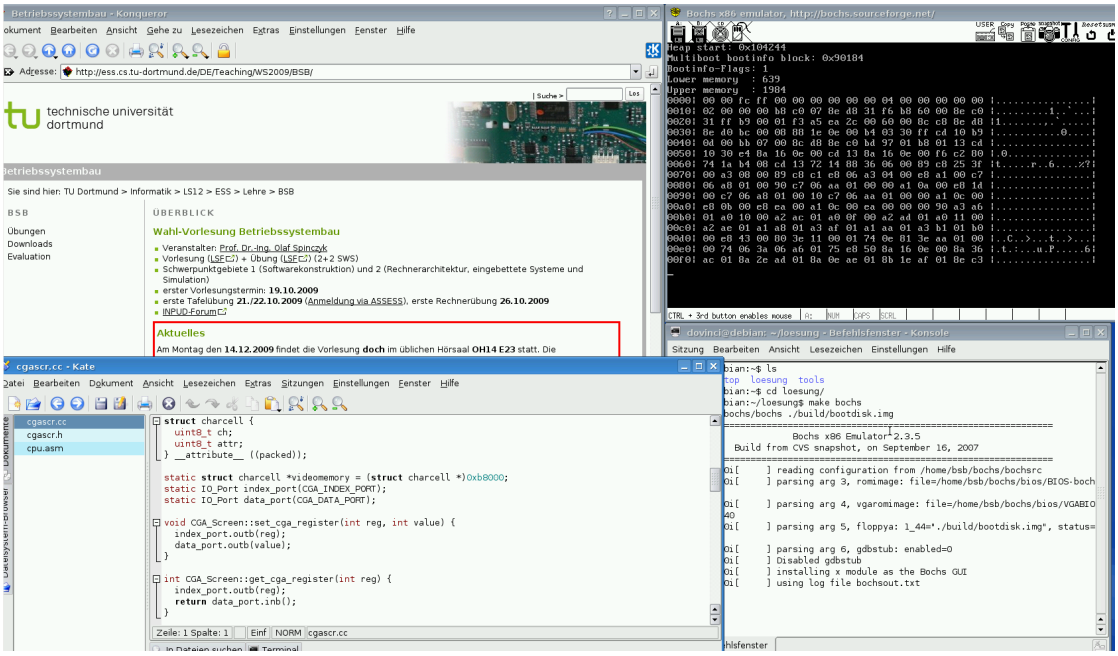

Abbildung 9: Screenshot der BSB-Appliance

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 wohl 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

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 eines paravirtualisierten Treibers zur Leistungssteigerung von 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 Treiberunterstützung 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 erzeugt ein komprimiertes Image von *190MB*. Dieses lässt sich noch ablasten auf etwa *104MB*. Eine minimale Installation des auf Debian basierenden Ubuntu der Version hardy mit Kernel der Version 2.4 erzeugt nach ablasten ein Image mit *115MB*. 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. DamnSmallLinux (DSL) beispielsweise erfordert nur *61MB*, arbeitet hierzu allerdings mit dem weniger performanten 2.4 Kernel, 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.

Tatsächlich verwendete Daten 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.

