Technische Fakultät
Albert-Ludwigs-Universität, Freiburg

Lehrstuhl für Kommunikationsysteme
Prof. Dr. Gerhard Schneider

Masterarbeit

Automatisierte Anwendungsinstallation zur Erzeugung von Ablaufumgebungen

16.07.2012

betreut durch
Klaus Rechert
Dr. Dirk von Suchodoletz

Erstprüfer
Prof. Dr. Gerhard Schneider

Zweitprüfer
Prof. Dr. Christian Schindelhauer

Cornelius Amzar
Matr.-Nr.: 2316690

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum Unterschrift

Danksagung

Ich bedanke mich bei Prof. Dr. Schneider für die Möglichkeit, die vorliegende Arbeit an seinem Lehrstuhl für Kommunikationssysteme verfassen zu dürfen. Mein Dank gilt auch meinen Betreuern Klaus Rechert und Dr. Dirk von Suchodoletz für di Unterstützung während der Arbeit. Weiterhin möchte ich mich bei Isgandar Valizada für seine Hilfe mit den Skripten und bei Alibek Kulzhabayev für die Beantwortung vieler Fragen zu seiner eigenen Masterarbeit bedanken. Die Korrektur der fertigen Arbeit hat Robin Garcia übernommen. Ihm gebührt Dank für die zahlreichen Korrekturen und Anmerkungen.

Inhaltsverzeichnis

Abbildungsverzeichnis

ABBILDUNGSVERZEICHNIS

1. Einleitung

> Wenn jemandem gelänge, die Bibliotheken und sämtliche Bücher zu vernichten – dreißig Jahre später könnte kein Schlosser mehr auch nur eine Schraube anziehen.
>
> *(Thomas Alva Edison)*

Es gibt über 4000 Jahre alte, in Stein gehauene Inschriften, die wir heute noch entziffern können und die uns vieles über das Leben unserer Vorfahren erzählen. Die heutige digitale Technik entwickelt sich jedoch so rasant, dass es Archiven und Bibliotheken kaum noch möglich ist, Schritt zu halten. Seit Einführung der Commodore-Personalcomputer, später der x86-Architektur mit den ersten Betriebssystemen von Microsoft sind erst knapp 30 Jahre vergangen – Personalcomputer wurden damals erst wirklich populär. Doch es entstanden in diesem vergleichsweise kurzen Zeitraum eine Vielzahl von (proprietären) Dokumenttypen und Softwarearchitekturen die zueinander inkompatibel sein können. Auch durch das Internet wurde das Informationsaufkommen vervielfacht. Die Menschheit erzeugt heute jeden Tag riesige Mengen von digitalen Objekten. Darunter versteht man in erster Linie Dokumente, aber weitergefasst auch Software oder Computerspiele. Gerade letztere sind für Museen interessant, da sie ein Bild unserer Kultur und Freizeitgestaltung liefern. Es muss eine Möglichkeit gefunden werden, alle digitalen Objekte so zu koonservieren, dass man sie auch nach langer Zeit noch betrachten und nutzen kann, um unser heutiges Leben und die damit verbundene Kommunikation und Freizeitgestaltung nachvollziehen zu können.

Es gibt viele Einsatzzwecke für die digitale Langzeitarchivierung. Alle Nutzer verfolgen unterschiedliche Zwecke und sind an bestehende Gesetze und Regeln zur Datenaufbewahrung gebunden [Strodl u. a. 2007]. Während Unternehmer an der langfristigen und sicheren Aufbewahrung riesiger Dokumentenbestände interessiert sind, wollen Privatlaute vielleicht nur ihre Fotos behalten. Das Open Planets Framework (früher PLANETS) bietet Werkzeuge und Strategien die dazu beitragen sollen, auch in ferner Zukunft noch Zugriff auf alte digitale Objekte zu haben [OPF 2011; Farquhar und Hockx-Yu 2007].

Neben Problemen beim Auslesen veralteten Datenträger [Rothenberg 1999] stellen auch nicht mehr gebräuchliche Dateiformate ein Problem für die Archivierung digitaler Objekte dar. Bei Textdateien kämpft man hier vorallem mit unbekannten Kodierungen, aber auch mit unbekannten internen Strukturen und Auszeichnungen der Dokumente. Komplizierter wird es beispielsweise schon bei Formaten von CAD-Programmen. In solchen Dateien liegen Baupläne für Flugzeuge oder Schiffe vor, die auch Jahrzehnte nach ihrem Entwurf noch genutzt werden. Wie sieht die Situation erst in 100 Jahren aus, wenn sich vielleicht Historiker für bestimmte digitale Objekte interessieren? Wer kann heute voraussehen, welche Technik wir dann einsetzen? Eine naheliegende Lösung wäre das Konservieren der Hardware zusammen mit der Software in einer Art Computermuseum. Dadurch ergeben sich aber Probleme hinsichtlich Wartung und Platzbedarf. Außerdem wäre ein Zugriff der Allgemeinheit auf die Ausstellungsstücke nur schwer möglich und mit Risiken verbunden [von Suchodoletz 2009; Philipps 2010].

Eine mögliche Lösung ist die wiederholte *Migration* der Dokumente in ein moderneres Format, sobald sich abzeichnet, dass das Ursprungsformat nicht mehr lange gebräuchlich sein wird. Dies hat jedoch einen Haken: interaktive Objekte wie Computerspiele lassen sich nicht trivial migrieren. Dies würde einer Neuentwicklung nahe kommen. Außerdem zeichnet es sich wie beim bekannten Rosette-Stein – dieser Stein wurde 1799 von Napoleons Expedition in Ägypten entdeckt und führte 1822 zur Entschlüsselung der Hieroglyphen – erst viel später ab, wie wichtig ein bestimmtes Objekt ist. Beim Migrationsansatz müsste man also generell alle jemals erzeugten Dokumente migrieren [Anderson u. a. 2010].

Eine andere Herangehensweise ist die *Emulation* der nötigen Hardware auf einem modernen Unterbau. Emulation bedeutet, dass die komplette Hardware simuliert wird, sodass die emulierte Software glaubt, auf der Hardware zu laufen, für die sie ursprünglich entwickelt wurde. Emulation ist also ein Ansatz, der es ermöglicht, Anwendungen auf völlig anderen Rechnersystemen auszuführen, zum Beispiel ein für den Commodore 64 entwickeltes Programm auf einem modernen x86-Prozessor. Der Emulator läuft in einem (aktuellen) Host-Betriebssystem und stellt unter diesem ganz normale Software dar. Im Emulator wird dann das veraltete Betriebssystem emuliert. Derzeit werden vor allem alte Versionen von Windows wie 3.11, 95 oder 98 emuliert. Diese Einschränlung rührt aber daher, dass ein Großteil der vorliegenden Dokumente mit diesen Systemen erzeugt wurden. Der Dateiaustausch zwischen den beiden Systemen ist mittels Datenträgerabbildern oder Netzwerkfreigaben möglich. Es existieren Emulatoren für alle verbreiteten Hardware-Plattformen, von Spielkonsolen bis hin zu den meisten Großrechenanlagen (Mainframes) [von Suchodoletz 2009]. Der Ansatz der Emulation in der digitalen Langzeitarchivierung wird in Abbildung 1.1 dargestellt.

Zwar kann ein Compiler den Emulator in Grenzen auch für zukünftige Plattformen übersetzen, doch dies funktioniert nur, solange sich die Hardware nicht zu sehr ändert. Ändert sich der Befehlssatz des Prozessors grundlegend, so muss der Code des Emulators selbst angepasst werden. Die Entwicklung eines Emulators für längst eingemottete Hardwareplattformen ist eine sehr anspruchsvolle Aufgabe und nur mit guter Dokumentation der alten Plattformen durchführbar. Dies ist der größte Nachteil des Emulationsansatzes. Es besteht die Gefahr, dass ein Emulator irgendwann nicht mehr weiterentwickelt wird, und eine Neuentwicklung am nicht mehr vorhandenen Wissen scheitert.

Migration und Emulation sollten gleichberechtig nebeneinander stehen [Anderson u. a. 2010]. Einen Großteil der einfachen Dokumente kann man durch Migration ohne Einschränkungen sichern. Komplexe digitale Objekte wie Computerspiele lassen sich durch Emulation archivieren. Durch die Möglichkeit der Migration *im* emulierten System entfällt auch die Notwendigkeit einer ständigen Migration in aktuell verbreitete Formate.

Mithilfe der Emulation lassen sich veraltete digitale Objekte in ihrer Orginalumgebung betrachten. Um ein bestimmtes digitales Objekt in einem emulierten Betriebssystem öffnen zu können, muss man in den allermeisten Fällen zusätzliche sekundäre Objekte – sprich Anwendungsprogramme – installieren, da nur sie den Umgang mit dem jeweiligen Format beherrschen und gegebenenfalls eine Migration in ein heute gebräuchlicheres Format durchführen können. Durch die Installation der benötigten Software erhält man eine *Ablaufumgebung* für das Betrachten des jeweiligen Dokumenttyps.

Häufig gibt es mehrere unterschiedliche Anwendungen, die ein bestimmtes Format öffnen können. Das sieht man sehr gut an Microsoft Office und OpenOffice. Letzteres kann zwar die Dokumente in den von Microsoft Office verwendeten, proprietären Formaten *.doc bzw. *.docx lesen und schreiben, jedoch ist hier nicht garantiert, dass wirklich alle Feinheiten des Formats miteinbezogen werden. So kommt es bei komplexeren Dokumenten häufig zu Darstellungsfehlern. Der Grund dafür liegt im nicht offengelegten Aufbau der Dateien, der mühsam durch Reverse Engineering entschlüsselt werden muss. Dreh- und Angelpunkt einer orginalgetreuen Darstellung sind also die verwendeten Anwendungen. Hier muss man möglicherweise mit mehreren Anwendungen experimentieren, um ein möglichst optimales Ergebnis zu erhalten [Philipps 2010].

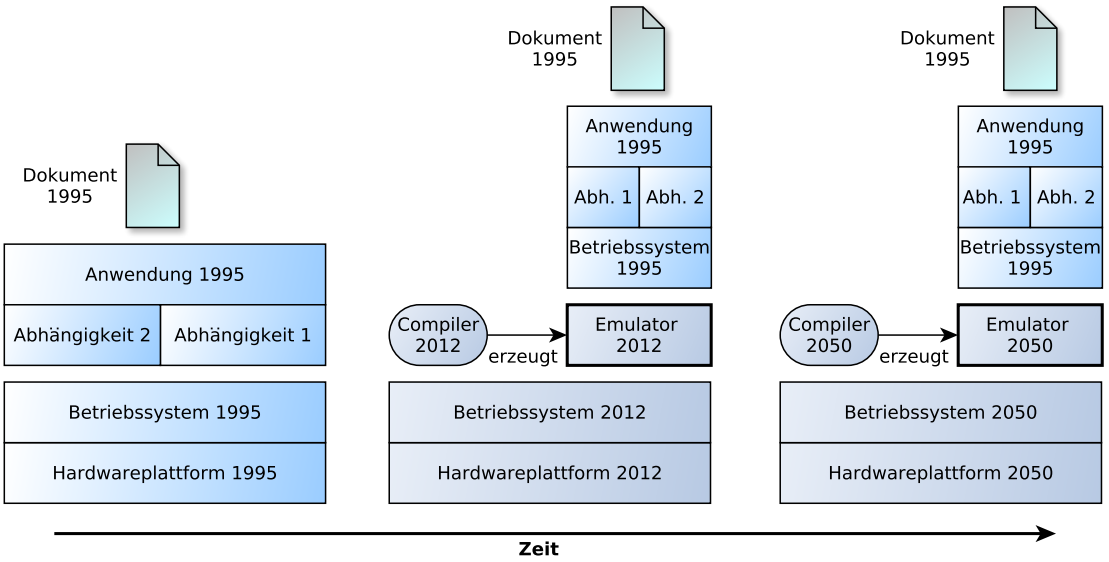


Abbildung 1.1.: Arbeitsweise der Emulation zur digitalen Langzeitarchivierung [Tchayep 2011]

1.1. Motivation

Digitale Langzeitarchivierung durch Emulation ist, abhängig von der Zahl der Dokumente, ihrem Alter und Typ und dem beteiligten Personal eine teuere Aufgabe, insbesondere für Bibliotheken und Archive, die die Aufgabe haben große Mengen solcher digitaler Objekte zu archivieren. Je mehr Zeit vergeht, desto mehr gerät das Wissen über die Bedienung der Anwendungen und Betriebssysteme verloren. Man kann also nicht erwarten, dass ein Benutzer des Archivs sich mit der Bedienung dieser Anwendungen gut auskennt.

Häufig wird zur Betrachtung eines bestimmten Dokuments nicht nur die eigentliche Anwendung benötigt, sondern noch weitere Abhängigkeiten wie Laufzeitumgebungen, Codecs, Treiber und Schriftarten [Reichherzer und Brown 2006]. Dies gilt in besonderem Maße für hochgradig interaktive Objekte wie Computerspiele. Sie bauen zunehmend auf die verwendete Hardware auf und benötigen bestimmte Softwarekomponenten wie (Grafik- oder Sound-)Treiber. Die Reihenfolge der einzelnen Installationen ist durchaus von Interesse [Rechert u. a. 2009]. So läßt sich ein Spiel möglicherweise nicht ohne eine bestimmte 3D-Grafik-Bibliothek installieren. Durch die Vielfalt der Anwendungen und ihrer Abhängigkeiten ergibt sich ein großer Zeit-

und Ressourcenaufwand für die Bereitstellung einer fertigen Ablaufumgebung für alle interessierenden Formate. Schriftarten sind teilweise sogar dokumentspezifisch, was eine korrekte Darstellung weiter erschwert. Allerdings sind Schriftarten nicht so kritisch für die reine Inhaltswiedergabe, da meist automatisch möglichst ähnliche Alternativen benutzt werden. Zur Erzeugung einer Ablaufumgebung sind folgende drei Schritte notwendig

1. Erkennung der benötigten Softwarepakete, des Betriebssystems und anderer Parameter
2. Archivierung dieser Pakete, Erfassung von Metadaten
3. (Automatisierte) Wiederherstellung einer Ablaufumgebung aus dem Archiv

Man unterscheidet zwischen statischer und dynamischer Erzeugung der Ablaufumgebungen. Bei der statischen Erzeugung werden für alle in Frage kommenden Dokumenttypen im Voraus Ablaufumgebungen erzeugt. Dies ist bei der Vielzahl der heute bekannten Dateiformate, Anwendungen und nicht zuletzt Abhängigkeiten zwischen den einzelnen Softwarepaketen ein nicht zu unterschätzender Aufwand. Bei der dynamischen Erzeugung dagegen werden die notwendigen Anwendungen erst dann aufgespielt, wenn jemand konkret ein bestimmtes Format betrachten möchte. Auch eine Kombination aus beiden ist denkbar, zum Beispiel in Form eines Caching-Algorithmus oder einer statischen Archivierung nur für die am meisten angefragten Objekte. Dies stellt in der Praxis wohl den besten Kompromiss aus Speicherbedarf für die Ablage statischer Ablaufumgebungen und Rechenzeit-Bedarf für die dymanische Erzeugung dar.

Die automatische Erzeugung von Ablaufumgebungen ist also sinnvoll, weil

- sie hilft Kosten zu sparen, gerade bei vielen Abhängigkeiten,
- dadurch Fehler durch unerfahrene Benutzer vermieden werden können,
- der Benutzer dann kaum Kenntnisse über veraltete Installationsroutinen braucht,
- dadurch erst die dynamische Erzeugung praxistauglich wird.

1.2. Forschungsstand

In [von Suchodoletz 2009] findet sich eine umfangreiche Beschreibung der digitalen Langzeitarchvierung und ihrer Zielsetzungen sowie eine Gegenüberstellung verschiedener Emulatoren. QEMU [QEMU Developers 2011] kristallisiert sich neben Dioscuri

[van der Hoeven 2007a] aufgrund seiner Vielfältigkeit und Automatisierbarkeit als am besten geeigneter Emulator heraus. Die Idee zum Einsatz von QEMU im Bereich der digitalen Langzeitarchivierung wird in [Welte 2009] und [von Suchodoletz u. a. 2010] aufgegriffen und weiterentwickelt. In [Valizada 2011] wurde ein Workflow zur der Migration digitaler Objekte vorgestellt. Das Verfahren wurde mithilfe von VNCplay [Zeldovich und Chandra 2005; Rechert u. a. 2010] implementiert. Die Grundlagen für den unbeaufsichtigten Ablauf der Migration stammen aus [Genev 2010]. Das Konzept dieser Aufzeichnungen wird hier benutzt um Softwareinstallationen unbeaufsichtigt durchzuführen.