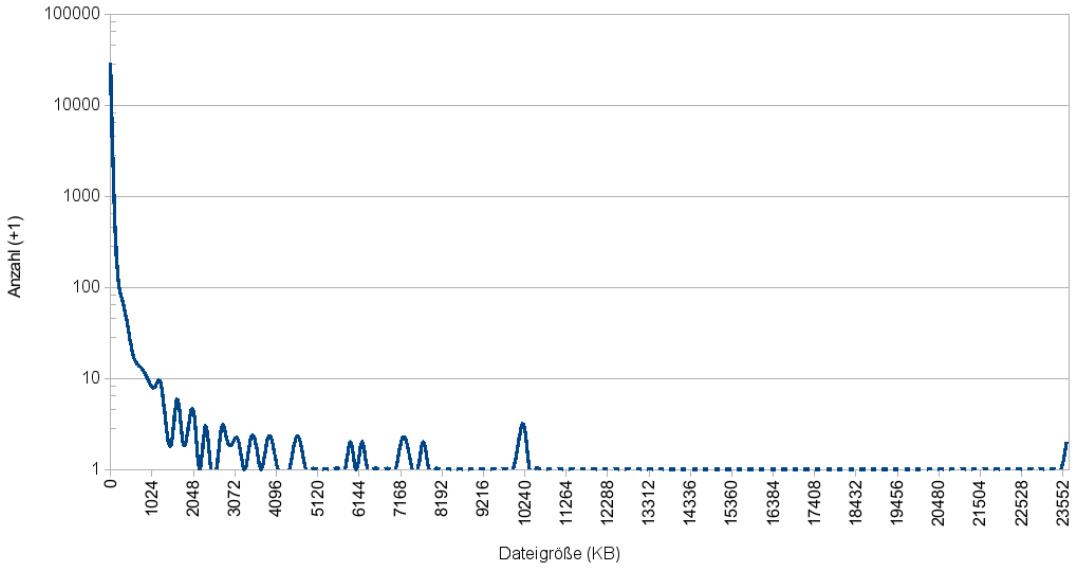

Abbildung 10: Größenverteilung der Dateien der BSB-VA

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.

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.

Das Paketmanagement, ein Mittel zur Maßschneiderung, kann im Rahmen einer Maß-

schneiderung 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 fast 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 Paketmanagements 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 auch 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 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.

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.