Dass sich die beiden grundsätzlichen Verfahren – Migration und Emulation – nicht gegenseitig ausschließen, sondern ergänzen, wird in [Anderson u. a. 2010] gezeigt. Dies widerspricht der ursprünglichen Meinung von [Rothenberg 1999]. Ein Vergleich der beiden Verfahren findet sich zum Beispiel in [Anderson u. a. 2009]. Beide Verfahren müssen gleichberechtigt nebeneinander existieren, denn man kann weder alle digitalen Objekte migrieren, noch kann man alle Probleme durch Emulation lösen. Dies wird anhand des Beispiels Computerspiele gezeigt. Außerdem wird in dieser Arbeit gezeigt, wie wichtig die Speicherung von Metadaten-Strukturen ist.

Auch bei PREMIS[1] [Caplan 2009] und METS[2] [McDonough 2006] handelt es sich um anerkannte Standards für die Speicherung von Metadaten in Bibliotheken und Archiven. Allerdings sind sie mehr auf die Migration ausgelegt. Bei der Emulation werden aufgrund der vielfältigen Plattformen und Architekturen viel mehr Metadaten benötigt. Beim gerade ausgelaufenen EU-Projekt KEEP[3] wird versucht, basierend auf PREMIS und METS einen Standard für die Erfassung von Metadaten und Abhängigkeiten bei der Emulation zu entwickeln [Anderson u. a. 2009]. Die hier vorgestellte Datenbank bezeichnet man als TOTEM.

In [Philipps 2010] wird die Konzeption und der Aufbau des Softwarearchivs beschrieben. Es wird ein detaillierter Überblick über die Voraussetzungen einer erfolgreichen Archivierungsstrategie gegeben. Dabei wird auch auf die Probleme bei einer dynamischen Erzeugung der fertigen Nutzungsumgebung eingegangen. Anschließend wird eine browserbasierte Anwendung vorgestellt, die den Zugriff auf das Archiv erlaubt. Es wird aber wenig auf die Schwierigkeiten bei der Archivierung von Software und der automatisierte Installation eingegangen.

Solche Probleme entstehen unter anderem durch Unterschiede bei der Verarbeitung von Mauseingaben in unterschiedlichen Betriebssystemen – sogar innerhalb einer Betriebssystemfamilie. Dies wurde in [Ruzzoli 2009] erkannt. In [Kulzhabayev 2012]

[1] PREservation Metadata: Implementation Strategies
[2] Metadata Encoding and Transmission Standard
[3] Keeping Emulation Environments Portable

werden die Gründe dafür durch Experimente offengelegt. Es wurden Schwellwerte festgelegt, bei deren Einhaltung, das Problem nicht synchroner Mausbewegungen zumindest abgemildert wird. Gelöst ist das Problem dadurch aber nicht, es erschwert weiterhin jede zuverlässige Aufzeichnung und Wiedergabe von Mausinteraktionen sowie auch die Steuerung emulierter Umgebungen durch den Benutzer. Dies spielt bei der Implementation in Kapitel 4 eine Rolle. Außerdem wurden in dieser Arbeit Unterteilungen der Aufzeichnungen definiert, die hier weiterverwendet werden.

In [TNA 2010] wird DROID[4], ein Tool zur Erkennung des Typs von Dateien vorgestellt. Dazu werden Dateityp-Header auf Byteebene untersucht, zusätzlich werden auch MIME-Type und Dateierweiterung genutzt. Da diese beiden bestehenden Identifikationsmöglichkeiten nicht eindeutig sind, wird die PUID[5] eingeführt. Dieses Verfahren ist Voraussetzung für die automatisierte Bereitstellung passender Ablaufumgebungen für die Betrachtung eines gegebenen digitalen Objekts.

Ein Tool zur Erkennung von Abhängigkeiten in Microsoft Office Dokumenten wird in [Reichherzer und Brown 2006] vorgestellt. Die Autoren belegen, dass es eine große Vielfalt von eingebetteten Schriftarten gibt und weisen auf die Schwierigkeiten hin, die dadurch bei der Archivierung entstehen. Bei in Office-Dokumente eingebetteten Multimediainhalten gibt es dagegen nur relativ wenige Formate, sodass hier eine Installation in der Ablaufumgebung leichter fällt. In [Rechert u. a. 2009] wird gezeigt, dass auch die Reihenfolge der Installation von Abhängigkeiten relevant ist. Beides verdeutlicht die Notwendigkeit einer automatischen Installation.

1.3. Zielsetzung

Das Ziel dieser Arbeit ist es, den dritten Arbeitsschritt in der obigen Aufzählung, das heißt die automatisierte Wiederherstellung einer Ablaufumgebung zu implementieren. Dazu müssen zuerst die notwendigen Fähigkeiten analysiert werden. Die zu implementierende Komponente baut auf ein Archiv von Softwareabbildern und die im vorhergehenden Abschnitt vorgestellten Lösungen zur Erkennung von Dokumenttypen auf. Anhand der daraus bekannten Abhängigkeiten soll dann automatisch eine Ablaufumgebung zur Verfügung gestellt werden, die die jeweiligen digitalen Objekte problemlos darstellen kann.

Die bestehende Lösung zur Aufzeichnung und Wiedergabe von Benutzerinteraktionen muss an die neuen Erfordernisse angepasst werden. Die Anpassungen ergeben

[4] Digital Record Object IDentification
[5] PRONOM Unique IDentifier

sich durch die Analyse von Installationsvorgängen. Darüber hinaus sind weitere Optimierungen der Benutzerfreundlichkeit vorgesehen. Dazu zählt es auch, die Bedienung möglichst eingängig zu gestalten, sodass ein Archivbenutzer in Zukunft nicht dicke Bedienungsanleitungen lesen muss.

Der Rest der vorliegenden Arbeit lässt sich grob in einen formalen und einen praktischen Teil gliedern. In Kapitel 2 werden Grundbegriffe erklärt und darauf aufbauend eine formale Definition der benötigten Werkzeuge gegeben. Es wird auf die existierenden Teile des Softwarearchivs eingegangen. In Kapitel 3 folgt eine Analyse der Installationsschritte unter verschiedenen Betriebssystemen und darauf aufbauend eine genaue Zielsetzung für die zu implementierende Komponente. In Kapitel 4 schließlich folgt nach Angabe von Designvorgaben die Implementation in die bestehende Software. Zum Schluss wird das Ergebnis evaluiert. Im letzten Kapitel erfolgt schließlich ein Fazit und ein Ausblick auf zukünftige Entwicklungen.

2. Überblick über das bestehende Framework

> The computer was born to solve problems that did not exist before.
>
> *(Bill Gates)*

Die digitale Langzeitarchivierung war und ist seit über zehn Jahren Gegenstand verschiedener Forschungsprojekte. In diesem Kapitel wird beschrieben, wie die digitale Langzeitarchivierung verwirklicht werden kann. Dazu werden die relevanten Punkte aus den vorhergehenden Arbeiten zusammengefasst und ihr Zusammenhang mit dieser Arbeit erklärt. Der grundlegende Ablauf der des hier vorgestellten Ansatzes wird in Abbildung 2.1 beschrieben.

2.1. Plattformunabhängigkeit und Bedienkonzepte

Das Ziel der digitalen Langzeitarchivierung ist es, digitale Objekte für möglichst unbegrenzte Zeit verfügbar zu halten. Dazu zählt insbesondere ein uneingeschränkter Zugriff unter allen denkbaren Soft- und Hardwareplattformen, unabhängig von der verwendeten Eingabemethode. Hier wird zur Langzeitarchivierung wie in der Einleitung vorgestellt die Emulationsmethode (Abbildung 1.1) eingesetzt.

Ein Emulator kann theoretisch jede Hardware emulieren, dazu sind aber Anpassungen an die zum jeweiligen Zeitpunkt moderne Hardware und das Betriebssystem notwendig. Sollte der Emulator eines Tages nicht mehr weiterentwickelt werden, so kann man für den hier vorgestellten Zweck auch andere Emulatoren benutzen, sofern sie über eine Schnittstellt zur Übertragung von Benutzereingaben und Bildschirminhalten (über Netzwerk) verfügen. Richtlinien und Tests zur Bestimmung eines geeigneten Emulators befinden sich in [Tchayep 2011], eine aktuelle Auswahl wird in [von Suchodoletz 2009] vorgestellt und bewertet.

Bei dem hier vorgestellten Ansatz werden Benutzerinteraktionen aufgezeichnet, damit man in Zukunft bei der Nutzung des Archivs möglichst wenig Fachkenntnisse

über die Bedienung benötigt. Die Aufzeichnungen (siehe auch Abschnit 2.4) werden mit einer möglichst plattformunabhängigen Software erzeugt und wiedergeben. So sind auch die Aufzeichnungen möglichst unabhängig vom Emulator selbst und der Wiedergabe-Software. Die Aufzeichnungen sollten in einem möglichst einfachen Format abgelegt werden. Optimal ist hier eine Textdatei in grundlegender ASCII-Kodierung[1]. Diese Kodierung ist sehr einfach aufgebaut und darüberhinaus weit verbreitet. Dadurch sollte es auch in Zukunft möglich sein, ein Programm zu schreiben, dass solche Aufzeichnungen wiedergeben kann. Die Aufzeichnungen an sich funktionieren zwar aufgrund der Anordnung von Bedienelementen und Unterschieden in der Interpretation von Benutzerinteraktionen nicht betriebssystemübergreifend, aber man kann in jedem denkbaren Betriebssystem Aufzeichnungen nach diesem Schema erzeugen.

Generell muss bei jeder Art von Bedienkonzept irgendwie eine Auswahl getroffen werden können, dazu sind Koordinaten notwendig. Damit ist klar, dass die Koordinaten, der jeweiligen Aktivität (Mausklicks, Gesten, etc.) angegeben werden müssen. Texteingaben dürften – wie bereits erwähnt – auf längere Sicht die heute bekannten Kodierungen benutzen. Tastatureingaben erfolgen bei allen bekannten Betriebssystemen an der Position des Cursors, und betreffen immer die im Vordergrund befindliche Anwendung.

Zukünftige Bedienkonzepte stellen damit an sich kein Problem dar – auch wenn dazu gegebenenfalls das Format der Aufzeichnungen erweitert werden muss. Das wird bei der aufkommenden Gestensteuerung deutlich. Einfache Gesten lassen sich noch als Tastendrücke (Pfeiltasten) simulieren, doch bei komplizierteren Gesten benötigt man einen neuen Typ von Benutzerinteraktion. Neuartige Gesten wie bei Apples Touchpad[2] oder Microsofts Kinect-Kamera[3] lassen sich als ein String (z.B. „stretch“, „shrink“, „turn-left“) beschreiben. Denn die Auswertung solcher Gesten funktioniert immer so, dass ein Algorithmus mit gewisser Fehlertoleranz festgelegte Bewegungsmuster erkennt und dann eine zum entsprechenden Kontext passende Reaktion auslöst. Die Wiedergabekomponente muss dann nur passende Ereignisse auf Ebene des Betriebssystems bzw. der verwendeten Laufzeitumgebung generieren. Heute stellt sich das Problem aber eigentlich noch nicht, da die jeweiligen Funktionen sich praktisch immer auch mit Maus oder Tastatur – schlimmstenfalls über das Menü – erreichen lassen. Solche neuen Bedienkonzepte müssen sich erst einmal überall etablieren und vorallem standardisiert werden. Bislang handelt es sich lediglich um Insellösungen.

[1] American Standard Code for Information Interchange

[2] man kann hier zahlreiche Wisch-Gesten mit bis zu vier Fingern vornehmen, http://www.apple.com/de/macosx/whats-new/gestures.html, 10.07.2012

[3] diese 3D-Kamera ermöglicht die Auswertung von Körperbewegungen im freien Raum.

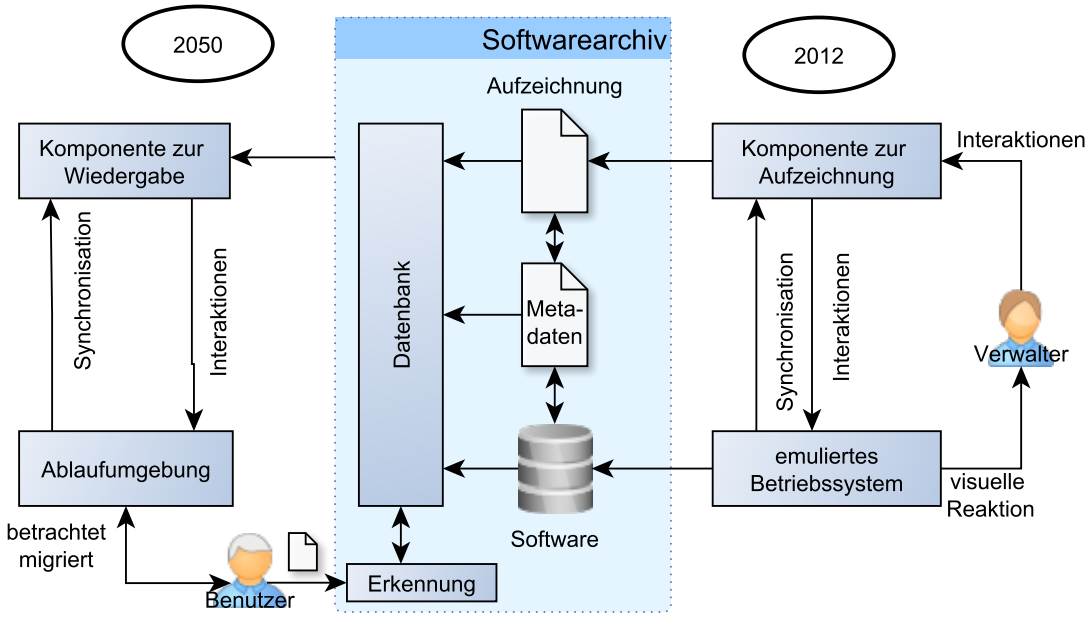


Abbildung 2.1.: Das Softwarearchiv und Aufzeichnung der Interaktionen und Wiederherstellung einer Ablaufumgebung

2.2. Aufbau des Softwarearchivs

Das Archiv besteht aus einer Menge von *Basisimages*, *Anwendungsimages*, *Aufzeichnungen* und *Metadaten*. Basisimages enthalten nur die bereits installierten Betriebssysteme, abgesehen von grundlegenden Komponenten wie Treibern. Anwendungsimages sind meist Abbilder der ursprünglichen Installationsdatenträger, also Disketten oder optische Datenträger. Die Anwendungen werden nicht auf die Basisimages aufgespielt, sondern es wird dazu eine Kopie erzeugt, die dann sowohl das Betriebssystem als auch die Anwendung(en) enthält. Bislang gibt es Aufzeichnungen, die die Migration eines Dokuments vom Format X in das Format Y beschreiben. Aufzeichnungen, die die Installation eines Programms auf einem Basisimage beschreiben, werden in den folgenden Kapiteln vorgestellt.

Metadaten verknüpfen die unterschiedlichen Images miteinander und legen reproduzierbar und unzweideutig die Betriebsparameter fest. Es gibt bereits mehrere Standards für die Speicherung von Metadaten, wie PREMIS, METS und METS. Diese Standards sind aus Projekten der digitalen Langzeitarchivierung hervorgegangen. Sie sind jedoch für den hier vorgestellten Ansatz unzureichend, da sie die abgelegten

Informationen zu wenig miteinander verknüpfen und sich zudem auf den Migrationsansatz beschänken. Das bedeutet, dass die Datenbanken kaum Informationen zur Hardware enthalten. Es handelt sich dabei also um Datenbanken, die beispielsweise alle Versionen des PDF-Standards enthalten und die Programme, die solche Dateien öffnen oder erzeugen können. Die im Projekt KEEP entwickelte Datenbank TOTEM entspricht schon eher den Bedürfnissen dieser Arbeit. Sie wurde speziell für die digitale Langzeitarchivierung durch Emulation entworfen.

Im Verlauf dieser Arbeit bezeichnet der Begriff *Datenbank* eine Speicherung der Systemanforderungen, Versionen und Abhängigkeiten zur Betrachtung eines bestimmten Dokumenttyps. Diese Datenbank besteht wie in [Anderson u. a. 2009] beschrieben aus Tabellen für Hardwareplattformen, Software, Dateitypen. Sie ermöglicht es somit, genau festzustellen. welches Programm in welcher Version auf welcher Plattform einen gegebenen Dokumenttyp öffnen kann.