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

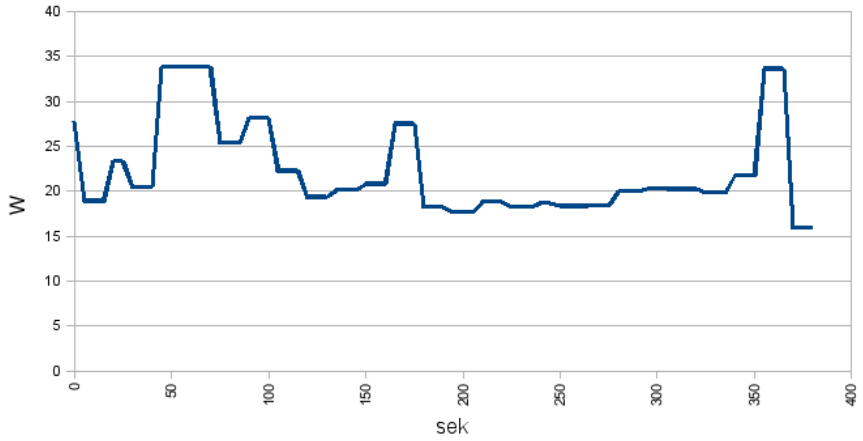

Abbildung 11: Einzelner Testlauf mit Boot → SuperPi → Bonnie++ → Shutdown

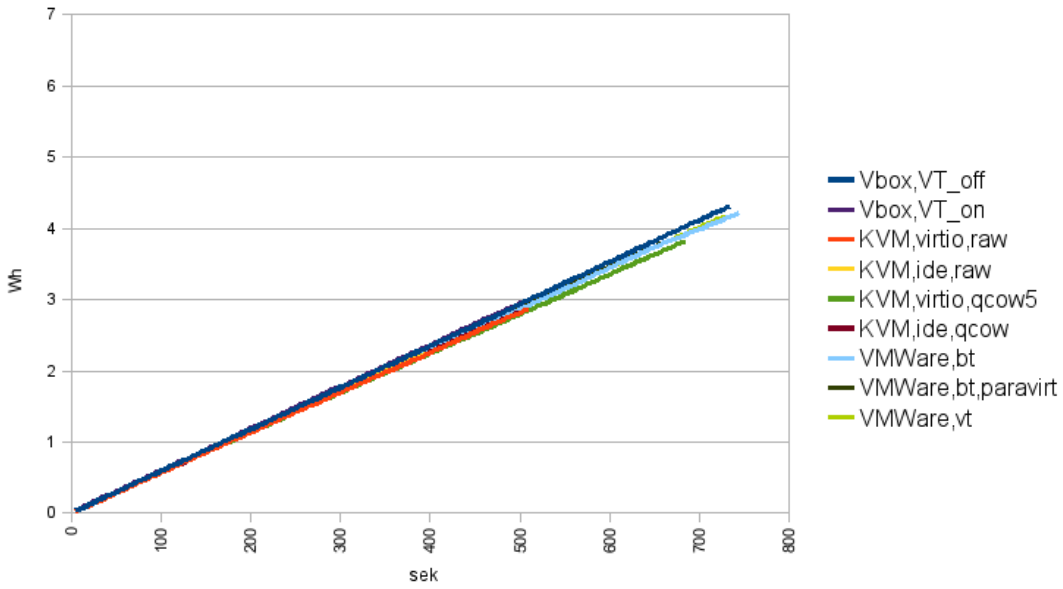

Abbildung 12: Leistungsentnahme der verschiedenen Konfigurationen bei festen 800MHz

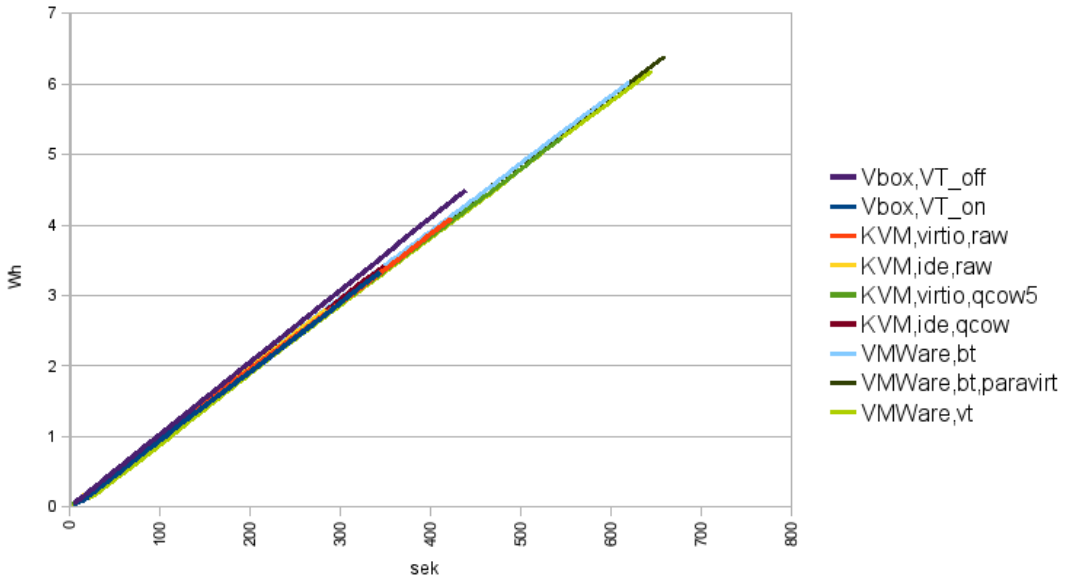

Abbildung 13: Leistungsentnahme der verschiedenen Konfigurationen bei festen 2000MHz