Alle Schritte, die notwendig sind, um eine fertige Nutzungsumgebung für einen bestimmten Objekttyp herzustellen bezeichnet man als *View-Path*[4]. Ein Beispiel dafür findet sich in Abbildung 2.2. Er enthält die Hardwarekonfiguration, die Installation der einzelnen Anwendungen und ihrer Abhängigkeiten und schließlich das Öffnen des Dokuments. Die Erzeugung eines View-Paths wird unter anderem in [Philipps 2010] besprochen. Der View-Path hängt stark von einer zuverlässigen Erkennung der zu betrachtenden Dateiformate ab. Verfahren dafür werden in [Reichherzer und Brown 2006] und [TNA 2010] vorgestellt. Mindestens ein möglicher View-Path muss aus den abgelegten Metadaten in Zusammenspiel mit der Datenbank generiert werden.

Das Archiv gliedert sich in *Back-* und *Frontend*. Darunter versteht man verschiedene Schnittstellen, die von unterschiedlichen Benutzergruppen eingesetzt werden. Das Backend ermöglicht das hinzufügen von Abbildern, Metadaten und Aufzeichnungen. Das Frontend ermöglicht dagegen die Nutzung der bereitgestellten Daten. Vereinfacht gesagt ermöglicht das Frontend nur Lesezugriff auf das Archiv, während über das Backend neue Images und Aufzeichnungen hinzugefügt werden können.

Die in Abbildung 2.1 gezeigte Datenbank ermöglicht eine Bestimmung der notwendigen Anwendungen, Abhängigkeiten und Betriebssysteme und somit letztendlich Aufzeichnungen. Das Softwarearchiv führt die notwendigen Aufzeichnungen auf entsprechenden Basisimages der Reihe nach aus und erzeugt dadurch eine Ablaufumgebung. Mit Hilfe dieser Ablaufumgebung kann der Endbenutzer seine Dokumente betrachten oder migrieren.

[4] in der Literatur auch als *Pathway* bezeichnet.

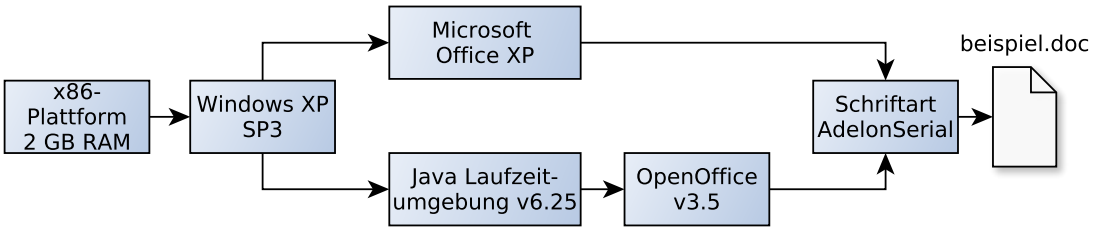


Abbildung 2.2.: View-Path eines *.doc-Dokuments

2.3. Benutzergruppen

Es gibt drei verschiedene Gruppen von Benutzern. Erstens den *Archivar* (Curator). Er trägt die zu archivierenden Programme zusammen. Mit der hier vorgestellten Technik hat er relativ wenig zu tun. Der *Verwalter* (Contributor). Er bereitet die Softwarepakete für die Installation vor und zeichnet die Installation auf. Ein Verwalter hat ein tieferes Verständnis des Archivsystems und der archivierten Betriebssysteme und kann gegebenenfalls Probleme bei der Installation lösen. Der Verwalter benutzt dafür vorwiegend das Backend des Archivsystems. Die dritte Gruppe von Benutzern stellen die sogenannten *Endbenutzer* dar. Endbenutzer sind die täglichen Nutzer des Archivs. Sie haben wahrscheinlich wenig Erfahrung in der Benutzung archivierter digitaler Objekte. Die Komplexität des Verfahrens wird deshalb bestmöglich vor ihnen verborgen. Die Betriebsparameter wie benötigte Basisimages und Anwendungen legt das Softwarearchiv anhand einer Erkennung der eingereichten digitalen Objekte automatisch fest. Die Gruppen unterscheiden sich durch unterschiedliche Zugriffsrechts auf die Datenbank. Schreibzugriff haben nur Verwalter und (eingeschränkt) Archivare. Die für den Endbenutzer erzeugten Basisimages werden natürlich dennoch gesichert.

2.4. Aufzeichnungen

Die aufgenommenenen Benutzerinteraktionen werden in einer Datei aufgezeichnet. Diese Dateien enthalten Informationen über alle (Maus-)Bewegungen und Tastatureingaben, die der Benutzer innerhalb der emulierten Oberfläche vorgenommen hat. Die Aufzeichnungen werden Zeile für Zeile interpretiert und die darin enthaltenen Informationen ausgewertet. Es ergibt sich dann eine Abfolge von unterschiedlichen

Interaktionen. Jede Aktion hat genau definierte Vorbedingungen und ein eindeutiges Ergebnis.

In der Praxis ist eine Aufzeichnung eine einfache Textdatei. Jede Zeile enthält genau einen Eintrag. Jeder Eintrag besitzt ein Pflichtfeld *type* und mehrere spezifische Felder. Es gibt bislang folgende Typen

1. Mausereignis
2. Tastaturereignis
3. Synchronisationspunkt

Beispiel:

```
1 118 id=502 button=1 when=1330777308574 modifiers=16 type=java.awt
    .event.MouseEvent y=195 x=309
2 6887 id=401 when=1330777315461 keycode=40 keychar=65535 modifiers
    =0 type=java.awt.event.KeyEvent
```

Am Anfang jeder Zeile steht der Delay, also der Zeitunterschied (Delay) zur vorhergehenden Zeile. In der ersten Zeile wird ein Klick mit der linken Maustaste auf den Pixel (309, 195) ausgelöst). Diese Angabe ist in den Koordinaten des Gastsystems. In der zweiten Zeile findet dann eine Tastatureingabe statt, die Taste mit dem KeyCode 40 wird gedrückt. Wie man sieht, haben KeyEvent und MouseEvent einige gemeinsame Attribute wie *when*, das einen Zeitstempel in Millisekunden enthält. Dieser ist jedoch *nicht* für den zeitlichen Ablauf notwendig. Der zeitliche Ablauf folgt lediglich grob dem Delay zwischen den einzelnen Interaktionen. Die *ID* gibt an, um was für ein Ereignis es sich genau handelt – Mausklick, Mausbewegung oder Drag&Drop. Man unterscheidet zwischen unterschiedlichen Modifikatoren (Strg-, Shift-, Alt-Tasten).

Synchronisationspunkte werden im folgenden Abschnitt eingehend beschrieben. Darüberhinaus gibt es noch Stufen. Stufen wurden in [Kulzhabayev 2012] eingeführt und dienen zur Strukturierung des Workflows. Sie haben bislang aber keinen Einfluss auf den Ablauf des Workflows. Dies ändert sich in der vorliegenden Arbeit.

2.5. Synchronisationspunkte

Die Synchronizität ist ein entscheidender Faktor bei der Wiedergabe. Durch hohe Last im Hostsystem kann es passieren, dass das Gastsystem träger reagiert als bei der

Aufzeichnung. Dadurch kann es vorkommen, dass eine Benutzerinteraktion gesendet wird, obwohl die betreffende Schaltfläche noch garnicht gezeichnet wurde, Dass dies zu einem fehlerhaften Ablauf führt, ist klar.

Aus diesem Grund wurden sogenannte Synchronisationspunkte (*SyncPoints*) eingeführt. Dabei handelt es sich um eine Art Fingerabdruck der Pixel in einem Ausschnitt des Bildschirms. Der SyncPoint wird in der Aufzeichnung abgelegt und bei der Wiedergabe geprüft. Fällt die Prüfung positiv aus, so wird normal fortgefahren. Man sagt „der SyncPoint stimmt überein“. Dafür wurde ein Schwellwert definiert, der den maximalen Anteil von Pixeln angibt, deren Farbe sich geändert haben darf. Andernfalls wird der Workflow angehalten, bis die Prüfung positiv ausfällt. Auf diese Weise wird effektiv verhindert, dass eine Interaktion ausgelöst wird, während die betreffenden Elemente der Benutzeroberfläche noch nicht gerendert sind.

Beispiel:

```
1 type=sync px422y292=-16777216 px422y293=-16777216 px422y290
    =-16777216 [...]
```

Der SyncPoint besteht aus den Farbwerten einzelner Pixel in einem kleinen Ausschnitt des Bildschirms. In der Praxis werden Wert von 10×10 Pixeln eingesetzt. Ein SyncPoint vom gesammten Bildschirminhalt ist wegen dynamischen Komponenten wie der Uhrzeit oder Desktopsymbolen und dem entstehenden Rechenaufwand nicht praxistauglich. Der Verwalter kann die Position dieses Ausschnitts frei wählen, muss aber darauf achten, dass keine solchen dynamischen Komponenten in diesem Bereich liegen.

2.6. Aufzeichnung von Interaktionen

Das Prinzip der Aufzeichnung wird in Abbildung 2.1 deutlich. Die Aufzeichnungen werden ein einem Verwalter über das Back-End des Systems erzeugt. Dazu wählt dieser ein Basisimage und Images von Installationsdatenträgern einer Anwendung. Er ruft dann den Emulator mit entsprechenden Parametern auf. Die Steuerung erfolgt durch die Aufzeichnungskomponente hindurch, die auf diese Weise alle Interaktionen aufzeichnen kann. Der Verwalter installiert die Software vom eingebundenen Image. Dabei ist es auch möglich, während der Aufzeichnung das Image auszuwerfen und ein anderes zu mounten. Dadurch lassen sich auch Anwendungen mit mehreren Installationsdatenträgern aufspielen. Folgende Zeilen zeigen beispielhaft den Aufruf des Emulators und den Start des Aufzeichnungskomponente.

```
1 qemu-system-i386 <Basisimage> -cdrom <CD-Image> -fda <Floppy-
     Image> -hdb <HDD-Image> -snapshot -vnc :0
2 java VncViewer HOST localhost PORT 5900 autorecord yes
```

Der Befehlsaufruf *qemu-system-i386* legt die Prozessor- und Rechnerarchitektur, die emuliert wird, fest. Natürlich muss die emulierte Software auch auf dieser Plattform lauffähig sein. Der Emulator QEMU liegt für ein sehr breites Spektrum von Plattformen vor. Der Schalter *-snapshot* sorgt dafür, dass die Images nicht verändert werden. Schreibzugriffe werden gepuffert und dann verworfen. Über den Schalter *-vnc* wird die Steuerung über das VNC-Protokoll[5] aktiviert. Erst dadurch ist die Aufnahme und Wiedergabe durch entsprechende Software möglich. In der zweiten Zeile wird die Aufnahmekomponente gestartet. HOST und PORT legen die Netzwerkparameter fest. Beide Komponenten verbinden sich dann via VNC-Protokoll mit dem bereitgestellten Image. Im obigen Beispiel läuft der Emulator aber auf dem gleichen Rechner wie die Komponente zur Aufzeichnung.

Die Aufzeichnungen werden zusammen mit den Basis- und Anwendungsimages und zusätzlichen Metainformationen abgelegt. Endbenutzer haben Lesezugriff auf die bestehenden Dateien, können aber neue Abbilder erzeugen.

2.7. Wiedergabe von Aufzeichnungen

Aufzeichnungen werden vom Softwarearchiv eigenständig anhand der eingereichten Dokumente zur Wiedergabe ausgewählt. Dabei muss unter Umständen eine bestimmte Reihenfolge beachtet werden, sodass alle Aufzeichnungen ohne Fehler durchlaufen können. Das liegt einerseits daran, dass manche Installationen andere Komponenten wie die Java-Laufzeitumgebung voraussetzen, andererseits funktionieren möglicherweise die Aufzeichnungen nicht, weil ein neues Desktopsymbol angelegt wurde. Nachdem die Aufzeichnungen der Reihe nach abgesapielt wurden, liegt eine Ablaufumgebung vor, die dem Endbenutzer zur Betrachtung seiner digitalen Objekte zur Verfügung gestellt wird.

[5] Virtual Network Computing, ein Protokoll für Fernsteuerung von Rechnern über ein Netzwerk

3. Analyse

> Software is like entropy: It is difficult to grasp, weighs nothing, and obeys the Second Law of Thermodynamics; i.e., it always increases.
>
> *(Norman Augustine)*

Ziel dieser Arbeit ist es, die vorhandenen Methoden zur Aufnahme und Wiedergabe von Nutzerinteraktionen so anzupassen, dass man damit jede beliebige Anwendung auf jedem Betriebssystem aufspielen kann, um eine Ablaufumgebung zur Betrachtung bestimmter digitaler Objekte zu schaffen. Dazu muss erst einmal analysiert werden, welche Fähigkeiten für solche Installationen notwendig sind. In diesem Kapitel wird auf unterschiedliche Methoden zur Installation eingegangen. Anschließend werden Probleme erörtert, die bei der Nutzung der bestehenden Software auftreten. Diese Probleme werden dann im folgenden Kapitel angegangen.

3.1. Besonderheiten der Betriebssysteme

In diesem Abschnitt wird eine Momentaufnahme aktueller Betriebssysteme gegeben. Sie ist für die digitale Langzeitarchivierung insofern interessant, als dass man daraus ableiten kann, welche Schwierigkeiten bei der Archivierung von Software generell auftreten und in welche Richtung sich die Installationsmethoden entwickeln.

3.1.1. Microsoft Windows

Windows ist mit Abstand das am weitesten verbreitete Betriebssystem. Deshalb existieren dafür auch schon eine ganze Reihe von Praxisversuchen zur Emulation und Archivierung u.a. [Reichherzer und Brown 2006; van der Hoeven 2007b]. Windows

nutzt Installationsdatenträger oder herunterladbare Installationsdateien. Die Installation kann hier durch einbinden des Datenträgers angestoßen werden. Die Installation folgt praktisch immer einem Dialogsystem mit mehreren Schritten (Lizenz, Installationspfad, Paketauswahl, usw.). Manche Programme kann man direkt als portable Variante beziehen, die keine Installation benötigt. Dies ist für die Archivierung optimal. Unter Windows besitzen viele Installationsdatenträger einen Autostart-Dialog, der starten soll, wenn man den Datenträger einlegt. Dies funktioniert bei Images aber nur höchst unzuverlässig. Bei gleichen Images erscheint er manchmal und manchmal nicht. Man muss also während der Aufzeichnung immer den Weg über den Dateimanager gehen.

3.1.2. Apple Mac OS

Mac OS ist ebenfalls ein weit verbreitetes Betriebssystem. Besonderheit ist hier die Kombination von Hard- und Software aus einem Haus. Das wirft rechtliche Fragen für die Emulation des Betriebssystems auf Nicht-Apple-Hardware auf. Mac OS nutzt ähnlich wie Windows meistens physikalische Datenträger oder downloadbare Installationsdateien. Diese liegen fast ausschließlich als Disk-Image (*.dmg) vor. Das sind direkt Abbilder, die im Betriebssystem gemountet werden müssen. Darin enthalten ist dann eine ausführbare Datei, die man direkt ausführen oder auf die Festplatte kopieren kann (*.app). Programme, die tief ins System eingreifen und daher an verschiedenen Stellen entpackt werden müssen, liegen meist als Installationsdialog (*.pkg) vor. Vorteil gegenüber Windows bei der Archivierung ist, dass sich die *.app Pakete generell ohne Installation nutzen lassen. Seit kurzer Zeit versucht Apple jedoch, alle Software in seinem Download-Portal App Store zu vereinigen. Dies wirft für die Archivierung ähnliche Probleme auf wie die Paketquellen (Repositories) bei Linux.

3.1.3. Linux

Linux setzt schon seit Jahren eine Paketverwaltung ein. Praktisch jede Linux-Software findet sich in irgendeinem Paket-Repository. Nur proprietäre Software wird noch einzeln zum Download bereitgestellt. Für den Anwender ist das fraglos sehr bequem. Abhängigkeiten werden meist automatisch aufgelöst und heruntergeladen. Die Zahl der Abhängigkeiten ist bei Linux generell sehr hoch. Wenn nun aber irgendwann das Repository seinen Dienst einstellt, kann man auf diesen Service nicht mehr zugreifen.

Für den Paketmanager APT[1] gibt es das Werkzeug APTonCD[2]. Welches die Pakete auf einer CD sichert und von dort auch wieder herstellen kann. Wenn jedoch Abhängigkeiten bestehen, die nicht auf der CD enthalten sind, so benötigt man wiederum Zugriff auf ein Online-Repository.

3.1.4. Zusammenfassung

Mit obigem Vergleich wird deutlich, dass Software in Zukunft zunehmend durch netzwerkgebundene Bezugsmöglichkeiten, also ohne physikalische Datenträger bezogen werden wird. Dies stellt für die Archivare eine neue Herausforderung dar. Das Hauptaugenmerk dieser Arbeit richtet sich aber auf einen anderen Aspekt, nämlich die Automatisierung von Installationen. Der obige Vergleich zeigt aber auch, welche Eigenschaften eine Automatisierung erfüllen muss, um betriebssystemunabhängig Anwendungen installieren zu können.