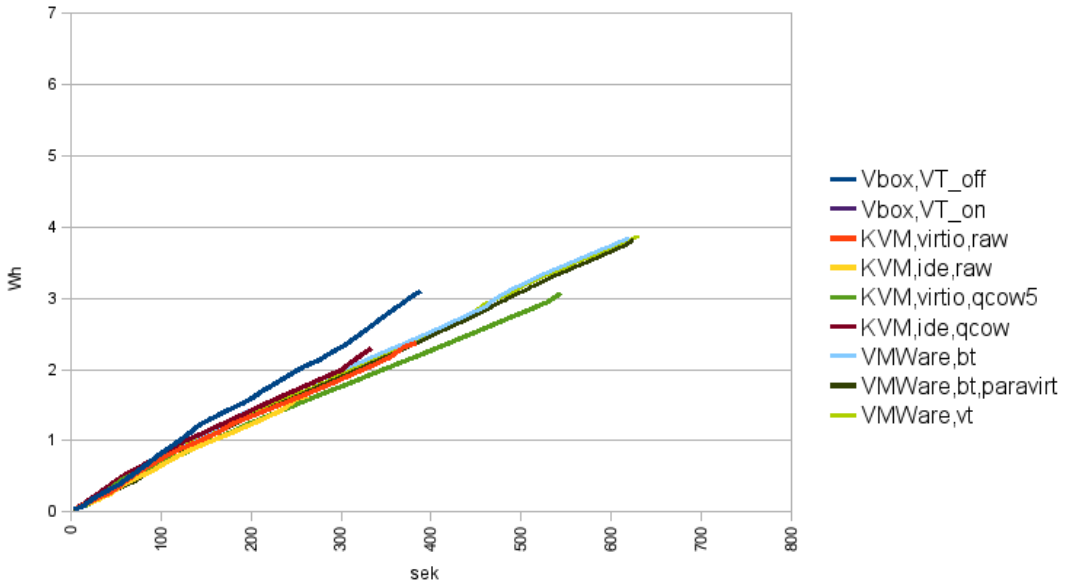

Abbildung 14: Leistungsentnahme der verschiedenen Konfigurationen bei variabler CPU-Frequenz

stabil waren, lässt sich durch aufaddieren (Multiplikator $\frac{5}{3600}$) ein Wert in Wattstunden angeben.

In einem exemplarischen Testlauf (Abbildung 11) 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 12) respektive 2000MHz (Abbildung 13) CPU-Frequenz erzeugt. Jedoch dominiert die CPU-Nutzung die Leistungsentnahme, so dass keine Unterschiede abzulesen sind.

Bei variabler CPU-Frequenz zwischen 800MHz und 2000MHz (Abbildung 14) 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. Sollte bis zur Vergleichsmessung nach Sharing und Maßschneiderung der Bug behoben sein, so werden die Ergebnisse im späteren Endbericht nachgeführt.

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.

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 15 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 16 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 der von SuperPi genutzten arithmeti-

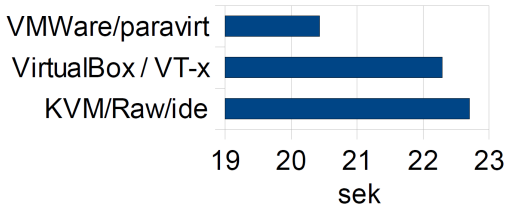

Abbildung 15: Beste Ergebnisse aller Virtualisierungslösungen

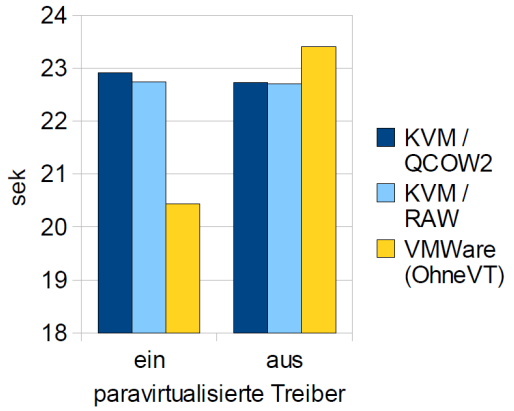

Abbildung 16: Performanzvergleich bei Betrieb mit Hardware-Virtualisierungsunterstützung

schen Operationen. 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 15) 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 eine Minimalkonfiguration erzeugt mittels *make allnoconfig* und anschließend der Kernel 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.

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 18). Dieses Verhalten lässt sich eventuell durch Cachingeffekte erklären, hervorgerufen durch die kleine virtuelle

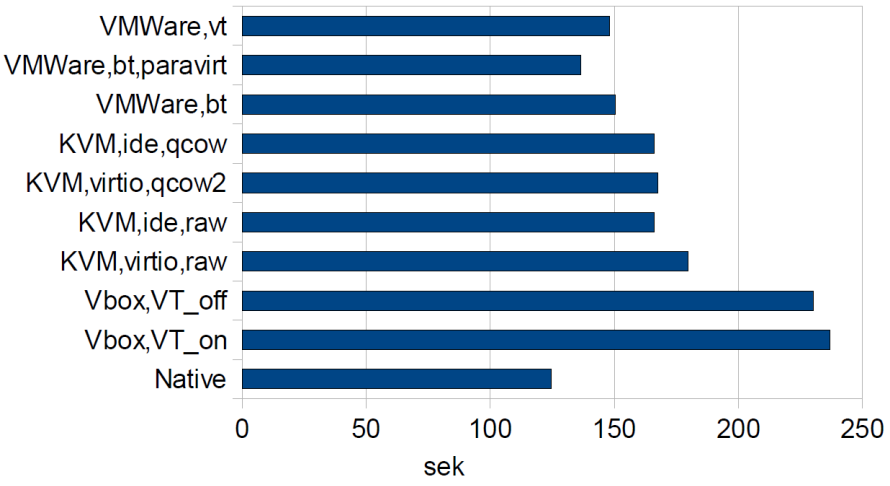

Abbildung 17: Kompilieren eines Kernels, Vergleich aller getesteten Konfigurationen

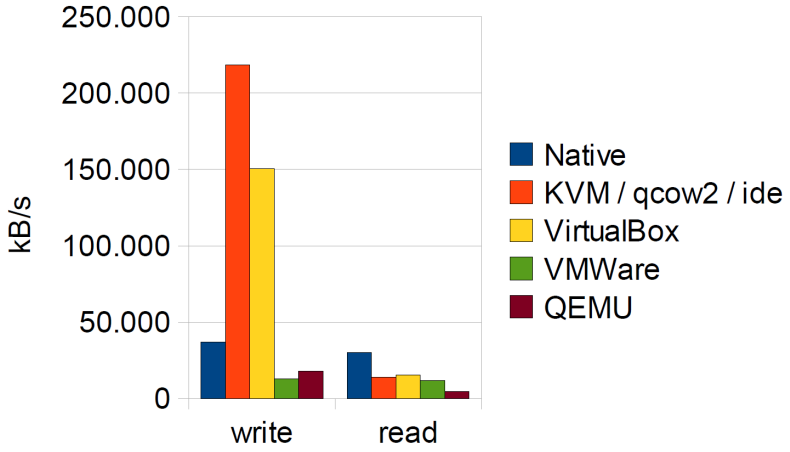

Abbildung 18: Bonnie++ Ergebnisvergleich aller Virtualisierungslösungen mit dem nativen System

Abbildung 19: Bonnie++ Vergleich der unterschiedlichen Imageformate von KVM

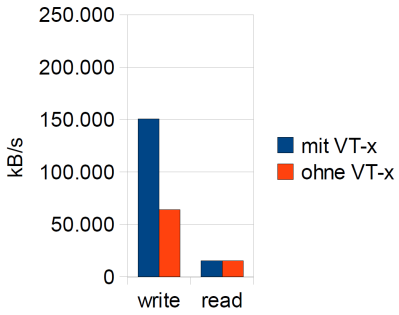

Abbildung 20: Bonnie++ in VirtualBox mit und ohne VT-x

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 Performanzunterschiede ergeben, werden die Durchsätze unter den verschiedenen Imageformaten gemessen und in Abbildung 19 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 20), so ergeben sich für den Schreibdurchsatz unter VirtualBox starke Leistungseinbußen.