Unter Windows kann man den Installationspfad frei wählen, bei manchen Programmen gibt es mehrere Installationsmöglichkeiten oder Optionen. Der Archivar kann nicht voraussehen, welche Funktionen man in 25 Jahren zum Betrachten oder Migrieren benötigt. Natürlich könnte er auch immer die vollständige Installation wählen, doch dies führt in manchen Fällen zu weiteren Abhängigkeiten und Problemen. Allein durch die unter Windows und Mac OS häufig notwendigen Lizenznnummern ist eine Benutzereingabe während der Installation zwingend erforderlich. Rechtlich ist es nicht erlaubt, die Lizenzen in den Aufzeichnungen zu speichern und dann unterschiedliche Benutzer damit zeitgleich arbeiten zu lassen. Auch die Wahl des Installationspfads sollte variabel sein, damit die Basisimages auf möglichst kleinen Festplattenabbildern abgelegt werden können. Man benötigt also zusätzlich zur Aufzeichnung der Nutzerinteraktionen eine Möglichkeit, die Wiedergabe (und damit auch die Aufnahme) an bestimmten Stellen zu unterbrechen und eine benutzerdefinierte Auswahl zu treffen. Nach Beendigung der Eingabe muss die Installation automatisch weiterlaufen.

Für eine betriebssystemübergreifende Installation von Anwendungen ist es unabdingbar, das sowohl Tastatur als auch Mauseingaben aufgezeichnet werden. Das ist insofern problematisch, als dass häufig die Synchronisation zwischen Host- und Gast-Mausposition verloren geht. Dieses Phänomen wurde in [Kulzhabayev 2012] behandelt. Die dort vorgeschlagenen Lösungen sollten weiter verbessert werden, denn einem Archivar oder einem Endbenutzer ist es nicht zumutbar, dass er erst lernen muss, wie

[1] Advanced Packaging Tool
[2] APTonCD, http://aptoncd.sourceforge.net, 05.07.2012

man die Maus in einem bestimmten Betriebssystem richtig bewegt. Die Bedienung sollte intuitiv erlernbar sein.

3.2. Abhängigkeiten

Abhängigkeiten bestehen zwischen unterschiedlichen Softwarepaketen. So kann man zum Beispiel OpenOffice nicht installieren, wenn nicht die Java-Laufzeitumgebung vorhanden ist. Es existieren unterschiedliche Typen von Abhängigkeiten.

Abhängigkeiten des Dokuments hierzu zählen zum Beispiel Schriftarten oder (Audio-, Video-) Codecs. Das Dokument kann ohne sie garnicht oder nur mit Einbußen betrachtet werden.

Abhängigkeiten der Programme dazu zählen vorallem Libraries und Laufzeitumgebungen, die für das Ausführen eines Programms, teils auch schon für die Installation benötigt werden. Besonders modulare Betriebssysteme wie Linux haben eine große Zahl solcher Abhängigkeiten.

Implizite Abhängigkeiten entstehen beispielsweise, wenn man versucht, ein älteres Programm mit einem modernen Kompiler zu übersetzen oder umgekehrt. Das kann dazu führen, dass die Software bestimmte Methoden des Kompilers voraussetzt, die mit der Zeit veralten (*deprecated*) und schließlich aus dem Kompiler entfernt wurden. Die Software läßt sich also nur bis zu einer bestimmten Kompilerversion korrekt kompilieren.

Die ersten beiden Abhängigkeiten bezeichnet man als *technisch*. Sie lassen sich vergleichsweise einfach lösen, denn es ist bekannt (durch die Dokumentation, Fehlermeldungen oder eine Analyse des Dokumenttyps) welche Abhängigkeiten bestehen. Die dritte Gruppe von Abhängigkeiten ist schwieriger aufzulösen, denn in den allermeisten Fällen sind sie nirgends dokumentiert. Dies liegt darin begründet, dass sie zur Zeit der Erstellung des jeweiligen Programms niemand aufgefallen sind. Fehlermeldungen enthalten in diesen Fällen nicht zwingend genaue Angaben über die benötigte Version, mit der die Kompilation erfolgreich durchgeführt werden kann. Manchmal treten solche Fehler auch nur in bestimmten Umgebungen auf – beispielsweise, weil das Betriebssystem im Zusammenspiel mit einer anderen Komponente einen Bug hat. Ein weiteres Beispiel für eine implizite Abhängigkeit wäre es, wenn ein Programm (welches prinzipiell unter Windows 3.11 lauffähig ist) eine bestimmte Farbtiefe (mehr als 16 Farben) voraussetzt. Nun ist es aber so, dass Windows 3.11 standardmäßig mit 16 Farben arbeitet. Zur Nutzung von mehr Farben muss man

spezielle Grafiktreiber nutzen oder sogar die Auflösung reduzieren. Alternativ kann man ein moderneres Betriebssystem einsetzen.

Probleme durch Abhängigkeiten lassen sich bei der digitalen Langzeitarchivierung nie ganz verhindern. Man kann lediglich versuchen, sie durch eine geeignete Installationsreihenfolge, eine gründliche Analyse der Dokumente und eine sehr genaue Dokumentation der Orginal-Umgebung in den Metadaten zu verringern. Trotzdem wird es aber immer wieder Fälle geben, die eine zeitaufwändige, manuelle Nachforschung erforderlich machen. Die erkannten Abhängigkeiten werden in einer Datenbank wie der des KEEP-Projekts (TOTEM) erfasst. Sie enthält für jedes archivierte Objekt Einträge wie “runs-on” oder “depends-on”. Diese Datenbank stellt also eine sehr umfangreiche Baumstruktur da, die von der Hardware-Plattform bis zur eigentlichen-GonZales56 Anwendung reichen sollte. Gesucht ist ein Pfad oder Schnitt durch diesen Baum, eben der View-Path.

3.3. Emulatoren

Emulatoren stellen bestimmte Hardware virtuell zur Verfügung. So emuliert *qemu-system-i386* im Überblick und ohne weitere Optionen folgende Hardware[3].

- Prozessor: Intel 80386, weithin bekannt als "386er“
- Chipsatz: Intel i440fx
- Speicher: 128 MB SDRAM
- Grafik: Cirrus CLGD 5446 PCI VGA
- Sound: Creative SoundBlaster 16
- Eingabe: PS/2 Tastatur und Maus
- Netzwerk: PCI Netzwerk Adapter

Diese emulierte Hardware kann sich theoretisch von einer Version zur nächsten ändern. So wurde in der Vergangenheit die BIOS-Emulation geändert, um Probleme zu beheben. Dadurch entstehen aber mitunter neue Probleme. Es ist unabdingbar, eine exakte Beschreibung der emulierten Hardware zu hinterlegen. Dies wird auf

[3]Quelle: qemu.org, Manpages, 20.06.2012

der QEMU-Homepage versucht. Nur mit einer solchen Beschreibung wäre es später möglich, einen neuen Emulator zu entwickeln. Zusätzlich braucht man dazu auch detaillierte Informationen zur Funktionsweise der Hardware. Manchmal treten bei der Emulation Probleme auf, die sich mit bestimmten Schaltern im Emulatoraufruf beseitigen lassen. Deshalb sollte man diesen Aufruf in Form eines Kommentars einbinden. Besonders häufig braucht man den Schalter *-m*, mit welchem sich der Speicherausbau festlegen lässt.

3.4. Metadaten

Die Metadaten einer Aufzeichnung stellen eine Verbindung zwischen der in Abschnitt 3.2 vorgestellten Datenbank mit Soft- und Hardwareplattformen und den Aufzeichnungen und Images her. Sie enthalten zum Betrieb des Softwarearchivs unbedingt notwendige Informationen und garantieren die Wiederherstellung der emulierten Umgebung in Zukunft. Deshalb dürfen sie auf keinen Fall verloren gehen. Wünschenswert ist eine möglichst menschenlesbare Speicherung, da auf diese Weise eventuelle Fehler bei der automatischen Auswertung leichter entdeckt werden können. Zudem ist dadurch die Lesbarkeit in Zukunft gewährleistet, wenn man zum Beispiel einen neuen Emulator entwickeln muss.

Es existieren bereits mehrere Standards für die Speicherung von Metadaten in der digitalen Langzeitarchivierung [McDonough 2006; Caplan 2009]. Diese bieten eine große Vielfalt von Angaben zum genauen Format eines Dokuments. Während sich METS und PREMIS an die Anforderungen der Migrationsmethode richten und daher kaum Angaben zur Hardware abspeichern, ist die Datenbank des KEEP-Projekts [Anderson u. a. 2009](TOTEM) zwar von vorneherein auf die Emulation ausgerichtet, jedoch für die hier vorgestellte, automatische Installation nicht ausreichend. Bei diesem Projekt werden in der Datenbank Abhängigkeiten wie „depends-on“, „runs-on“, „is-suitable-for“ und andere zwischen Hardware-Plattformen, Softwarepaketen und Betriebssystemen gespeichert. Das ist aber immer noch nicht detailliert genug, wie man an den impliziten Abhängigkeiten sieht. Im Zusammenhang dieser Arbeit ist der entscheidende Punkt die exakte Reproduzierbarkeit der Ablaufumgebung. Es muss daher ein Abgleich zwischen der TOTEM-Datenbank und den Metadaten der Aufzeichnungen stattfinden. So kann festgestellt werden, für welchen mit Hilfe der Datenbank ermittelten View-Path die notwendigen Aufzeichnungen und Images vorliegen.

Folgende Informationen müssen zusätzlich zu der Datenbank in den Metadaten jeder Aufzeichnung enthalten sein.

1. Emulator und seine Version, mit der die Aufzeichnung erzeugt wurde.

2. Bezeichner des Betriebssystems und der installierten Anwendung.

3. Hardwareplattform, auf der die Aufzeichnung erstellt wurde, z.B. „x86-64“. Diese Angabe ist ggfs. bereits im Aufruf des Emulators enthalten.

4. Informationen zur Aufzeichnung, wie das Vorhandensein von Eingabemöglichkeiten durch den Benutzer und die Art der vorzunehmenden Eingaben.

5. In Form eines Kommentars die Kommandozeile des Emulatoraufrufs. Sie liefert Informationen über spezielle Schalter, die manchmal benötigt werden, damit eine bestimmte Software überhaupt läuft.

Genauere Angaben zur Hardwareplattform sind nur in bestimmten Fällen notwendig, da sie durch das verwendete Betriebssystem und die sonstige Software gegeben und damit in der Datenbank der Abhängigkeiten (TOTEM) enthalten sind. Im Gegensatz dazu stehen Abhängigkeiten, die nicht direkt aus den gespeicherten Systemanforderungen hervorgehen. Diese Abhängigkeiten gehen nicht direkt aus der Datenbank hervor, weil sie die Kombination zweier oder mehr Angaben erfordern. Sie sollten deshalb (sofern bekannt) in den Metadaten einer Aufzeichnung enthalten sein. Hierfür ist es aber wichtig, dass in den Metadaten eine Schreibweise abgelegt ist, die den Abgleich mit der Datenbank erlaubt.

3.5. Wahl der SyncPoints und Hinweise

Synchronisationspunkte (Abschnitt 2.5) sind ein wichtiger Bestandteil der Aufzeichnungen. Ein SyncPoint wird automatisch zu Beginn der Aufzeichnung erzeugt. Während der Installation ist der Bildschirmausschnitt, aus dem der SyncPoint gewonnen wird, jedoch besonders wichtig, denn Installationen laufen im Gegensatz zu den meisten Anwendungen nicht im Vollbildmodus. Da die SyncPoints für einen zuverlässigen Ablauf der Benutzerinteraktion-Phase mitverantwortlich sind ist ihre Wahl besonders kritisch. Man denke an folgende Situationen

- Der SyncPoint enthält die eingegebene Lizenznummer

- Der SyncPoint enthält nur einen (einfarbigen) Ausschnitt des Dialogs oder den Desktop.

- Der SyncPoint wird komplett vergessen

- Der Endbenutzer ist unsicher, wann er die Tastenkombination zum Beenden des Interaktionsmodus betätigen soll und tut das zu früh oder zu spät.

Die Position muss sich vom Verwalter frei bestimmen lassen, d.h. er wird dort erzeugt, wo der Mauszeiger ist. Dabei muss der Verwalter darauf achten, dass er nicht nur einen einfarbigen Ausschnitt, sondern bestenfalls einen (mehrzeiligen) Text – der sich dann im folgenden Dialogschritt ändert – als SyncPoint markiert. Die automatischen SyncPoints werden aber in der Bildschirmmitte erzeugt. Dies klappt bei den getesteten Applikationen zuverlässig, da sich hier die Dialogbox befindet.

Da dem Benutzer nicht unbedingt klar ist, dass von ihm eine Eingabe erwartet wird, sollte es auch dafür einen eindeutigen Hinweis geben. Beim größtenteils automatischen Installationsablauf ist es ihm nicht unbedingt bewußt, welche Anwendung gerade installiert wird und was genau das Dialogfeld von ihm will.

3.6. Mausbedienung

Bei der Emulation hat man es mit zwei völlig unterschiedlichen Betriebssystemen zu tun, die miteinander interagieren. Das merkt man besonders bei der Bewegung des Mauszeigers. Unterschiedliche Betriebssysteme verarbeiten die einzelnen Mausereignisse nicht zwingend gleich. Das führt dazu, dass der Zeiger des Host- und der des Gastsystems manchmal nicht mehr synchron sind, auch wenn der Zeiger des Hostsystems standardmäßig ausgeblendet wird. Diese Problematik wurde bereits in [Kulzhabayev 2012] untersucht. Als Lösung wird unter anderem angeraten die Mausbeschleunigung abzuschalten und Schwellwerte für die Mausbewegungen einzuhalten.

Ein weiteres Problem ergibt sich durch die Position des Mauszeigers bei der Benutzerinteraktion. Wenn der Benutzer während der Interaktionsphase die Maus bewegt, würden ja sämtliche Mausklicks, -bewegungen und auch SyncPoints danach fehlschlagen. Das Problem tritt sowohl bei der Aufzeichnung als auch bei der Wiedergabe auf. Es lässt sich auf zwei Arten lösen.

- man erlaubt von vorneherein nur Tastatureingaben und keine Mauseingaben. Das reicht für Lizenznummern völlig aus, ist aber eine sehr eingeschränkte Lösung.
- die Position des Mauszeigers wird zurückgesetzt.

Die Mausbedienung ist trotz der in [Kulzhabayev 2012] vorgestellten Optimierungen immernoch ziemlich umständlich und fehlerträchtig. Es ist aber nicht Hauptziel

dieser Arbeit, die Mausbedienung benutzerfreundlicher zu machen. Dies geschicht nur im Rahmen der notwendigen Änderungen für die automatisierte Installation.

3.7. Datenträgerabbilder

Das Gastsystem wird bereits beim Aufruf des Emulators mit den jeweiligen Datenträgerabbildern bestückt. Diese liegen meist in der Form von Disketten- oder CD/DVD-Abbildern, selten direkt als ausführbare Datei vor. QEMU bietet über den QEMU-Monitor die Möglichkeit, Disketten und CDs auszuwerfen und neue Images zu laden. Diese Vorgehensweise ist jedoch abhängig vom Emulator, da die nötigen Kommandos im Emulator fest einprogrammiert sind. Das würde bedeuten, dass man die Aufnahmen nicht mit einem anderen Emulator nutzen kann, was die Langlebigkeit und universelle Verwendbarkeit der Aufzeichnungen beeinträchtigen würde. Um den Datenträgerwechsel von vorneherein zu vermeiden kann man folgendes versuchen

- Bei Diskettenabbildern kann man einfach alle Abbilder in einen Ordner[4] entpacken. Das Installationsprogramm läuft dann meist ohne Nachfragen durch. Teilweise ist das auch bei CD-Abbildern (ISO) möglich. Den Ordner kann man dann als Festplattenabbild einbinden.

- Bei wenigen Einzeldatenträgern lässt sich das Problem umgehen, indem man das emulierte System mit zwei getrennten (virtuellen) CD-Laufwerken ausstattet. Dann muss man die Installation aber meist auf den geänderten Laufwerksbuchstaben hinweisen. Dies ist aber ebenfalls kritisch – sofern die Laufwerksauswahl per Tastatur erfolgt – da nicht garantiert werden kann, dass in allen Systemen die gleichen Laufwerksbezeichnungen vergeben werden.

- Oft braucht man für die „typische“ Installation nur eine CD, für eine „vollständige“ Installation aber alle.

Der zweite Punkt fordert wieder eine Möglichkeit, während der Installation Benutzerinteraktionen vorzunehmen. Aus allen drei Punkten folgt die Notwendigkeit, zusätzlich zu den eigentlichen Abbildern und der Aufzeichnung der Installation auch noch Metainformationen bereitzustellen, wie die Abbilder in einem Emulator gemountet werden müssen, damit die Aufzeichnung funktioniert. Hierzu gehört neben der verwendeten Emulatorversion auch der genaue Aufruf. Häufig lassen sich defekte Datenträger zwar noch auslesen, aber die Dateien sind korrupt, insbesondere bei mechanisch beschädigten CDs und in den heute üblichen schnellen Laufwerken. Images

[4] oder eine entsprechende Ordnerstruktur wie DISK1, DISK2, ...

müssen daher vor der endgültigen Archivierung sorgfältig getestet werden um zu vermeiden, dass man später im Archiv defekte Datenträger vorfindet.

Für diese Arbeit wurden Skripte genutzt, die ein virtuelles Image erzeugen und dann die Programmdateien dorthinein injizieren können. FAT16 Images müssen aber nach Experimenten mindestens 8 MB, laut anderen Quellen[5] mindestens 32 MB groß sein. Kleinere Images ließen sich zwar erzeugen, aber nicht mounten. Aus diesem Grund sollten die Programme als reine Dateien archiviert werden und zur Verwendung automatisch in ein Image injiziert werden.

[5] Wikipedia (FAT16), http://de.wikipedia.org/wiki/File_Allocation_Table, 07.07.2012

4. Umsetzung

Am Ende hängen wir doch ab
Von Kreaturen, die wir machten.

(Johann Wolfgang von Goethe)

Nachdem im letzten Kapitel die Anforderungen an die neue Komponente festgestellt wurden, wird in diesem Kapitel die praktische Umsetzung besprochen. Im Abschnitt Design wird ein Dateiformat für die Metadaten eingeführt und beschrieben wie der Workflow für die Komponenten aussieht. Darauf aufbauend wird der vorgestellte Entwurf im Abschnitt Implementation in das existierende Projekt VNCplay implementiert. Zum Schluss erfolgt eine Evaluation der vorgestellten Software.

4.1. Design

4.1.1. Formate für Metadaten und Aufzeichnungen

Das Format XML[1] ist für die Speicherung der Metadaten prädestiniert. XML-Dokumente lassen sich mit heutigen Programmiersprachen sehr effizient verarbeiten, da es in allen wichtigen Programmiersprachen direkt Bibliotheken zur XML-Verarbeitung gibt. Gleichzeitig sind sie aber auch für Menschen durch aussagekräftige Tags und Kommentare leicht verständlich. Über ein XML-Schema lässt sich exakt festlegen wie der Datensatz aufgebaut sein muss, welche Kodierung er benutzt und welchen Datentyp die Felder haben. Bei der Erzeugung kann so auf Validität geprüft werden. Durch diese Eigenschaften ist gewährleistet, dass auch in Zukunft die darin abgelegten Informationen nicht verloren gehen. Auch existierende Standards nutzen XML. Die Struktur der Metadaten für diese Arbeit ist als Beispiel im folgenden Listing gegeben.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <recording>
3  <description>
```

[1] eXtensible Markup Language, http://www.w3.org/XML, 05.07.2012

```
  <recordingName>WP6_win3.11</recordingName>
  <emulator>QEMU 1.0</emulator>
  <contributorName>Max Mustermann</contributorName>
  <timestamp>2012-04-07 11:30</timestamp>
  <UserInteractions>1</UserInteractions>
  <infoRequired>License</infoRequired>
  <comments>qemu-system-i386 images/win_311_wp.qcow2 -cdrom
     software/win311/wp6.0.iso -vnc :0 -snapshot</comments>
 </description>
 <installs>
   <app>
        <name>ACDSee</name>
        <version>1.01</version>
        <language>English</language>
        <build>unknown</build>
  </app>
  <os>
        <name>Microsoft Windows for Workgroups</name>
        <version>3.11</version>
        <language>German</language>
  </os>
  <platform>x86</platform>
  <implicitDepend>Graphics->256colors</implicitDepend>
</installs>
</recording>
```

Die Metadaten gliedern sich in zwei Abschnitte. Der erste Abschnitt enthält Informationen zur Aufzeichnung selbst, während *Installs* Informationen zu Software, Betriebssystem und Hardwareplattform enthält, für welche die Aufzeichnung erzeugt wurde. Abgespeichert werden die Metadaten unter dem gleichen Dateinamen wie die zugehörige Aufzeichnung, nur mit einer anderen Dateierweiterung.

Durch die hier vorgestellte Form der Metadaten ist die Aufzeichnung weitgehend unabhängig von Betriebssystem, Speicherorten und Dateinamen. Darauf wird im folgenden Abschnitt weiter eingegangen. Wegen der SyncPoints muss die Sprachversion während der Aufzeichnung und der Wiedergabe zwingend übereinstimmen. Bei der Programm- und Betriebssystem-Version sollte aber eine gewisse Unschärfe miteinbezogen werden, denn nur die wenigsten Sicherheits-Updates haben Einfluss auf die Benutzeroberfläche und bei der Fülle von solchen Patches scheint eine Sicherung aller Versionen nicht möglich. Nichtsdestotrotz sollte eine genaue Versionsangabe (Buildnummer) in den Metadaten enthalten sein, damit im Fehlerfall eine einfache Rekonstruktion möglich ist. Aus dem gleichen Grund wird auch die Kommandozeile

des Emulators als Kommentar mit in die Metadaten aufgenommen. Sie enthält in seltenen Fällen Schalter, mit denen sich bestimmte Hardware erzwingen lässt, beispielsweise eine USB-Maus statt der standardmäßigen PS/2-Maus oder ein größerer Speicherausbau.

Für die Aufzeichnungen selbst eignet sich XML nicht, da sie viel zu umfangreich sind. Zusätzliche XML-Tags würden noch weitaus größere Dateien erzeugen. Für die Aufzeichnungen wird daher das in Abschnitt 2.4 vorgestellte Format weiter verwendet. Die einzige Veränderung ist der Hinweistext, welcher in den Stages abgelegt wird. Dazu muss die Wiedergabe-Komponente aber so modifiziert werden, dass sie bei diesem Hinweistext nicht mehr Leerzeichen, sondern Anführungszeichen (") als Trennzeichen nutzt. Dies wird aber ausschließlich beim Hinweistext so gehandhabt, da sich sonst das gesamte Format der Aufzeichnungen ändern würde.

4.1.2. Speicherung des Archivs

Für die automatisierte Wiedergabe ist der Speicherort von Basisimages, Software und Aufzeichnungen maßgeblich. In den Metadaten der Aufzeichnung steht nur, welche Bestandteile man benötigt, nicht aber wo diese konkret gespeichert sind. Eine Suche oder Indizierung der abgelegten Dateien ist fehlerträchtig und zeitaufwändig. Eine Datenbank dagegen bringt das Problem mit sich, dass man in Zukunft die Datenbankdateien vielleicht nicht mehr lesen kann, denn sie sind vergleichsweise komplex aufgebaut. Ideal wäre eine Möglichkeit, direkt zu *berechnen* wo die jeweiligen Dateien abgelegt sind. Dies wird durch eine Hashfunktion realisiert, die alle notwendigen Parameter auf geeignete Weise übermittelt bekommt. Als Ergebnis liefert sie dann den Dateinamen der jeweiligen Komponente.

Algorithmus 4.1 Hashfunktion für die Berechnung der Dateinamen

```
function FILENAME(manufacturer, product, language, version)
    manufacturer ← trim(manufacturer)
    product ← trim(product)
    language ← trim(language)
    version ← trim(version)
    string ← manufacturer + product + language + version
    return md5(string) + ".log";
end function
```

Die Funktion *trim* entfernt unnötige Leerzeichen am Anfang und Ende der Bezeichner. Anschließend werden die Strings konkateniert und der Hashfunktion übergeben.

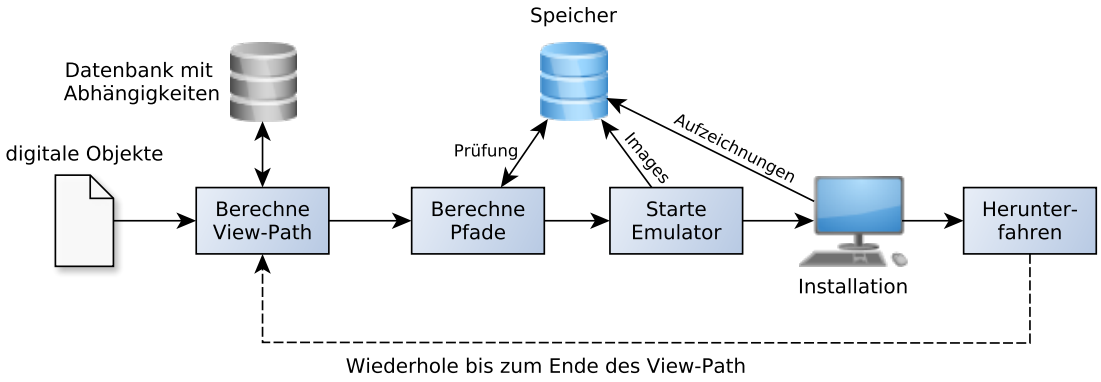


Abbildung 4.1.: Erzeugung einer Ablaufumgebung

Es muss aber darauf geachtet werden, dass die Bezeichner im gesamten Softwarearchiv identisch sind und nicht z.B. „Microsoft Corporation" parallel zu „Microsoft" verwendet wird. Als Hashverfahren wird hier beispielhaft MD5 eingesetzt. Das Verfahren ist gut dokumentiert und die mangelnde Sicherheit von MD5 spielt hier keine Rolle. Die Images und Aufzeichnungen liegen entweder lokal oder im Netzwerk unter einem bestimmten Pfad. Unterhalb dieses Pfads können sie optional nach Hersteller in eine Ordnerstruktur gegliedert werden. Die eigentlichen Dateien haben als Bezeichner nur einen Hashwert.

4.1.3. Der Installations-Workflow

Gegeben ist der Name einer Anwendung, ihre Version und Sprache. Diese Daten werden mithilfe einer Datenbank bestimmt, die zum Beispiel den Dokumenttyp analysiert oder (bei Computerspielen) die Systemanforderungen. Das alles läuft auf einer abstrakten Ebene wie in Abbildung 4.1 dargestellt ab

1. Berechne den View-Path für ein gegebenes digitales Objekt

2. Installiere die jeweilige Anwendung

   - Lokalisiere die benötigten Komponenten
   - Rufe den Emulator mit den ermittelten Pfaden auf
   - Starte die Wiedergabe mit der ermittelten Aufzeichnung
   - (Optional) Benutzerinteraktionen während der Installation

- Beende die Installation, fahre das Betriebssystem herunter

3. Wiederhole Schritt 2 bis der View-Path vollständig ist.

4. Betrachte die digitalen Objekte mithilfe der Ablaufumgebung

Das Wissen über die notwendigen Schritte einer Installation muss in den Metadaten zu dieser Aufzeichnung abgelegt werden, damit man von vorneherein weiß, welche Phasen vorkommen und welche Daten man dafür benötigt.

In der ersten Phase der obigen Aufzählung werden die zu betrachtenden digitalen Objekte analysiert und anhand einer Datenbank von Abhängigkeiten ein View-Path berechnet für den alle notwendigen Images und Aufzeichnungen vorliegen. Gibt es mehrere Möglichkeiten, wird die kürzeste gewählt. Die Bestimmung des View-Path geschieht unter Einbeziehung der Abhängigkeiten, die aus einer Datenbank wie der des KEEP-Projekts stammen und den impliziten Abhängigkeiten aus dem Feld *implicitDepend* der Metadaten. Die Abhängigkeiten müssen in einem standardisierten Format vorliegen, das von der Datenbank ausgewertet werden kann. Im Beispiel deutet der Pfeil an, dass die Grafik (mindestens) 256 Farben haben muss. Mehrere implizite Abhängigkeiten sollten durch Kommata getrennt werden.

Anschließend beginnt das Aufspielen der einzelnen Teile des View-Path. Anhand der vorgestellten Hashfunktion werden die Pfade zu Aufzeichnungen, Basisimages und Anwendungen bestimmt. Danach wird der Emulator mit dem Betriebssystem-Basisimage und den für die Installation nötigen Abbildern gestartet. Die Aufzeichnug läuft automatisiert ab. Benutzerinteraktionsphasen sind optional und werden in den Metadten angegeben.

Am Ende der Aufzeichnung muss das Betriebssystem heruntergefahren werden, da nur so das eingehängte Datenträgerabbild geändert werden kann, ohne die Methoden des QEMU-Monitors zu nutzen.

4.1.4. Zeitlicher Ablauf der Wiedergabe

Während der eigentlichen Installation, d.h. der Wiedergabe der gewählten Aufzeichnungen, sind Benutzerinteraktionsphasen möglich. Abbildung 4.2 zeigt den zeitlichen Ablauf einer solchen Phase. Die Abbildung gliedert sich in drei Phasen, die letzteren beiden sind optional, sie werden nur gebraucht, wenn während der Wiedergabe der Installationsaufzeichnung Benutzereingaben ermöglicht werden sollen. Es ist aber auch möglich, mehrere solcher Phasen mit Benutzerinteraktionen festzulegen. Beginn und

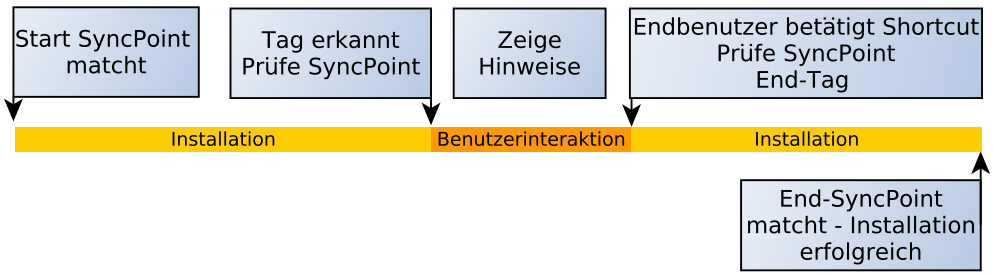


Abbildung 4.2.: Zeitlicher Ablauf der Wiedergabe einer Installation

Ende der einzelnen Phasen sind durch Vor- und Nachbedingungen festgelegt und werden in der Aufzeichnung durch Stufen (Stages) eingeleitet bzw. beendet. Die Phase *Installation* endet, sobald nach der Stufe *UserInteractionBegin* ein SyncPoint übereinstimmt. Dadurch wird gewährleistet, dass wirklich der richtige Dialog sichtbar ist. Die Position der Maus wird abgespeichert, um später wiederhergestellt werden zu können.

Der während der Phase *Benutzerinteraction* angezeigte Hinweis enthält Informationen über die geforderte Eingabe, und darüber ab wann der Benutzer die Tastenkombination zum Beenden der Phase betätigen darf. Dadurch wird vermieden, dass er dies zu früh tut, woraufhin der SyncPoint nicht übereinstimmt. Der Hinweistext kann zu Beginn der Aufzeichnung in einem Dialog eingegeben werden, oder danach manuell in die Aufzeichnungen geschrieben werden.

Die Phase *Benutzerinteraktion* endet wiederum, sobald drei Bedingungen erfüllt sind. Zuerst muss der Benutzer aktiv – d.h. durch eine Tastenkombination bestätigen, dass er mit der Eingabe fertig ist. Die zweite Bedingung ist erneut ein SyncPoint. Ohne diese beiden Bedingungen könnte es passieren, dass der Benutzer noch garnicht mit der Eingabe fertig ist, oder versucht, im falschen Dialog den Interaktionsmodus zu beenden. Zwischen den beiden Stufen-Markierungen befinden sich *immer* genau zwei SyncPoints. Dafür wird die Erstellung des SyncPoints erzwungen, auch wenn sich der Inhalt nicht geändert hat. Das sollte aber in der Praxis nicht auftreten, siehe Abschnitt 3.5. Anders lässt sich der Ablauf nicht deterministisch vorhersagen.

4.2. Implementation

Folgende Klassen des existierenden VNCplay wurden für diese Arbeit angepasst und erweitert. Eine Beschreibung der wichtigsten Methoden folgt in diesem Abschnitt. In

Anhang A findet sich eine Funktionsreferenz.

- *VncViewer* – die Hauptklasse, Steuerung von Aufnahme und Wiedergabe
- *VncCanvas* – enthält das Zeichnen den Bildschirminhalts und die Verarbeitung von Maus- und Tastaturereignissen
- *VncInputRecorder* – Aufnahmekomponente
- *VncInputReplayer* – Wiedergabekomponente

Darüberhinaus wurde die Klasse *MessagePanel* hinzugefügt, sie erzeugt eine Box zum Anzeigen des Hinweistextes.

4.2.1. Zustände

Es wurde eine Zustandssteuerung implementiert, die bestimmt, ob Eingaben erlaubt sind, oder nicht und ob aufgenommen oder wiedergeben wird. Es gibt insgesamt sechs Zustände. Der Startzustand 0 ist bei Aufnahme und Wiedergabe identisch. Die Zustände 1 bis 3 gehören zur Aufnahme-, 4 und 5 zur Wiedergabekomponente. Zustand 2 wird von beiden Komponenten genutzt, er stellt den Pause-Modus dar, in dem keine Eingaben möglich sind. Dies dient zum Blockieren von Eingaben in bestimmten Situationen.

- 0 – initialer Zustand
- 1 – Aufnahme
- 2 – Pause, keine Eingaben möglich.
- 3 – Benutzerinteraktionsmodus. Eingaben möglich, aber keine Aufnahme.
- 4 – Wiedergabe, keine Interaktionen des Benutzers möglich.
- 5 – Benutzerinteraktionsphase während der Wiedergabe.

Diese Zustände sind ein wesentlicher Unterschied zum bisherigen VNCplay, wo die Verarbeitung von Interaktionen auf Funktionen wie *isRecording* basierten. Dadurch wurde es unmöglich, einen Zustand zu realisieren, in dem zwar nicht aufgenommen wird, aber trotzdem Interaktionen verarbeitet werden. Die existierenden Methoden wurden an die neuen Zustände angepasst. Die endlichen Automaten (DFA) in den

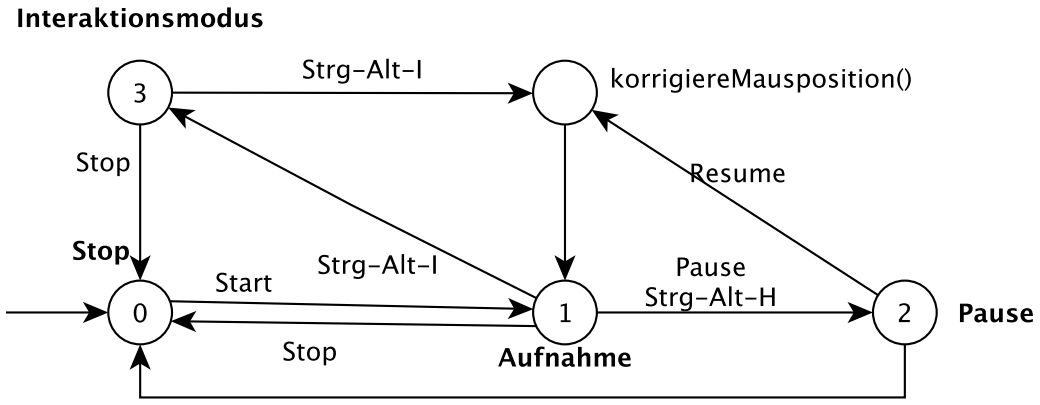


Abbildung 4.3.: Endlicher Automat, der die Zustandsübergänge bei der Aufzeichnung beschreibt

Abbildungen 4.3 und 4.4 beschreiben die Zustandsübergänge bei der Aufnahme und Wiedergabe. Es folgt zuerst eine Beschreibung der Zustände während der Aufzeichnung.

In *Zustand 0* befindet sich die Aufnahmekomponente direkt nach der Initialisierung. Betätigt der Benutzer nun den Start-Button, so beginnt *Zustand 1*. Hier wird alles aufgezeichnet, die Maus ist dabei im Fenster des Gastsystems gefangen und kann dieses nicht verlassen. So werden ungewollte Interaktionen mit dem Hostsystem vermieden. Dieser Zustand entspricht bis auf die Mausbarriere der Aufnahmekomponente des bisherigen VNCplay. Die Mausbarriere ist deshalb sinnvoll, weil die Maus nun nicht mehr ungewollt mit dem Hostsystem interagieren kann. Das Fangen der Maus wurde in der Funktion *mouseExited* realisiert, indem eine Instanz der Java-Klasse *Robot* den Hostzeiger beim Verlassen des Fensters zur zuletzt bekannten Position bewegt, die dazu von *mouseMoved* immer abgespeichert wird.

Aus Zustand 1 kann man (abgesehen von Zustand 0) in zwei Zustände wechseln. *Zustand 2* stellt den Pausemodus dar. Hier wird die Aufnahme angehalten, die Eingaben im Gastsystem deaktiviert und die Maus befreit. Das Gastsystem lässt sich also nicht mehr bedienen, Berechnungen oder Installationen laufen aber weiter. Dieser Zustand wird benötigt, wenn der Benutzer Interaktionen im Hostsystem vornehmen will. Er wird aber auch während der Mauskorrektur benutzt, um Eingaben zu verhindern.

In *Zustand 3* dagegen wird zwar die Aufnahme angehalten, jedoch nicht die Benutzereingaben. Man kann das Gastsystem hier also völlig normal bedienen, nur das keine Aufzeichnungen gemacht werden. Dies dient dazu, Installationen durchzuführen, bei denen der Endanwender selbst bestimmte Dialoge bedienen muss. Dieser

Modus wird *UserInteractionMode* genannt. Sowohl bei der Aufzeichnung als auch bei der Wiedergabe gibt es einen Zustand dieses Namens. Bei der Rückkehr von Zustand 3 zu Zustand 1 wird die Methode *correctMousePos* (siehe Algorithmus 4.4) aufgerufen. Sie sorgt dafür, dass der Mauszeiger sich an der gleichen Position befindet, wie vor eintritt in die Benutzerinteraktionsphase.

Die Wiedergabe-Komponente (Abb. **??**) besitzt drei Zustände, wobei der Zustand 0 mit dem der Aufnahmekomponente übereinstimmt. In *Zustand 4* dagegen läuft die Wiedergabe, Eingaben sind nicht möglich. In *Zustand 5* sind Eingaben erlaubt. Genau wie bei der Aufnahme muss auch bei der Wiedergabe der Mauszeiger zurückgesetzt werden, da darauffolgende Mausinteraktionen in der Aufzeichnung nicht das gewünschte Resultat haben könnten.

4.2.2. Beschreibung der Methoden

Die Steuerung der Aufnahme wurde in den Methoden *pauseRecording* und *resumeRecording* in der Hauptklasse *VncViewer* implementiert. Beide sind abhängig von einer booleschen Variable, die bestimmt, ob die Eingaben angehalten werden sollen. Existierende Methoden mussten so modifiziert werden, dass sie Benutzerinteraktionen erlaubt, die nicht aufgenommen werden. Dies wurde mit der vorgestellten Zustandssteuerung realisiert. Aufgenommen wird ausschließlich, wenn der Zustand 2 vorliegt. Dies wird dann in der Aufzeichnung durch die Stufen „UserInteractionBegin“ bzw. „-End“ gekennzeichnet, wie in folgendem Listing erkennbar.