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 21) 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

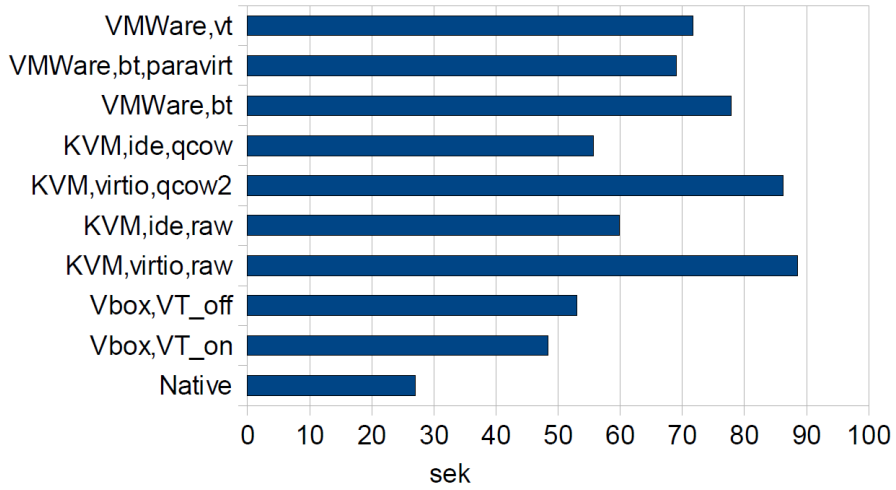

Abbildung 21: Entpacken von 64MB Kernelsourcen, Vergleich aller getesteten Konfigurationen

Platz, gefolgt von VMWare. Am längsten hat das Entpacken mit KVM mit paravirtualisierten Treibern gedauert.

5.4 Checkliste Anforderungen

5.4.1 Anforderungen an den Server

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

5.4.2 Anforderungen an den Client

C01 Dienstfindung Priorität: 1 $\times$
C02 Integration der Virtualisierungslösung Priorität: 1 $\times$
C03 VA Verwaltung Priorität: 1 $\surd$
C04 Zeitbeschränkte Laufzeit von VAs Priorität: 1 $\times$
C05 VA Update Priorität: 1 $\times$
C06 Sharing Priorität: 1 $\times$
C07 Authentifizierung von Benutzern Priorität: 1 $\times$

C08 Nutzerdaten lokal speichern Priorität: 1 $\times$
C09 Nutzerdaten zentral speichern Priorität: 3 $\times$
C10 Benutzerinterface Priorität: 3 $\surd$
C11 Plattformunabhängigkeit Priorität: 3 $\times$
C12 USB-Boot Priorität: 3 $\times$

5.4.3 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 $\times$

5.4.4 Anforderungen an die VA

A01 Universeller Einsatz Priorität: 1 $\surd$, durch GeneralPurpose Betriebssysteme
A02 Peripheriezugriff Priorität: 1 $\surd$, durch virtualisierte Schnittstellen in VirtualBox
A03 Performanz, Stabilität und Ressourcennutzung Priorität: 1 $\surd$, durch Evaluation
A04 Isolation Priorität: 1 $\surd$, durch Virtualisierung
A05 Systemverträglichkeit Priorität: 1 $\surd$, durch Virtualisierung
A06 Zugriffsschutz Priorität: 2 $\times$
A07 safety & security Priorität: 2 $\times$
A08 Fehlerbenachrichtigung Priorität: 3 $\times$

5.5 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-Addresse hat und die Client Dienstfindung auf diese Addresse 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 Funktion, Appliances dynamisch an das System des Benutzers anzupassen, also Maßschneiderung bzw. Sharing zu benutzen, ist in DoVinci noch nicht verwirklicht.