```
1 40 SID=0 title=UserInteractionBegin when=1338145108369 type=stage
     msg="Hinweis"
2 72 type=sync px305y229=-1 px305y224=-1 px326y245=-16777048 [...]
3 50 type=sync type=sync px305y229=-1 px305y224=-1 [...]
4 40 SID=1 title=UserInteractionEnd when=1338145109437 type=stage
5 9999 id=503 button=0 when=1338145128581 modifiers=10 type=java.
    awt.event.MouseEvent y=0 x=0
```

Ein Problem entsteht dadurch, dass die Benutzerinteraktions-Phase bei der Wiedergabe nicht zwingend die gleiche Zeit benötigt wie bei der Aufnahme. Dadurch kann es sein, dass der Endbenutzer lange warten muss, bis auf dem Bildschirm eine neuerliche Aktivität erkennbar ist. Um das zu vermeiden schreibt die Aufnahmekomponente in die erste Zeile nach der Stufe *UserInteractionEnd* ein Delay von 9999. Die Wiedergabekomponente erkennt das und ersetzt den Wert durch die tatsächlich gemessene Dauer.

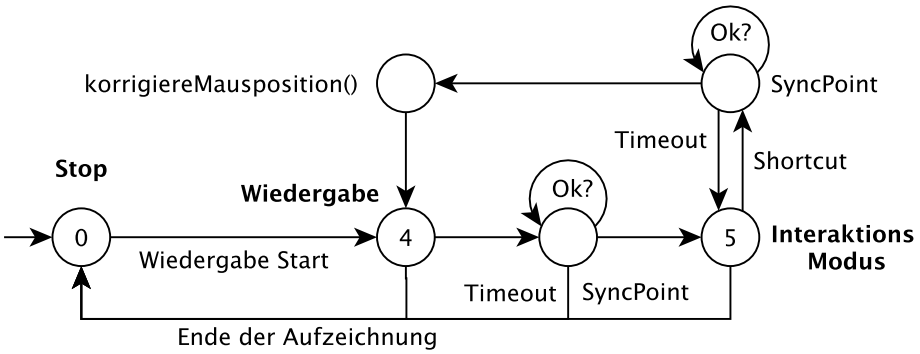


Abbildung 4.4.: Endlicher Automat, der die Zustandsübergänge bei der Wiedergabe beschreibt. Es fehlt aus Gründen der Übersichtlichkeit die Überprüfung der Stages.

Außerdem wurden Methoden implementiert, die ein einfaches Erzeugen eines SyncPoints an einer bestimmten Bildschirmposition erlauben. So lässt sich mit einem SyncPoint in der linken, oberen Ecke gezielter eine geöffnete Vollbildanwendung verifizieren, während ein SyncPoint in der Bildschirmmitte besser für Dialoge geeignet ist. Auch eine Erzeugung an der Position des Mauszeigers ist vorgesehen, aber nicht so praktikabel (siehe Abschnitt 4.2.3). Zur Steuerung werden zusätzlich zu den vorhandenen Buttons Tastenkombinationen eingesetzt. Das Handling der Tastenkombinationen wurde in die Hauptklasse *VncViewer* eingebaut. Folgende Tastenkombinationen wurden implementiert.

Strg-Alt-H wechselt in den Pause-Modus

Strg-Alt-I beginnt eine Benutzerinteraktion und beendet sie. Der gleiche Shortcut wird auch bei der Wiedergabe benutzt um aus dem UserInteractionMode wieder herauszukommen.

Strg-Alt-M erzeugt einen SyncPoint an der Mausposition

Der Emulator QEMU nutzt die Kombinationen Strg-Alt-1 und -2 für den Wechsel zum QEMU-Monitor. Wegen der nicht ganz zu vermeidenden Wechselwirkungen mit Programmen im Gastsystem werden Shortcuts, die Strg-Alt enthalten, nicht mehr an den Emulator durchgeleitet. Sie werden nur von VNCplay verarbeitet. Die Einschränkung dadurch ist vernachlässigbar, denn auf solchen speziellen Tastenkombinationen liegen meistens ebenso spezielle Funktionen, die man meist ebensogut über das Menü erreichen kann.

Bei der Wiedergabe waren viele Anpassungen des bestehenden Codes notwendig, da sich hier der Ablauf grundlegend ändert. Bisher wurde die Wiedergabe komplett

von einem Timer gesteuert, der alle 20ms ein Ereignis erzeugt. Diese Ereignisse werden von der Methode *actionPerformed* verarbeitet. Der komplett automatische Ablauf muss nun unterbrochen werden, damit überhaupt ein Stoppen der Wiedergabe möglich ist. Davor muss aber geprüft werden, ob die Bedingungen (Stages, SyncPoints, Tastatur-Shortcut) eingehalten werden, wie in Abbildung 4.2 dargestellt.

Das wurde erreicht, indem in *actionPerformed* entsprechende Abfragen stattfinden und dann als boolesche Variablen abgelegt werden. Stimmt ein SyncPoint nicht überein, so wird wiederholt die Methode *checkSync* aufgerufen, bis er übereinstimmt oder ein Timeout erreicht wurde. Die Methode *checkSync* setzt dabei die Variable *syncWait* auf false sobald ein SyncPoint übereinstimmt. Sind alle Vorbedingungen erfüllt, so wird *userInteraction* aufgerufen, der Timer angehalten und Benutzereingaben erlaubt.

Zum Beenden der Interaktionsphase bei der Wiedergabe wird ebenfalls die Tastenkombination Strg-Alt-I betätigt. Dadurch wird der Timer wieder aktiviert und *actionPerformed* beginnt damit den SyncPoint zu prüfen. Stimmt der SyncPoint nicht überein, so wird nur ein vergleichsweise kurzer Timeout abgewartet und es wird nicht die gesamte Wiedergabe abgebrochen, sondern in die Benutzerinteraktionsphase zurückgekehrt. Denn in diesem Fall hat meistens der Benutzer im falschen Dialogschritt den Shortcut betätigt. In Algorithmus 4.2 findet sich der Pseudocode für the Methode *actionPerformed*. Die While-Schleife am Ende blieb praktisch unverändert. Sie übernimmt die Verarbeitung aller Teile der Aufzeichnung ohne Benutzerinteraktion. Die Methode *userInteraction* ist in Algorithmus 4.3 dargestellt. Sie steuert die Anzeige der Nachricht, das Korrigieren der Maus, das Setzen der entsprechenden booleschen Variablen (und Zustände) und nicht zuletzt das Stoppen des Timers. Der Timer wird dann wieder gestartet, sobald der End-SyncPoint überprüft wird. Die Variable *mode* wird auf 1 gesetzt, wenn die aktuelle Zeile die Stage *UserInteractionBegin* enthält und auf 2, wenn die die Stage *UserInteractionEnd* enthält. Bei allen anderen Zeileninhalten hat sie den Wert 0.

4.2.3. Mauskorrektur

Beim Fortsetzen der Aufnahme nach der Benutzerinteraktions-Phase und auch beim Fortsetzen der Wiedergabe muss der Mauszeiger an der gleichen Position sein, wie vor Eintritt in diese Phase. Sonst kann nicht garantiert werden, dass spätere Mausinteraktionen alle an der vorgesehenen Stelle erfolgen. Folgende Zitate belegen die Schwellwerte, die auch in der vorliegenden Arbeit verwendet wurden.

> Windows 3.11 for Workstations processed events correctly when pause was 5ms [Kulzhabayev 2012, S. 24].

Algorithmus 4.2 Verarbeitung der Timer-Ereignisse

```
procedure actionPerformed(TimerEvente)
    if syncWait then
        if time > maxTimeout ∧ ¬UImode then
            stopTimer()
            stopPlayback()
        end if
        if time > maxTimeout ∧ UImode then
            UIModeEndShortcut ← false
            syncWait ← false
            stopTimer()
            return
        end if
        if (UIModeBeginStage ∧ UIModeBeginSync) ∨ (UIModeEndShortcut ∧
¬UIModeEndSync) then
            checkSync()  ▷ syncWait ← false, wenn wenn SyncPoint übereinstimmt
        end if
    else if ¬syncWait then
        if UIModeBeginStage ∧ UIModeBeginSync then
            userInteraction(true)
        end if
        if UIModeEndShortcut ∧ ¬UIModeEndSync then
            if ¬backToUIMode then
                nextLine()                          ▷ Gehe zum zweiten SyncPoint
            else      ▷ Wenn true, teste den SyncPoint, wenn er nicht übereinstimmt,
kehre zurück in den UIModus
            end if
        end if
        if UIModeEndShortcut ∧ UIModeEndSync then
            userInteraction(false)
        end if
        while ¬syncWait ∧ mode = 0 do
                                ▷ Hauptschleife, verarbeitet die restliche Aufzeichnung
            doEvent()                               ▷ Verarbeitet die Interaktionen
        end while
    end if
end procedure
```

Algorithmus 4.3 Start und Ende des Benutzerinteraktionsmodus

```
procedure USERINTERACTION(boolean enable)
    if enable then
        UIModeBeginSync ← true
        if message != null then
            showMessage(message)
        end if
        correctMousePos()
        stopTimer()
        setState(5)
        UIMode ← true
        UIModeBeginStage ← false
        print(UserInteractionModeBegin)
    else if ¬enable then
        removeMessage()
        UIMode ← false
        mode ← 0
        UIModeBeginStage ←
        UIModeBeginSync ← false
        UIModeEndShortcut ← false
        UIModeEndSync ← false
        backToUIMode ← false
        correctMousePos()
        setState(4)
    end if
end procedure
```

> OSes as Windows 95 and 98 processed events properly when delay between each event was 50ms [Kulzhabayev 2012, S. 24].
>
> When there was heavy load of CPU or I/O operations over primary disk of the host system, it was necessary to increase time of the pause between each event for about 10ms. [Kulzhabayev 2012, S. 24]
>
> One pointer movement from one position to another cannot exceed 126 pixels. If it exceeds, then the pointer just stops after reaching that threshold [Kulzhabayev 2012, S. 30]

Die Bewegung muss also in eine Anzahl Schritte der Länge 126 zerlegt werden, anschließend folgt (meistens) ein einzelner Schritt der Länge kleiner 126 zum Ziel. Aus

Gründen der Einfachheit wird hier eine maximale Schrittlänge von 100 Pixeln benutzt. Als Problem stellte sich heraus, dass den Hostzeiger den Gastzeiger ungewollt beeinflusst. Wenn man den Hostzeiger zu schnell bewegt, so fängt der Gastzeiger an zu springen und sich unvorhersehbar zu bewegen. Als Lösungsansatz dient eine Zerlegung der beiden Bewegungen. Zuerst wird der Gastzeiger an die Zielposition bewegt, dann folgt der Hostzeiger nach, wobei aber in den Pause-Modus gewechselt wird. Auf diese Weise sollte eine zuverlässige Bewegung beider Zeiger an eine beliebige Position möglich sein. Der Pseudocode des implementierten Algorithmus findet sich in Algorithmus 4.4.

Algorithmus 4.4 Korrektur der Mausposition