6 Ausblick

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.

Literatur

[Avi07] Yaniv Kamay Dor Laor Uri Lublin Anthony Liguori Avi Kivity. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, pages 225–230, 2007.

[BDB+08] Jörg Brakensiek, Axel Dröge, Martin Botteck, Hermann Härtig, and Adam Lackorzynski. Virtualization as an enabler for security in mobile devices. In *IIES '08: Proceedings of the 1st workshop on Isolation and integration in embedded systems*, pages 17–22, New York, NY, USA, 2008. ACM.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[Bog08] Andre Bogus. *Lighttpd*. PACKT Publishing, 2008.

[Dan09] Wanja Hofer Wolfgang Schröder-Preikschat Jochen Streicher Olaf Spinczyk Daniel Lohmann. Ciao: An aspect-oriented operating-system family for resource-constrained embedded systems. In *Proceedings of the 2009 USENIX Annual Technical Conference (USENIX '09)*, pages 215–228, San Diego, CA, USA, June 2009. USENIX Association.

[EN09] Michael Engel and Jörg Nolte. Iies '09: Proceedings of the second workshop on isolation and integration in embedded systems. New York, NY, USA, 2009. ACM.

[ES08] Michael Engel and Olaf Spinczyk. Iies '08: Proceedings of the 1st workshop on isolation and integration in embedded systems. New York, NY, USA, 2008. ACM.

[Fis09] Marcus Fischer. *Ubuntu GNU, Linux*. Galileo Press, 2009.

[Fuh01] Howard Fuhs. *GNOME, der intuitive Desktop für Linux-Unix-Konfiguration und Anwendung*. dpunkt-Verlag, 2001.

[gYsGdNd08] Giheung gu Yongin-si Gyeonggi-do Nongseo-dong. *Secure Xen on ARM User's Guide*. Samsung Electronics Co., Ltd., Samsung Electronics Co., Ltd. 14-1, Korea 446-712, 1.1 edition, 12 2008.

[Hei08] Gernot Heiser. The role of virtualization in embedded systems. In *IIES '08: Proceedings of the 1st workshop on Isolation and integration in embedded systems*, pages 11–16, New York, NY, USA, 2008. ACM.

[HSH+08] Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, and Chul-Ryun Kim. 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*, pages 257–261, Jan. 2008.

[L+07] Andriy Luntovskyy et al. Protocol stack and capacity modeling for wlan. Technical report, TU Dresden, 2007.

[Mar07] Peter Marwedel. *Eingebettete Systeme*. eXamen.press. Springer, Berlin [u.a.], 2007. Embedded system design <dt.>.

[PG74] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[Pre84] Ford Prefect. *Undefinierte Referenzen und Du*. Megadodo Publications, 1984.

[RSS+08] Marko Rosenmüller, Norbert Siegmund, Horst Schirmeier, Julio Sincero, Sven Apel, Thomas Leich, Olaf Spinczyk, and Gunter Saake. 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*, pages 1–6, New York, NY, USA, 2008. ACM.

[Set05] Jochen Setz. Industriestandards: Upnp. 2005.

[SLB07] Jan Stoess, Christian Lang, and Frank Bellosa. Energy management for hypervisor-based virtual machines. In *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2007. USENIX Association.

[SN05] James E. Smith and Ravi Nair. *Virtual machines : versatile platforms for systems and processes*. Elsevier, Amsterdam [u.a.], 2005.

[Su08] Disheng Su. Mini vm - extending kvm towards embedded systems. speaker at virtualization mini summit, Ottawa, 2008, 2008.

[TG08] R. Schmalenberg Th Godawa. Virtualbox/2. 2008.

[The07] Johan Thelin. *Foundations of Qt Development*. apress, 2007.

[Zim08] Dennis Zimmer. *VMware ESX 3.5*. Galileo Press, 2008.