```
procedure correctMousePos
                                        ▷ Die Schritte sind maximal 100 Pixel groß.
    stepsX ← ⌊(save.X − last.X)/100⌋
    stepsY ← ⌊(save.Y − last.Y)/100⌋
    repeat
        if stepsX > 0 ∧ stepsY = 0 then
            tempX ← last.X ± 100
            tempY ← last.Y
            stepsX ← stepsX − 1
        else if stepsX = 0 ∧ stepsY > 0 then
            tempX ← last.X
            tempY ← last.y ± 100
            stepsY ← stepsY − 1
        else
            tempX ← last.X ± 100
            tempY ← last.y ± 100
            stepsX ← stepsX − 1
            stepsY ← stepsY − 1
        end if
        guestPointerMove(tempX, tempY)
        Sleep(60ms)
    until stepsX = 0 ∨ stepsY = 0
    moveGuestPointer(save.X, save.Y)
    setState(2)                         ▷ Blockt alle Eingaben
    hostPointerMove(save.X, save.Y)
end procedure
```

Die Bewegung des Gastzeigers geschieht in der Implementation durch ein *MouseEvent*, welches danach direkt an die Methode *mouseMoved* übergeben wird. Das bedeutet, dass hier ein künstliches Mausereignis generiert wird, welches *nicht* von

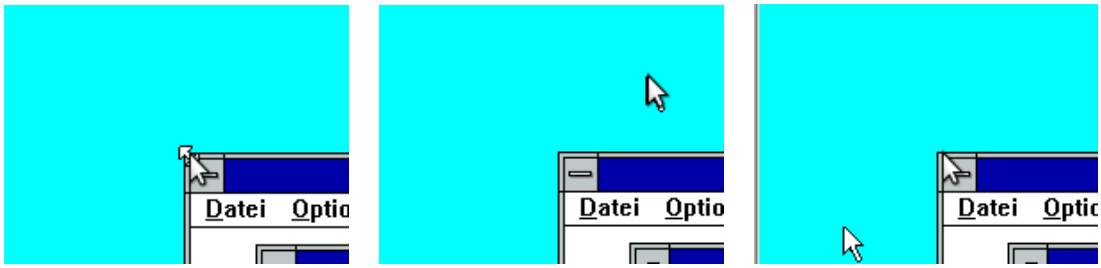
Abbildung 4.5.: Dialogfeld von Microsoft Windows 95 ohne die fragliche Option zur Mausbeschleunigung. Diese ist normalerweise im unteren Bereich zu finden.

der physikalischen Maus ausgeht.

Ist der Gastzeiger an der richtigen Stelle, wird die Eingabe angehalten und der Hostzeiger wird mit Hilfe der Klasse *Robot* an die gleiche Position bewegt. So kann die ruckartige Bewegung durch den Robot nicht den Gastzeiger beeinflussen. Umgekehrt funktioniert das nicht. Wird zuerst der Hostzeiger bewegt, so wird der Gastzeiger durch die Distanz zwischen Host- und Gastzeiger beeinflusst, sobald die Eingaben wieder angeschaltet sind. Man sollte also mit dem Hostzeiger immer nur dorthin springen, wo der Gastzeiger sich befindet.

Das Verfahren mit dem Robot wird auch genutzt, um die Benutzerfreundlichkeit von VNCplay zu verbessern. So springt der Mauszeiger nach einer Pause beim Eintritt in das *VncCanvas*-Element von selbst an die zuletzt gespeicherte Position. Auch das Fangen der Maus am Fensterrand in der Methode *mouseExited* nutzt diese Technik.

Es stellte sich jedoch heraus, dass die Generierung künstlicher Mausereignisse auf diese Weise nicht zuverlässig funktioniert. Auch die Ergebnisse von [Kulzhabayev 2012] halfen nur begrenzt weiter. Die künstlichen *MouseEvents* haben zwar eigentlich die richtige Zielposition, eine maximale Schrittlänge von 100px und einen zeitlichen Abstand von 60ms. Dennoch bewegt sich der Gastzeiger unter Microsoft Windows



(a) Ausgangsposition (b) vor dem Zurücksetzen (c) nach dem Zurücksetzen

Abbildung 4.6.: Darstellung der Sprünge bei der Generierung künstlicher Mausbewegungen. Der Gastzeiger befindet sich im rechten Bild unten.

3.11 um etwa den Faktor 1,8 bis 2,0 zu weit. Dies ist in Abbildung 4.6 illustriert. Unter windows 9x verhielt sich der Mauszeiger viel träger, es war kein Problem, den Gastzeiger mit dem Hostzeiger zu überholen. Dadurch müssen allerdings alle Mausbewegungen sehr langsam ausgeführt werden, was die Laufzeit der Aufzeichnungen erhöht.

Ein Zusammenhang mit der möglicherweise aktivierten Mausbeschleunigung konnte nicht abschließend geklärt werden, denn die Option dafür zeigt das Betriebssystem mit dem Standardtreiber nicht an, wie in Abbildung 4.5 zu erkennen. Darauf wird in [Kulzhabayev 2012] nicht eingegangen. Versuche mit einer anderen emulierten Maus (PS/2, seriell, USB) brachten keinen Erfolg, genauso wie Versuche mit extrem langen Wartezeiten oder besonders kurzen oder langen Intervallen. Ein Zusammenhang mit QEMU lässt sich nicht ganz ausschließen.

Weil das Hauptinteressie dieser Arbeit nicht auf der Untersuchung betriebssystemspezifischer Mausprobleme liegt und weil der Nutzen solcher Untersuchungen in der digitalen Langzeitarchivierung sowieso begrenzt ist, wurde die Rücksetzung des Mauszeigers an eine bestimmte Position nicht weiterverfolgt. Stattdessen wird der Mauszeiger nun einfach immer beim Eintritt und Verlassen der Benutzerinteraktionsphase in die linke obere Ecke bewegt. Dies funktioniert in allen untersuchten Betriebssystemen zuverlässig, hat jedoch den Nachteil, dass diese Bewegung auch aufgezeichnet wird.

4.3. Evaluation

In den vorhergehenden Abschnitten wurde ein Verfahren vorgestellt, das es erlaubt, beliebige Installationsvorgänge aufzuzeichnen und anschließend wiederzugeben. Da-

bei ist es möglich, während der Installationen auch Eingaben vorzunehmen. Dieses Verfahren soll nun evaluiert werden. Dazu werden ausgewählte Anwendungsinstallationen aufgezeichnet. Die Aufzeichnungen werden dann per Skript nacheinander auf ein Basisimage aufgespielt. Wurde das Skript erfolgreich abgeschlossen, wird anschließend getestet, ob die Programme funktionieren.

4.3.1. Getestete Anwendungen

Die Anwendungen stammen entweder aus der lehrstuhleigenen Sammlung oder aus dem Internet[2]. Möglichst viele Installationen sollten eine Möglichkeit zur Interaktion mit dem Benutzer haben, d.h. eine Abfrage der Lizenznummer, eine Wahl der Installationsvariante oder ähnliches.

Letztendlich getestet wurden folgende Programme, die sich auf der beigefügten DVD befinden.

Hersteller	**Name**	**Jahr**[3]	**Art**	**Getestet unter**
Adobe	Photoshop 2.56	1992	Bildverarbeitung	Windows 3.11
Autodesk	AutoCAD R13	1994	CAD-Software	Windows 3.11
Lotus	AmiPro 3.1	1994	Textverarbeitung	Windows 3.11
Microsoft	Excel 5.0	1995	Tabellenkalkulation	Windows 3.11
Microsoft	Word 6.0	1993	Textverarbeitung	Windows 3.11
ACD Systems	ACDSee 1.0	1994	Fotodatenbank	Windows 95
Adobe	Acrobat Reader 3.0	1995	PDF-Betrachter	Windows 95
Corel	PaintShop Pro 4.1	1997	Bildverarbeitung	Windows 95
Microsoft	Office 95	1995	Office-Paket	Windows 95
Netscape	Navigator 2.0	1996	Browser	Windows 95
Nullsoft	Winamp 2.79	2000	Multimediaplayer	Windows 98

Diese Auswahl bildet einen guten Durchschnitt der typischen Büroprogramme, mit denen viele proprietäre Dokumenttypen eingeführt wurden. Die Programme wurden auf dem Betriebssystem getestet, auf das zum Erscheinungszeitpunkt modern war. Grund dafür ist, dass nicht alle Programme ein unpassendes Betriebssystem erkennen können. So schloss sich der Adobe Reader unter Windows 3.11 kommentarlos. Dadurch ergeben sich jeweils fünf Tests unter Windows 3.11 und unter Windows 95. Windows 98 nur mit WinAmp getestet. Da es vom Handling her sehr ähnlich zu

[2]Software: Vetusware, http://www.vetusware.com, 23.06.2012
Treiber: QEMU OS Support List, http://www.claunia.com/qemu, 06.07.2012

Windows 95 ist, sollten die Aufzeichnungen aber in den meisten Fällen übertragbar sein.

4.3.2. Evaluation der Installation und Aufzeichnung

Zum Test der Aufzeichnung wurde die Anwendung auf ein Basisimage des jeweiligen Betriebssystems aufgespielt. Während der Installation wurden an allen Stellen, an denen das sinnvoll war, Benutzerinteraktionen eingesetzt. Dies war beispielsweise bei der Eingabe von Name und Organisation, der Lizenznummer und bei der Auswahl des Installationstyps der Fall. Jede Aufzeichnung wurde direkt im Anschluss noch einmal wiedergeben, um die grundsätzliche Funktionsfähigkeit sicherzustellen.

Die Installation von Adobe Photoshop konnte nicht evaluiert werden, da die Seriennummer einen Strich enthält. Dieser lässt sich nicht eingeben (Unknown Keycode). Verwendet man nur QEMU ohne VNCplay (und damit auch ohne Aufzeichnung), so funktioniert es aber. Die genannte Meldung erscheint nicht im Terminal von VNCplay, sondern in dem Terminal, in dem QEMU läuft. Es liegt also nahe, dass hier ein Problem mit dem RFB oder VNC-Protokoll vorliegt. ACDSee wurde trotz früherer Verfügbarkeit unter Windows 95 getestet, da es sich nicht ohne Weiteres unter Windows 3.11 mit seinen standardmäßig nur 16 Farben installieren ließ. Jedoch handelte es sich bei dem Programmpaket nicht um eine installierbare, sondern um eine direkt ausführbare Version. Der Netscape Navigator fragte bei der Installation immer wieder die Disketten-Verzeichnisse ab, man muss aber nur abnicken, da die Daten alle im selben Verzeichnis liegen. So entfällt der Wechsel von Images durch den QEMU-Monitor. Nach der Installation von WinAmp erschienen mehrere Fehlermeldungen mit Bezug auf fehlende Netzwerkfähigkeiten. Dies ist aber nicht verwunderlich, denn die Netzwerkschnittstelle war nicht eingerichtet.

Der Verwalter muss wissen, wie die Installation abläuft, damit er dann einen entsprechenden Hinweistext eingeben kann. Nachträglich kann dieser Text nur umständlich durch Editieren der Aufzeichnung bearbeitet werden. Da aufgrund der vielen Probleme bei Installationen diese ohnehin nicht immer beim ersten Versuch abgeschlossen werden können, ist die Einschränkung nicht so schlimm. In einem noch zu entwickelnden Framework für die digitale Langzeitarchivierung kann auch ein Editor für die Aufzeichnungen zur Verfügung gestellt werden.

Übunto 12.04 fängt als Gastsystem manche für das Hostsystem bestimmte Tastenkombinationen ab. Besonders störend war das bei der Windows-Taste, die normalerweise das Startmenü öffnen sollte. Dadurch musste man den in Windows 95 sehr trägen Mauszeiger von der linken oberen in die linke untere Ecke bewegen. Das

war fast so zeitaufwändig wie die Installation selbst. Generell werden aber sehr viele Sondertasten aufgrund des oben beschriebenen Fehlers garnicht an die emulierte Software übertragen. Letztendlich arbeiteten aber sämtliche Installationen – ausgenommen Photoshop – fehlerfrei.

4.3.3. Evaluation der Wiedergabe

Bei der Wiedergabe wurden alle Aufzeichnungen für ein bestimmtes Bestriebssystem per Skript auf ein Basisimage aufgespielt. Anschließend wurde die Funktionsfähigkeit der Programme getestet.

Alle zehn Wiedergaben funktionierten letztendlich fehlerfrei. Dafür waren allerdings in zwei Fällen mehrere Versuche nötig. Dies lag an Mausklicks, die – vermutlich wegen zu schneller Bewegungen – während ihrer Wiedergabe nicht die entsprechenden Dateien starteten. Andere Klicks in der gleichen Aufzeichnung funktionierten aber korrekt. Es ist möglich komplizierte Benutzerinteraktionen wie die Auswahl von Installationstypen durchzuführen. Getestet wurde das durch Aufzeichnung der „typischen“ Installation von AutoCAD, Microsoft Excel und Micorosft Word. Während der Wiedergabe wurde dann die benutzerdefinierte Installation gewählt. Wichtig ist in diesem Fall, dass man die Benutzerinteraktion an einer Stelle beendet, an der beide Varianten gleich aussehen, also auf keinen Fall in einem Dialog, der nur bei der „vollständigen“ Variante erscheint.

Bei der Wiedergabe entstanden trotz des in Abschnitt 4.2.2 vorgestellten Workarounds Pausen am Ende der Benutzerinteraktionsphase. Diese waren jedoch nicht zuverlässig reproduzierbar. Ein Zusammenhang mit den in den Aufzeichnungen enthaltenen Zeitstempeln ist nicht auszuschließen.

Die Verwalter müssen lernen, wann und wo sie SyncPoints erzeugen müssen. Dies ist vorallem wichtig, um Fehler frühzeitig abzufangen und entsprechend zu melden. Durch die SyncPoints wird die Aufzeichnung zuverlässiger und benutzerfreundlicher. Die Erkennung von Fehlermeldungen, wie sie gerade bei Installationen häufig auftreten wurde bereite in [Kulzhabayev 2012] besprochen. Generell kann man mit der vorgestellten Lösung jede gewünschte Software mit einem beliebigen Grad der Automatisierung aufspielen.

Die Auswertung der Metadaten konnte nicht evaluiert werden, da sich das gesamte Framework noch im Aufbau befindet. Für eine Evaluation der Metadaten bräuchte man Zugriff auf die Datenbank der Abhängigkeiten und eine Komponente zur Analyse von Dokumenttypen. Erst dann ließe sich die Auswertung der Metadaten wirklich unter praxisnahen Bedingungen prüfen.

5. Fazit und Ausblick

Wichtig ist, dass man nie aufhört zu fragen.

(Albert Einstein)

Die Evaluation im letzten Kapitel hat gezeigt, dass hier vorgestellte Komponente grundsätzlich funktioniert. Die bei der Aufzeichnung und Wiedergabe entstandenen Probleme beruhen vorallem auf der mangelnden Abschottung von Host- und Gastsystem und einem Fehler bei der Umsetzung von Tastaturcodes.

Das in dieser Arbeit skizzierte Framework vermag es mit variablem Automatisierungsanteil Ablaufumgebungen bereitzustellen. Dies ist ein wichtiger Baustein für ein noch zu entwickelndes, digitales Archivsystem mit dem in Zukunft jeder Mensch digitale Objekte der Vergangenheit betrachten und damit interagieren können soll. Auch bei der Migration von Dokumenten in der emulierten Umgebung ist der Benutzerinteraktionsmodus hilfreich. So kann man den Ziel-Dateityp auswählen ohne für jeden Dateityp eine eigene Sitzung aufnehmen zu müssen. Durch seine Unabhängigkeit von physikalischen Speicherorten, Betriebssystemen und Bedienkonzepten sollte das Framework in dieser Form sehr lange in Verwendung bleiben können. Dies ist für den Einsatz in der digitalen Langzeitarchivierung ausschlaggebend.

Neben dem aufwändigen Zusammentragen und Erfassen der Software und und ihrer Abhängigkeiten verbleiben noch einige Verbesserungen an der zur Aufnahme und Wiedergabe eingesetzten Software VNCplay. VNCplay wurde ursprünglich für Benchmarkzwecke entwickelt. Aus dieser Zeit stammen vermutlich die zahlreichen Bezüge zur aktuellen Systemzeit. Diese sind jedoch für den hier vorgestellten Einsatzzweck unnötig und stören in manchen Situationen den Ablauf. Das äußert sich durch Wartezeiten während der Wiedergabe. Man sollte diese Zeitstempel entfernen und den Ablauf ausschließlich durch den zeitlichen Abstand zwischen verschiedenen Interaktionen und durch SyncPoints steuern. Dazu ist aber ein weitgehender Umbau des Programmes notwendig.

Weiterhin sollte das emulierte System noch tiefgreifender vom Hostbetriebssystem abgekoppelt werden, ähnlich wie man das von Virtualisierern wie VMware kennt.

So sollte es auf keinen Fall vorkommen, dass Tasten(-Kombinationen) vom Gastbetriebssystem abgefangen werden. Auch die Maussteuerung ließe sich durch so eine Abkapselung eventuell weiter verbessern. Bei QEMU klappt das bereits. Setzt man jedoch QEMU zusammen mit VNCplay ein, werden die Systeme nicht mehr korrekt getrennt.

Ein Benutzerinterface würde die fehlerträchtige Steuerung per Shell-Skript überflüssig machen und könnte darüberhinaus direkt an eine Datenbank angeboten werden, sodass ein Benutzer direkt sieht, welche Programme und Basisimages zur Verfügung stehen. Die Installation kann dann optisch ansprechend mit einem Fortschrittsbalken dargestellt werden. Ein Editor für Aufzeichnungen macht nur mit den in [Kulzhabayev 2012] vorgestellten Bausteinen mit Hilfe von Stages Sinn. Dann könnte man die Aufzeichnungen nachträglich anpassen. So wäre zum Beispiel eine wiederverwendbare Aufzeichnung zum Herunterfahren sinnvoll.

Die digitale Langzeitarchivierung steckt noch in den Kinderschuhen, doch wird sie in Zukunft zu den normalen Medien in jeder Bibliothek zählen. Kritisch zu sehen ist in diesem Zusammenhang die Zunahme von Aktivierungs-Zwang bei aktuellen Softwareprodukten. Dadurch gestaltet sich der Archivierungsprozess erheblich schwieriger, da die Programme so mit der Hardware verdongelt werden. Auch die zunehmenden Bezugsmöglichkeiten aus dem Internet sind nicht nur positiv zu sehen, denn sie haben ein noch kürzeres Verfallsdatum als althergebrachte physikalische Datenträger. Apple verfolgt sogar den Einsatz seiner Betriebssysteme auf Nicht-Apple-Hardware. Dies würde rein rechtlich auch die Emulation betreffen.

Glossar

Abhängigkeiten bestehen zwischen unterschiedlichen Softwarekomponenten. Eine Softwarekomponente A kann nicht ohne eine Softwarekomponente B betrieben werden

Aufnahmekomponente Komponente von VNCplay, die für die Aufzeichnung zuständig ist. Erzeugt eine Aufzeichnung.

Aufzeichnung Eine einfache Textdatei, die alle aufgenommenen Benutzerinteraktionen enthält. Die Dateien werden auch als IWD-Dateien bezeichnet (Interactive Workflow Description).

Backend Schnittstelle, die nur von eingewiesenen Benutzern, hier den Verwaltern, zur Administration benutzt wird.

Basisimage Ein Image, das (nur) ein Betriebssystem enthält. Wird als Grundlage für Anwendungsinstallationen genutzt. Durch eine Installation wird das Image nicht verändert, es wird eine Kopie erzeugt.

Compiler Ein Programm, dass Quellcode von einer Sprache in eine andere übersetzt. Meist von einer höheren Programmiersprache in Maschinencode (assembler) oder eine Zwischenstufe (Bytecode). Der Wortschatz der Maschinensprache ist Hardware- bzw. Plattformabhängig.

Container Synonym für Image.

Datenbank Im Kontext dieser Arbeit beschreibt dieser Begriff eine Speicherung der Systemanforderungen, Hardwareplattformen, Softwareversionen und ähnlichen Angaben. Aus der Datenbank kann man meist mehrere View-Paths ableiten. Eine solche Datenbank wird zum Beispiel im KEEP-Projekt vorgestellt.

Frontend Schnittstelle, die von den normalen Benutzern zur Interaktion benutzt wird.

Gastsystem Die emulierte Umgebung, in der Regel das proprietäre Betriebssystem, z.B. Windows 3.11 oder 9x.Synonym für „emuliertes Betriebssystem“

Hostsystem Das System, auf dem der Emulator (hier Qemu) läuft. Meistens Unixoid.

Image Abbild eines Datenträgers, z.B. ISO-, IMG-, QCOW2- Formate.

Laufzeitumgebung Software-Komponente, von der andere Anwendungen abhängig sind. Als Beispiel wäre das Java Runtime Environment anzuführen.

Metadaten Hintergrundinformationen, die von großer Bedeutung für die langfristige Reproduzierbarkeit sind. Sie werden getrennt von den eigentlichen Daten gespeichert.

Mounten Der Vorgang des Einbindens eines Datenträger(-abbild)s in ein Betriebssystem, sodass man darauf zugreifen kann.

QEMU Die in dieser Arbeit verwendete Emulationssoftware [QEMU Developers 2011].

View-Path Enthält die einzelnen Schritte vom Basisimage zur fertigen Ablaufumgebung mit allen notwendigen Anwendungen, die für das Betrachten eines bestimmten Dokumenttyps notwendig sind. Wird aus den Metadaten im Zusammenspiel mit einer Datenbank ermittelt.

VNCplay Java-Software, die das Aufnehmen und Wiedergeben von Benutzerinteraktionen durchführt [Zeldovich und Chandra 2005].

Wiedergabekomponente Komponente von VNCplay, die für die Wiedergabe zuständig ist. Spielt eine Aufzeichnung ab.

Anhang

A. Funktionsreferenz

Hier sei ein Überblick der wichtigsten, in dieser Arbeit implementierten Methoden gegeben.

A.1. VncCanvas

In dieser Klasse wurden Methoden zur benutzerfreundlicheren Steuerung der Maus, sowie die Zustandssteuerung eingebaut.

getState und setState liefern bzw. setzen den Zustand, es gibt insgesamt 6 Zustände.
mouseExited hier ist die Maus-Begrenzung implementiert.
setInputBlock setzt eine Variable, so dass die Methoden des Listeners nichts mehr tun.
hostPointerMove bewegt den Zeiger des Host-Systems zur angegebenen Position.
guestPointerMove erzeugt ein MouseEvent, welches an den Gast durchgereicht wird und dazu führt, dass sich der Gastzeiger bewegt.

A.2. VncInputPlayback

In dieser Klasse waren viele Anpassungen des bestehenden Programms notwendig, weil die Benutzerinteraktionsphase den normalen Ablauf des Playbacks komplett unterbricht.

actionPerformed verarbeitet die Timer-Ereignisse, die den Ablauf der Wiedergabe steuern, diese Methode wurde komplett überarbeitet.
processStages prüft, ob eine Stufe UserInteractionBegin oder -End vorliegt.
readNext wurde so modifiziert, dass sie mit Leerzeichen in den Hinweistexten umgehen kann.

userInteraction wechselt bei der Wiedergabe in den Interaktionsmodus und zurück, zeigt den Hinweistext an und setzt die Variablen zurück

A.3. VncInputRecorder

Hier wurde nur die Aufzeichnung der Benutzerinteraktionen eingefügt.

userInteraction leitet die Benutzerinteraktionsphase bei der Aufzeichnung ein und beendet sie. Es werden die entsprechenden Logeinträge und SyncPoints erzeugt.
newStage schreibt eine neue Stage ins Log, der Code stammt großtenteils aus [Kulzhabayev 2012].
sync-Methoden erzeugen einen SyncPoint an verschiedenen Positionen des Bildschirms (Mitte, EckeLinks-oben, Position des Mauszeigers).

A.4. VncViewer

Die Verarbeitung der Zustände wurde komplett geändert, da das ursprüngliche VNC-Play keine richtigen Zustände hatte.

pauseRecording wechselt in den Pazse-Modus (Zustand 2) bzw. BenutzerInteraktionsmodus (Zustand 3).
processCtrlAltCommands verarbeitet die Tastaturshortcuts zur Steuerung von VNC-Play.
resumeRecording wechselt aus den Zuständen 2 und 3 zurück in Zustand 1.

B. Inhalt der DVD

Auf der beigefügten DVD befindet sich der Quellcode sowie die im Verlauf dieser Arbeit eingesetzten Programme, Skripte. Auch die während der Evaluation benutzten Daten sind vorhanden.

- Java-Quellcode von VNCplay
- eine digitale Kopie dieser Arbeit
- Basisimages von Windows 3.11, 95 und 98

- Aufzeichnungen
- in der Evaluation genutzte Programme in Form von ausführbaren Dateien und als Image
- Skripte (größtenteils vom Lehrstuhl zur Verfügung gestellt)
- die verwendeten QEMU-Versionen
- Anleitung zur Installation von QEMU und zur Nutzung der Skripte

Literaturverzeichnis

Anderson u. a. 2010
Anderson, David ; Delve, Janet ; Pinchbeck, Dan: Towards a workable, emulation-based preservation strategy: rationale and technical metadata. In: *New review of information networking* (2010), Nr. 15, S. 110–131. – URL http://eprints.port.ac.uk/2705. – ISSN 1361-4576

Anderson u. a. 2009
Anderson, David ; Delve, Janet ; Pinchbeck, Dan ; Alemu, Getaneh A.: Preliminary document analyzing and summarizing metadata standards and issues across Europe / KEEP - Keeping Emulation Environments Portable. 2009. – Forschungsbericht

Caplan 2009
Caplan, Priscilla: *Understanding PREMIS*. Februar 2009. – URL http://www.loc.gov/standards/premis/understanding-premis.pdf

Farquhar und Hockx-Yu 2007
Farquhar, Adam ; Hockx-Yu, Helen: Planets: Integrated Services for Digital Preservation. In: *International Journal of Digital Curation* 2 (2007), Nr. 2. – URL http://www.ijdc.net/ijdc/article/view/46. – ISSN 1746-8256

Genev 2010
Genev, Evgeni: *VNC Interface for Java X86-Emulator Dioscuri*. Online, http://hdl.handle.net/10760/15102. Oktober 2010. – URL http://hdl.handle.net/10760/15102

van der Hoeven 2007a
Hoeven, Jeffrey van der: Dioscuri: emulator for digital preservation. In: *D-Lib Magazine* 13 (2007), Nr. 11/12. – URL http://www.dlib.org/dlib/november07/11inbrief.html#VANDERHOEVEN. – ISSN 1082-9873

van der Hoeven 2007b
Hoeven, Jeffrey van der: Emulation for Digital Preservation in Practice: The Results. In: *International Journal of Digital Curation* 2 (2007), S. 123–132

Kulzhabayev 2012
KULZHABAYEV, Alibek: *Abstract Unattended Workflow Interactions*. Masterthesis. Januar 2012. – URL http://hdl.handle.net/10760/16791

McDonough 2006
MCDONOUGH, Jerome P.: METS: Standardized Encoding for Digital Library Objects. In: *International Journal on Digital Libraries* 6 (2006), S. 148–158. – URL http://hdl.handle.net/2142/177

OPF 2011
OPF: *Open Planets Foundation*. Online, *http://www.planets-project.eu*. 2011. – URL http://www.openplanetsfoundation.org

Philipps 2010
PHILIPPS, Mario: *Entwurf und Implementierung eines Softwarearchivs für die digitale Langzeitarchivierung*, Diplomarbeit, Juli 2010. – URL http://hdl.handle.net/10760/14712

QEMU Developers 2011
QEMU DEVELOPERS: *QEMU – Open Source Processor Emulator*. Online, *http://wiki.qemu.org*. 2011. – URL http://wiki.qemu.org

Rechert u. a. 2009
RECHERT, Klaus ; SUCHODOLETZ, Dirk von ; WELTE, Randolph ; DOBBELSTEEN, Maurice van den ; ROBERTS, Bill ; HOEVEN, Jeffrey van der ; SCHRODER, Jasper: Novel Workflows for Abstract Handling of Complex Interaction Processes in Digital Preservation. In: *Proceedings of the Sixth International Conference on Preservation of Digital Objects (iPRES09)*, 2009

Rechert u. a. 2010
RECHERT, Klaus ; SUCHODOLETZ, Dirk von ; WELTE, Randolph ; RUZZOLI, Felix ; VALIZADA isgandar: Reliable Preservation of Interactive Environments and Workflows. In: LALMAS, Mounia (Hrsg.) ; JOSE, Joemon M. (Hrsg.) ; RAUBER, Andreas (Hrsg.) ; SEBASTIANI, Fabrizio (Hrsg.) ; FROMMHOLZ, Ingo (Hrsg.): *Research and Advanced Technology for Digital Libraries, 14th European Conference, ECDL 2010, Glasgow, UK, September 6-10, 2010. Proceedings* Bd. 6273, Springer, 2010, S. 494–497

Reichherzer und Brown 2006
REICHHERZER, Thomas ; BROWN, Geoffrey: Quantifying software requirements for supporting archived office documents using emulation. In: *Digital Libraries, 2006. JCDL '06. Proceedings of the 6th ACM/IEEE-CS Joint Conference on*, june 2006, S. 86–94

Rothenberg 1999
ROTHENBERG, Jeff: *Ensuring the Longevity of Digital Information.* Online, *http://www.clir.org/pubs/archives/ensuring.pdf.* 1999. – URL http://www.clir.org/pubs/archives/ensuring.pdf.

Ruzzoli 2009
RUZZOLI, Felix: *Ein Framework für die zustandsbasierte Fehlererkennung und Behandlung von interaktiven Arbeitsabläufen.* Bachelorthesis. 2009

Strodl u. a. 2007
STRODL, Stephan ; BECKER, Christoph ; NEUMAYER, Robert ; RAUBER, Andreas: How to Choose a Digital Preservation Strategy Evaluating a Preservation Planning Procedure. In: *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, 2007

von Suchodoletz 2009
SUCHODOLETZ, Dirk von: *Funktionale Langzeitarchivierung digitaler Objekte – Erfolgsbedingungen für den Einsatz von Emulationsstrategien*, Dissertation, 2009

von Suchodoletz u. a. 2010
SUCHODOLETZ, Dirk von ; RECHERT, Klaus ; WELTE, Randolph ; DOBBELSTEEN, Maurice van den ; ROBERTS, Bill ; HOEVEN, Jeffrey van der ; SCHRODER, Jasper: Automation of Flexible Migration Workflows. In: *International Journal of Digital Curation* 2 (2010), Nr. 2. – URL http://www.ijdc.net/index.php/ijdc/article/view/172/240. – ISSN 1746-8256

Tchayep 2011
TCHAYEP, Achille N.: *Emulatoren-Testing für die digitale Langzeitarchivierung.* Masterthesis. März 2011. – URL http://hdl.handle.net/10760/15549

TNA 2010
TNA, The National Archives: *The technical registry PRONOM.* Online, *http://www.nationalarchives.gov.uk/pronom.* 2010. – URL http://www.nationalarchives.gov.uk/pronom. – Online resource

Valizada 2011
VALIZADA, Isgandar: *Large-Scale, Transparent Format Migration System.* Masterthesis. 2011. – URL http://eprints.rclis.org/handle/10760/16073

Welte 2009
WELTE, Randolph: *Funktionale Langzeitarchivierung digitaler Objekte – Entwicklung eines Demonstrators zur Internet-Nutzung emulierter Ablaufumgebungen.* Südwestdeutscher Verlag für Hochschulschriften, 2009

Zeldovich und Chandra 2005

ZELDOVICH, Nickolai ; CHANDRA, Ramesh: Interactive performance measurement with VNCplay. In: *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA : USENIX Association, 2005, S. 54–64